McBits Revisited

ia.cr/2017/793

Tung Chou

Osaka University, Japan

Code-based cryptography (encryption)

Sender		Receiver
	$\vec{m} + \vec{e} = \vec{r}$	
$\vec{m}$	- - - - - - - - - - - →	$\vec{r} \neq \vec{m}$
	(noisy channel)	

Code-based cryptography (encryption)

Sender		Receiver
$\vec{c} = \vec{m}G$	$\vec{c} + \vec{e} = \vec{r}$ - - - - - - - - → (noisy channel)	$\vec{c},\ \vec{e} = \mathsf{Decode}(\vec{r})$

Code-based cryptography (encryption)

Sender | Receiver

$$\vec{r} = \vec{m}G + \vec{e} \quad \overset{\vec{r}}{\dashrightarrow} \quad \vec{c},\ \vec{e} = \mathsf{Decode}(\vec{r})$$

Code-based cryptography (encryption)

<u>Sender</u> <u>Receiver</u>

$$\vec{r} = \vec{m}G + \vec{e} \quad \xrightarrow{\vec{r}} \quad \vec{c},\ \vec{e} = \mathsf{Decode}(\vec{r})$$

- McEliece (1978) using binary Goppa code remains secure.
- Niederreiter as the dual system.
- Confidence-inspiring post-quantum cryptosystems.

The old and the new McBits

The old McBits (2013)

- "*McBits:* **Fast constant-time** *code-based cryptography*"
  by Daniel J. Bernstein, Tung Chou, Peter Schwabe
- Bitslicing, non-conventional algorithms for decoding
- Using **external** parallelism
- High throughput, high latency

The old and the new McBits

The old McBits (2013)

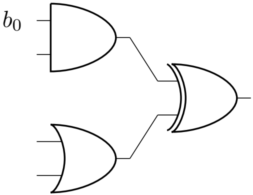
- "*McBits:* **Fast constant-time** *code-based cryptography*"
  by Daniel J. Bernstein, Tung Chou, Peter Schwabe
- Bitslicing, non-conventional algorithms for decoding
- Using **external** parallelism
- High throughput, high latency

The new McBits (2017)

- Using **internal** parallelism
- High throughput, low latency

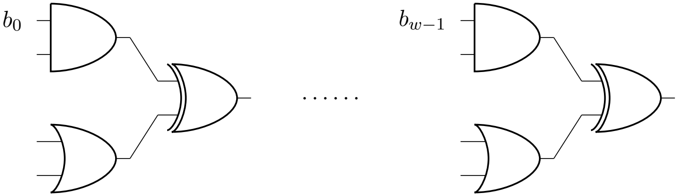
Bitslicing

"Simulating w copies of a circuit using bitwise logical operations."



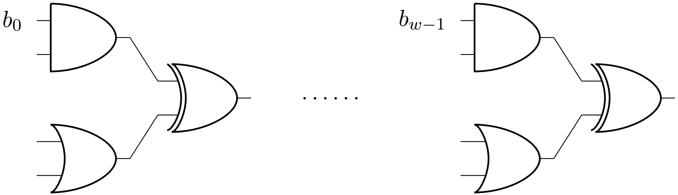
Bitslicing

"Simulating w copies of a circuit using bitwise logical operations."



Bitslicing

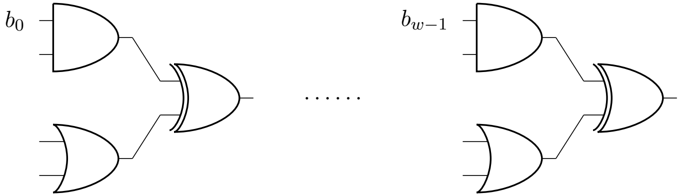
"Simulating w copies of a circuit using bitwise logical operations."



McBits 2013: Inst. 1 Inst. w

Bitslicing

"Simulating w copies of a circuit using bitwise logical operations."



McBits 2013:	Inst. 1	Inst. w
McBits 2017:	Inst. 1	Inst. 1

Speeds

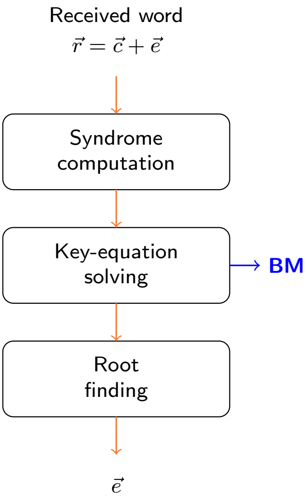
reference	m	n	t	bytes	sec	perm	synd	key eq	root	all	arch
McBits 2013	13	6624	115	958482	252	23140	83127	102337	65050	444971	IB
	13	6960	119	1046739	263	23020	83735	109805	66453	456292	IB
McBits 2017	13	8192	128	1357824	297	3783	62170	170576	53825	410132	IB
						3444	36076	127070	34491	275092	HW

Timings for decoding

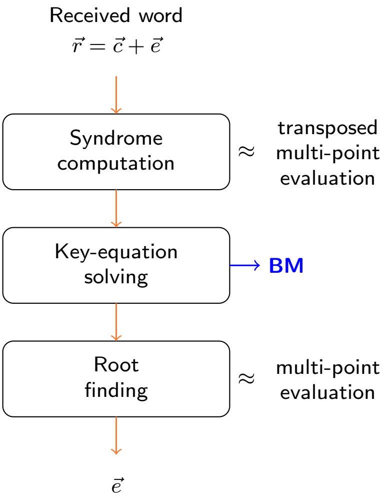
key-generation	encryption	decryption	arch
1552717680	312135	492404	IB
1236054840	289152	343344	HW

Timings for key generation, encryption, and decryption

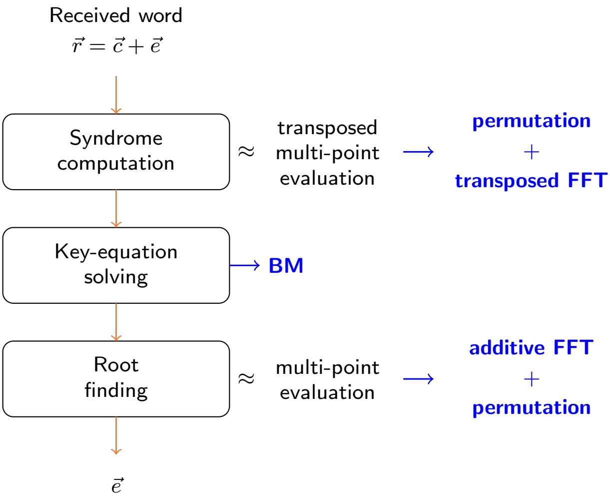
Decoder


Received word
$\vec{r} = \vec{c} + \vec{e}$
Syndrome computation
Key-equation solving
BM
Root finding
$\vec{e}$

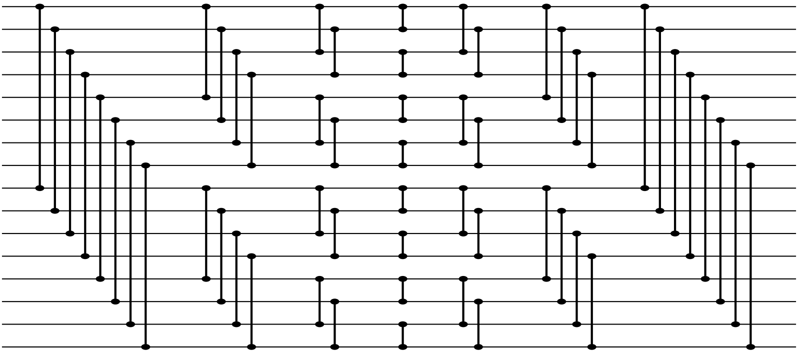
Decoder


Received word
$\vec{r} = \vec{c} + \vec{e}$
Syndrome computation
≈ transposed multi-point evaluation
Key-equation solving
→ BM
Root finding
≈ multi-point evaluation
$\vec{e}$

Decoder


Received word
$\vec{r} = \vec{c} + \vec{e}$
Syndrome computation
≈ transposed multi-point evaluation
permutation + transposed FFT
Key-equation solving
BM
Root finding
≈ multi-point evaluation
additive FFT + permutation
$\vec{e}$

Beneš network

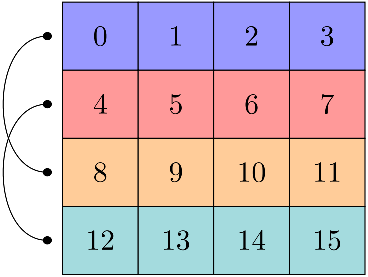


- if c, swap(b_0, b_1)
- $d \leftarrow b_0 \oplus b_1;\ d \leftarrow cd;\ b_0 \leftarrow b_0 \oplus d;\ b_1 \leftarrow b_1 \oplus d;$

Beneš network

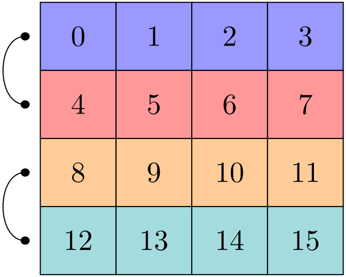
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Beneš network



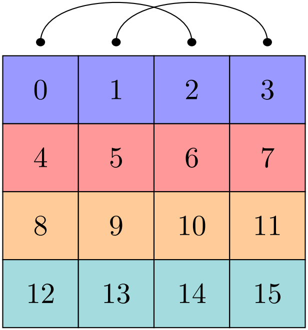
Stage 1

Beneš network



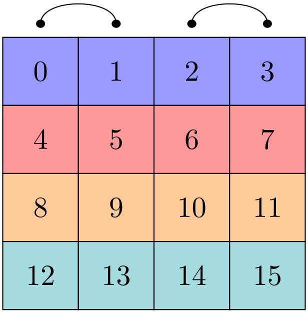
Stage 2

Beneš network



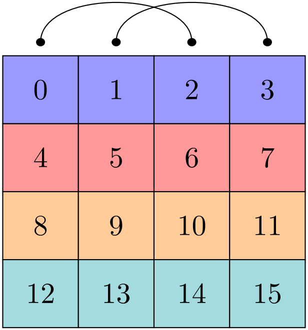
Stage 3

Beneš network



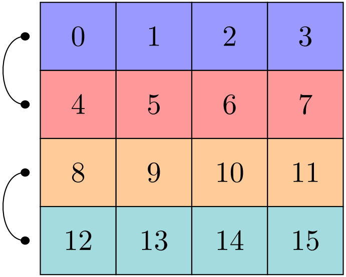
Stage 4

Beneš network



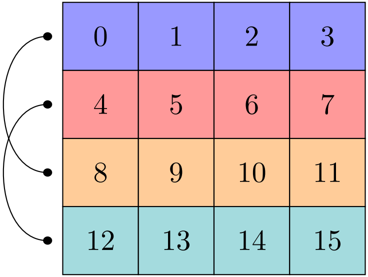
Stage 5

Beneš network



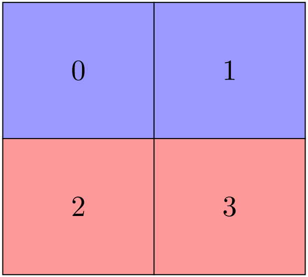
Stage 6

Beneš network

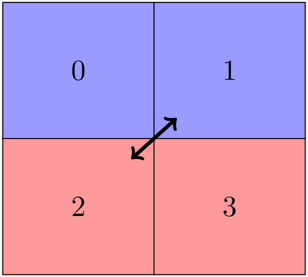


Stage 7

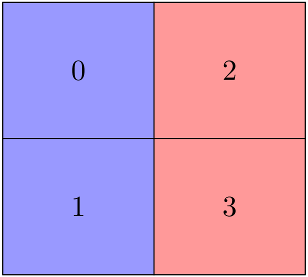
Bit-matrix transposition



Bit-matrix transposition



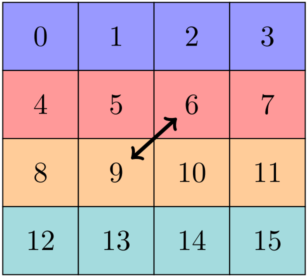
Bit-matrix transposition



Bit-matrix transposition

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

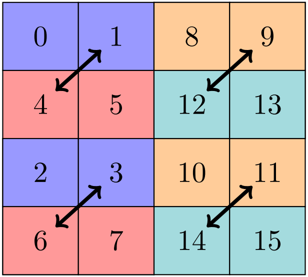
Bit-matrix transposition



Bit-matrix transposition

0	1	8	9
4	5	12	13
2	3	10	11
6	7	14	15

Bit-matrix transposition



Bit-matrix transposition

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

The Gao–Mateer Additive FFT

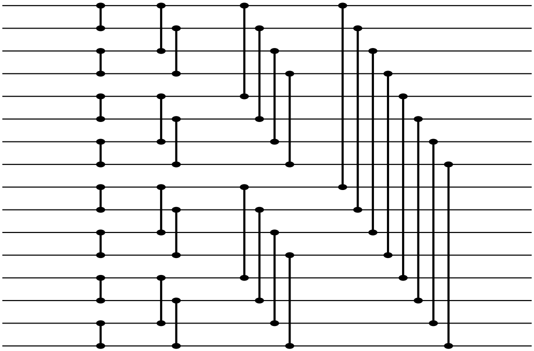
- Multiplicative FFT

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

- Additive FFT

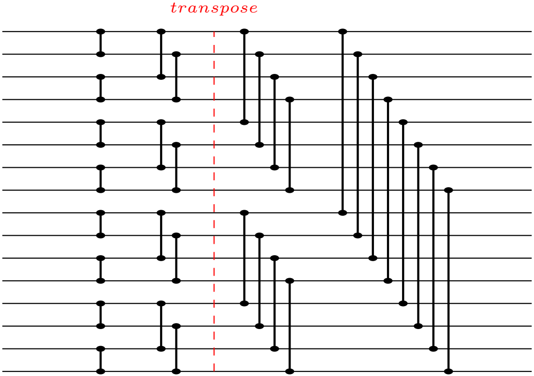
$$f(x) = f^{(0)}(x^2 + x) + xf^{(1)}(x^2 + x)$$

Additive FFT (butterflies)



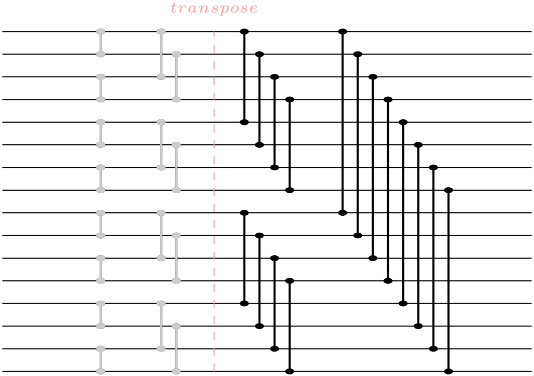
“Full” FFT

Additive FFT (butterflies)



"Full" FFT

Additive FFT (butterflies)

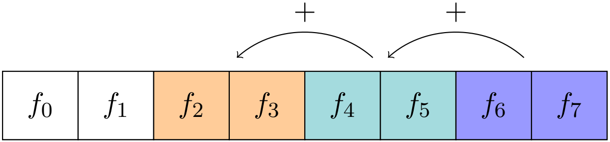


Low-degree FFT

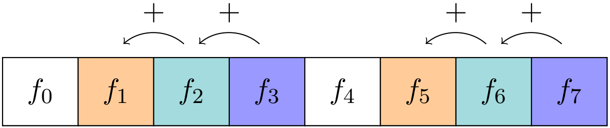
Additive FFT (radix conversions)

f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7

Additive FFT (radix conversions)



Additive FFT (radix conversions)

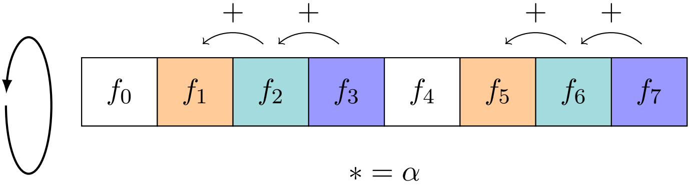


Additive FFT (radix conversions)

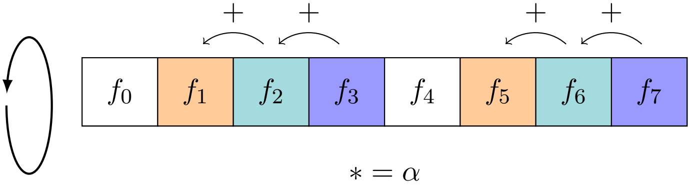
f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7

$$* = \alpha$$

Additive FFT (radix conversions)

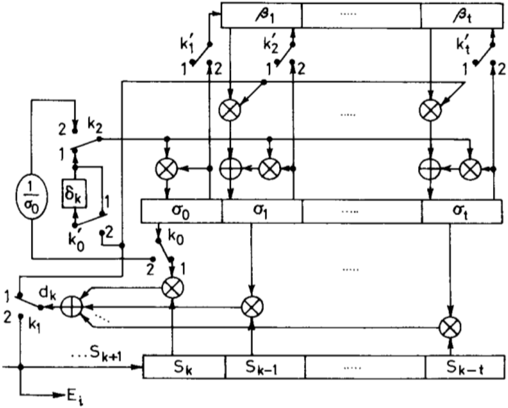


Additive FFT (radix conversions)



- Additions: logical operations $\&$, ^, $\gg$, $\ll$.
- Bitsliced multiplications.
- Small polynomial degree $\Rightarrow$ relatively cheap.

Berlekamp-Massey algorithm



Picture from:

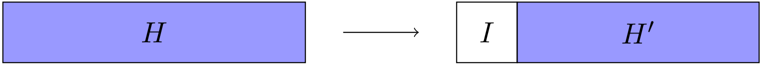
"Implementation of Berlekamp-Massey algorithm without inversion"

by Xu Youzhi

Key generation

Public-key generation

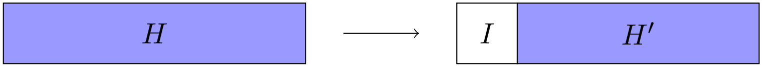
- Constant-time Gaussian elimination in $\mathbb{F}_2$.



Key generation

Public-key generation

- Constant-time Gaussian elimination in $\mathbb{F}_2$.



Secret-key generation

- Goppa polynomial: degree-t, irreducible $g \in \mathbb{F}_{2^m}[x]$.
- Generating random element $\alpha \in \mathbb{F}_{2^{mt}}$.
- Derive **minimal polynomial** of α with Gaussian elimination in $\mathbb{F}_{2^m}$.

tungchou.github.io/mcbits/