QcBits:
constant-time small-key code-based cryptography

Tung Chou

Technische Universiteit Eindhoven, The Netherlands

April 1, 2016. CWG, Utrecht, the Netherlands

Coding theory

Coding theory

Linear codes

Coding theory

Linear codes

- a linear subspace in $\mathbb{F}_2^N$

Coding theory

Linear codes

- a linear subspace in $\mathbb{F}_2^N$
- can be defined by a parity-check matrix H, e.g.,

$$C = \{c \mid Hc = 0\}$$

Coding theory

Linear codes

- a linear subspace in $\mathbb{F}_2^N$
- can be defined by a parity-check matrix H, e.g.,

$$C = \{c \mid Hc = 0\}$$

Decoding

Coding theory

Linear codes

- a linear subspace in $\mathbb{F}_2^N$
- can be defined by a parity-check matrix H, e.g.,

$$C = \{c \mid Hc = 0\}$$

Decoding

- compute e (or c) given $c + e$, where e is of weight $\leq t$

Coding theory

Linear codes

- a linear subspace in $\mathbb{F}_2^N$
- can be defined by a parity-check matrix H, e.g.,

$$C = \{c \mid Hc = 0\}$$

Decoding

- compute e (or c) given $c + e$, where e is of weight $\leq t$
- compute e given the syndrome $He = H(c + e)$

Code-based encryption

- McEliece versus Niederreiter

	plaintext	ciphertext
McEliece	c	$c + e$
Niederreiter	e	$H^* e$

Code-based encryption

- McEliece versus Niederreiter

	plaintext	ciphertext
McEliece	c	$c + e$
Niederreiter	e	$H^* e$

- General shape

McEliece/Niederreiter + **some code**

Binary-Goppa and QC-MDPC McEliece/Niederreiter

Binary-Goppa and QC-MDPC McEliece/Niederreiter

	Binary Goppa codes	QC-MDPC codes
Confidence	unbroken since 1978	unbroken since 2013

Binary-Goppa and QC-MDPC McEliece/Niederreiter

	Binary Goppa codes	QC-MDPC codes
Confidence	unbroken since 1978	unbroken since 2013
Efficiency	fast (McBits, CHES 2013)	not so fast

Binary-Goppa and QC-MDPC McEliece/Niederreiter

	Binary Goppa codes	QC-MDPC codes
Confidence	unbroken since 1978	unbroken since 2013
Efficiency	fast (McBits, CHES 2013)	not so fast
Key size	$\approx$ 100 kilobytes	$\approx$ 1 kilobyte

Timeline

2013 QC-MDPC McEliece (ISIT)

Timeline

2013 QC-MDPC McEliece (ISIT)

Bochum people felt like implementing it...

Timeline

2013	QC-MDPC McEliece (ISIT)
	Bochum people felt like implementing it...
	on FPGAs (CHES)
2014	on FPGAs, microcontrollers (PQCrypto, DATE)
2015	on Haswell CPUs (ACM-TECS)
2016	on microcontrollers (PQCrypto)

Timeline

- 2013 — QC-MDPC McEliece (ISIT)
- *Bochum people felt like implementing it...*
- on FPGAs (CHES)
- 2014 — on FPGAs, microcontrollers (PQCrypto, DATE)
- 2015 — on Haswell CPUs (ACM-TECS)
- 2016 — on microcontrollers (PQCrypto)
- **QcBits (new)**

Timeline

- 2013 — QC-MDPC McEliece (ISIT)
- *Bochum people felt like implementing it...*
- on FPGAs (CHES)
- 2014 — on FPGAs, microcontrollers (PQCrypto, DATE)
- 2015 — on Haswell CPUs (ACM-TECS)
- 2016 — on microcontrollers (PQCrypto)
- **QcBits (new)**

The problem is timing attacks.

Timeline

2013	QC-MDPC McEliece (ISIT)
	Bochum people felt like implementing it...
	on FPGAs (CHES)
2014	on FPGAs, microcontrollers (PQCrypto, DATE)
2015	on Haswell CPUs (ACM-TECS)
2016	on microcontrollers (PQCrypto)
	QcBits (new)

The problem is timing attacks.

Timeline

- 2013 — QC-MDPC McEliece (ISIT)
- *Bochum people felt like implementing it...*
- on FPGAs (CHES)
- 2014 — on FPGAs, microcontrollers (PQCrypto, DATE)
- 2015 — on Haswell CPUs (ACM-TECS)
- 2016 — on microcontrollers (PQCrypto)
- **QcBits (new)**

The problem is timing attacks.

- PQCrypto 2014: constant-time operations assuming no caches

Timeline

2013	QC-MDPC McEliece (ISIT)
	Bochum people felt like implementing it...
	on FPGAs (CHES)
2014	on FPGAs, microcontrollers (PQCrypto, DATE)
2015	on Haswell CPUs (ACM-TECS)
2016	on microcontrollers (PQCrypto)
	QcBits (new)

The problem is timing attacks.

- PQCrypto 2014: constant-time operations assuming no caches
- QcBits: constant-time for a wide-variety of 32/64-bit platforms

Cache-timing attacks

C
C
C
C
C
C

Cryptographic software overwrites some cache lines.

Cache-timing attacks

C
A
A
A
C
C

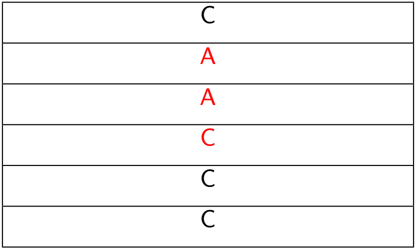
Adversarial software overwrites some cache lines.

Cache-timing attacks

C
A
A
C
C
C

Cryptographic software accesses a cache line.

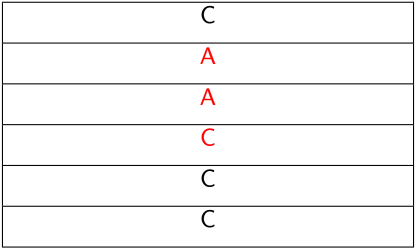
Cache-timing attacks



Cryptographic software accesses a cache line.

Adversary gains information about the index from timing.

Cache-timing attacks



Cryptographic software accesses a cache line.

Adversary gains information about the index from timing.

Solution: don't use secret memory indices.

Performance results

platform	key-pair	encrypt	decrypt	reference	scheme
Haswell	784 192	82 732	1 560 072	**(new) QcBits**	KEM/DEM
	14 234 347	34 123	3 104 624	ACMTECS 2015	McEliece
Cortex-M4	140 372 822	2 244 489	14 679 937	**(new) QcBits**	KEM/DEM
	63 185 108	2 623 432	18 416 012	PQCrypto 2016	KEM/DEM
	148 576 008	7 018 493	42 129 589	PQCrypto 2014	McEliece

Cycle counts for key-pair generation, encryption, and decryption for 80-bit pre-quantum security. Numbers in RED are non-constant-time. Numbers in BLUE are constant-time.

QC-MDPC code

QC-MDPC code

- MDPC: moderate-density-parity-check

QC-MDPC code

- MDPC: moderate-density-parity-check
- QC: quasi-cyclic (for saving bandwidth and memory)

QC-MDPC code

- MDPC: moderate-density-parity-check
- QC: quasi-cyclic (for saving bandwidth and memory)

$$\begin{pmatrix} H^{(0)} & H^{(1)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{F}_2^{n \times 2n}$$

QC-MDPC code

- MDPC: moderate-density-parity-check
- QC: quasi-cyclic (for saving bandwidth and memory)

$$\begin{pmatrix} H^{(0)} & H^{(1)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{F}_2^{n \times 2n}$$

QcBits:

- $[n = 4801, w = 90, t = 84]$ for 80-bit security

QC-MDPC code

- MDPC: moderate-density-parity-check
- QC: quasi-cyclic (for saving bandwidth and memory)

$$\begin{pmatrix} H^{(0)} & H^{(1)} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \in \mathbb{F}_2^{n \times 2n}$$

QcBits:

- $[n = 4801, w = 90, t = 84]$ for 80-bit security
- further requires $H^{(i)}$ to have row weight $w/2$
  (same for the Bochum papers)

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

+)

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$+)$

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Flip v_i if u_i is large. Repeat until $Hv = 0$.

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$+)$

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Flip v_i if u_i is large. Repeat until $Hv = 0$.

Rationale

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

+)

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Flip v_i if u_i is large. Repeat until $Hv = 0$.

Rationale

- parity$=0$: perhaps no errors. no information.

Statistical decoding

Start with finding $v = c + e$ such that $H^* v = H^* e$. Compute Hv.

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

+)

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Flip v_i if u_i is large. Repeat until $Hv = 0$.

Rationale

- parity$=0$: perhaps no errors. no information.
- parity$=1$: one score for each possible position.

Statistical decoding

High-level view

- compute the syndrome
- compute the “probability” that each position is in error
- flip the ones with “higher” probability
- repeat until success

Statistical decoding

High-level view

- compute the syndrome
- compute the “probability” that each position is in error
- flip the ones with “higher” probability
- repeat until success

Natural questions

Statistical decoding

High-level view

- compute the syndrome
- compute the "probability" that each position is in error
- flip the ones with "higher" probability
- repeat until success

Natural questions

- what do you mean by higher probability? (don't know)

Statistical decoding

High-level view

- compute the syndrome
- compute the "probability" that each position is in error
- flip the ones with "higher" probability
- repeat until success

Natural questions

- what do you mean by higher probability? (don't know)
- repeat how many times? (don't know)

Statistical decoding

High-level view

- compute the syndrome
- compute the "probability" that each position is in error
- flip the ones with "higher" probability
- repeat until success

Natural questions

- what do you mean by higher probability? (don't know)
- repeat how many times? (don't know)
- always work? (probably not)

Statistical decoding

High-level view

- compute the syndrome
- compute the "probability" that each position is in error
- flip the ones with "higher" probability
- repeat until success

Natural questions

- what do you mean by higher probability? (don't know)
- repeat how many times? (don't know)
- always work? (probably not)
- constant-time iterations?

Goppa code decoding with Berlekamp decoder

Goppa code decoding with Berlekamp decoder

- Syndrome computation

$$H = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \cdots & \alpha_n^{2t-1} \end{pmatrix}$$

Goppa code decoding with Berlekamp decoder

- Syndrome computation

$$H = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \cdots & \alpha_n^{2t-1} \end{pmatrix}$$

- Use Berlekamp-Massey to get the *error locator* $\mathscr{E}(x)$

$$(x - \alpha_i) \mid \mathscr{E}(x) \text{ iff } e_i = 1$$

Goppa code decoding with Berlekamp decoder

- Syndrome computation

$$H = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \alpha_1^2 & \alpha_2^2 & \cdots & \alpha_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_1^{2t-1} & \alpha_2^{2t-1} & \cdots & \alpha_n^{2t-1} \end{pmatrix}$$

- Use Berlekamp-Massey to get the *error locator* $\mathscr{E}(x)$

$$(x - \alpha_i) \mid \mathscr{E}(x) \text{ iff } e_i = 1$$

- Root finding

Statistical decoding: naive approach

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

Statistical decoding: naive approach

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$+)$

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Statistical decoding: naive approach

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

$+)$

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Step 1 computing the syndrome: $O(n^2)$

Statistical decoding: naive approach

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

+)

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Step 1 computing the syndrome: $O(n^2)$

Step 2 computing the unsatisfied parity checks: $O(n^2)$

Statistical decoding: naive approach

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} v = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}$$

+)

$$u = \begin{pmatrix} 2 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 2 \end{pmatrix} \in \mathbb{Z}^{2n}$$

Step 1 computing the syndrome: $O(n^2)$

Step 2 computing the unsatisfied parity checks: $O(n^2)$

- Bochum strategy: compute u_0, flip v_0, compute u_1, flip u_1, etc.

Syndrome computation: polynomial view

$$f, g \in \mathbb{F}_2[x]/(x^n - 1)$$

Syndrome computation: polynomial view

$$f, g \in \mathbb{F}_2[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_{n-1} & \cdots & f_1 & g_0 & g_{n-1} & \cdots & g_1 \\ f_1 & f_0 & \cdots & f_2 & g_1 & g_0 & \cdots & g_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_{n-1} & f_{n-2} & \cdots & f_0 & g_{n-1} & g_{n-2} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2n-1} \end{pmatrix} = s$$

Syndrome computation: polynomial view

$$f, g \in \mathbb{F}_2[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_{n-1} & \cdots & f_1 & g_0 & g_{n-1} & \cdots & g_1 \\ f_1 & f_0 & \cdots & f_2 & g_1 & g_0 & \cdots & g_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_{n-1} & f_{n-2} & \cdots & f_0 & g_{n-1} & g_{n-2} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2n-1} \end{pmatrix} = s$$

$$\downarrow$$

$$\begin{pmatrix} f & xf & \cdots & x^{n-1}f & g & xg & \cdots & x^{n-1}g \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2n-1} \end{pmatrix} = s$$

Syndrome computation: polynomial view

$$f, g \in \mathbb{F}_2[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_{n-1} & \cdots & f_1 & g_0 & g_{n-1} & \cdots & g_1 \\ f_1 & f_0 & \cdots & f_2 & g_1 & g_0 & \cdots & g_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_{n-1} & f_{n-2} & \cdots & f_0 & g_{n-1} & g_{n-2} & \cdots & g_0 \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2n-1} \end{pmatrix} = s$$

$$\downarrow$$

$$\begin{pmatrix} f & xf & \cdots & x^{n-1}f & g & xg & \cdots & x^{n-1}g \end{pmatrix} \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2n-1} \end{pmatrix} = s$$

$$\downarrow$$

$$s = v^{(0)} f + v^{(1)} g \in \mathbb{F}_2[x]/(x^n - 1)$$

Sparse-times-dense polynomial in $\mathbb{F}_2[x]/(x^n - 1)$

Compute $vf \in \mathbb{F}_2[x]/(x^n - 1)$

- v dense, represented as b-bit words (typically $b = 32/64$)
- f sparse, represented as $I_f = \{i \mid f_i = 1\}$

Sparse-times-dense polynomial in $\mathbb{F}_2[x]/(x^n - 1)$

Compute $vf \in \mathbb{F}_2[x]/(x^n - 1)$

- v dense, represented as b-bit words (typically $b = 32/64$)
- f sparse, represented as $I_f = \{i \mid f_i = 1\}$

QcBits computes vf as

$$x^{i_1} v \ + \ x^{i_2} v \ + \ \cdots$$

- Each $x^i v$ is simply a rotation of v.

Sparse-times-dense polynomial in $\mathbb{F}_2[x]/(x^n - 1)$

Compute $vf \in \mathbb{F}_2[x]/(x^n - 1)$

- v dense, represented as b-bit words (typically $b = 32/64$)
- f sparse, represented as $I_f = \{i \mid f_i = 1\}$

QcBits computes vf as

$$x^{i_1} v + x^{i_2} v + \cdots$$

- Each $x^i v$ is simply a rotation of v.

Sparse-times-dense polynomial in $\mathbb{F}_2[x]/(x^n - 1)$

Compute $vf \in \mathbb{F}_2[x]/(x^n - 1)$

- v dense, represented as b-bit words (typically $b = 32/64$)
- f sparse, represented as $I_f = \{i \mid f_i = 1\}$

QcBits computes vf as

$$x^{i_1} v \; + \; x^{i_2} v \; + \; \cdots$$

- Each $x^i v$ is simply a rotation of v.
- Addition can be carried out using XOR instrctions.

Sparse-times-dense polynomial in $\mathbb{F}_2[x]/(x^n - 1)$

Compute $vf \in \mathbb{F}_2[x]/(x^n - 1)$

- v dense, represented as b-bit words (typically $b = 32/64$)
- f sparse, represented as $I_f = \{i \mid f_i = 1\}$

QcBits computes vf as

$$x^{i_1} v + x^{i_2} v + \cdots$$

- Each $x^i v$ is simply a rotation of v.
- Addition can be carried out using XOR instrctions.
- Constant-time rotations?

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \ldots i_0)_2$ bits:

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

	00000000_2	01010101_2	00110011_2	00001111_2	11110000_2

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

	00000000_2	01010101_2	00110011_2	00001111_2	11110000_2
010011_2	01010101_2	00110011_2	00001111_2	11110000_2	00000000_2

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

	00000000_2	01010101_2	00110011_2	00001111_2	11110000_2
010011_2	01010101_2	00110011_2	00001111_2	11110000_2	00000000_2
010011_2	00001111_2	11110000_2	00000000_2	01010101_2	00110011_2

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \dots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

	00000000_2	01010101_2	00110011_2	00001111_2	11110000_2
010011_2	01010101_2	00110011_2	00001111_2	11110000_2	00000000_2
010011_2	00001111_2	11110000_2	00000000_2	01010101_2	00110011_2
010011_2	00110011_2	00001111_2	11110000_2	00000000_2	01010101_2

Barrel Shifter

Rotating by $i = (i_k i_{k-1} \ldots i_0)_2$ bits:

- conditionally rotate by 2^k bits.
- conditionally rotate by 2^{k-1} bits, and so on.
- Example for $i = 010011_2$ and polynomial

$$(x^8 + x^{10} + x^{12} + x^{14}) + (x^{16} + x^{17} + x^{20} + x^{21}) + (x^{24} + x^{25} + x^{26} + x^{27}) + (x^{36} + x^{37} + x^{38} + x^{39})$$

	00000000_2	01010101_2	00110011_2	00001111_2	11110000_2
010011_2	01010101_2	00110011_2	00001111_2	11110000_2	00000000_2
010011_2	00001111_2	11110000_2	00000000_2	01010101_2	00110011_2
010011_2	00110011_2	00001111_2	11110000_2	00000000_2	01010101_2
010011_2	01100001_2	11111110_2	00000000_2	00001010_2	10100110_2

Computing u: polynomial view

$$f, g \in \mathbb{Z}[x]/(x^n - 1)$$

Computing u: polynomial view

$$f, g \in \mathbb{Z}[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_1 & \cdots & f_{n-1} & g_0 & g_1 & \cdots & g_{n-1} \\ f_{n-1} & f_0 & \cdots & f_{n-2} & g_{n-1} & g_0 & \cdots & g_{n-2} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_1 & f_2 & \cdots & f_0 & g_1 & g_2 & \cdots & g_0 \end{pmatrix} v = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix}$$

Computing u: polynomial view

$$f, g \in \mathbb{Z}[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_1 & \cdots & f_{n-1} & g_0 & g_1 & \cdots & g_{n-1} \\ f_{n-1} & f_0 & \cdots & f_{n-2} & g_{n-1} & g_0 & \cdots & g_{n-2} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_1 & f_2 & \cdots & f_0 & g_1 & g_2 & \cdots & g_0 \end{pmatrix} v = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix}$$

$$\downarrow$$

$$\begin{pmatrix} f & g \\ xf & xg \\ \vdots & \\ x^{n-1}f & x^{n-1}g \end{pmatrix} v = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix}$$

Computing u: polynomial view

$$f, g \in \mathbb{Z}[x]/(x^n - 1)$$

$$\downarrow$$

$$\begin{pmatrix} f_0 & f_1 & \cdots & f_{n-1} & g_0 & g_1 & \cdots & g_{n-1} \\ f_{n-1} & f_0 & \cdots & f_{n-2} & g_{n-1} & g_0 & \cdots & g_{n-2} \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots \\ f_1 & f_2 & \cdots & f_0 & g_1 & g_2 & \cdots & g_0 \end{pmatrix} v = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix}$$

$$\downarrow$$

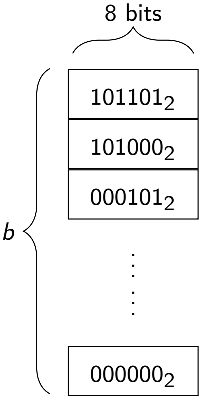
$$\begin{pmatrix} f & g \\ xf & xg \\ \vdots & \\ x^{n-1} f & x^{n-1} g \end{pmatrix} v = \begin{pmatrix} s_0 \\ s_1 \\ \vdots \\ s_{n-1} \end{pmatrix}$$

$$\downarrow$$

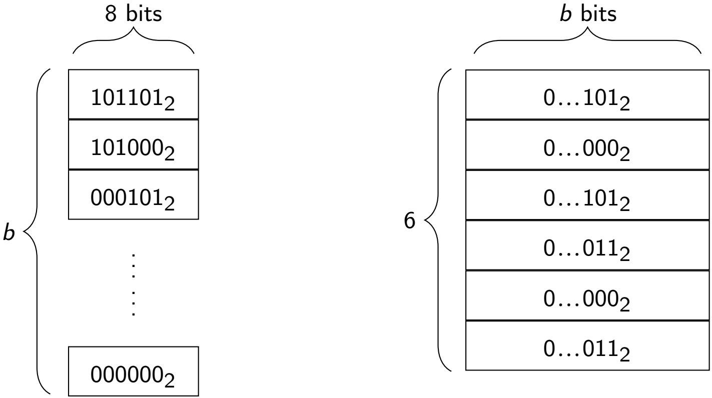
$$u = (sf,\ sg) \in \mathbb{Z}[x]/(x^n - 1)$$

Accumulating $x^i s$

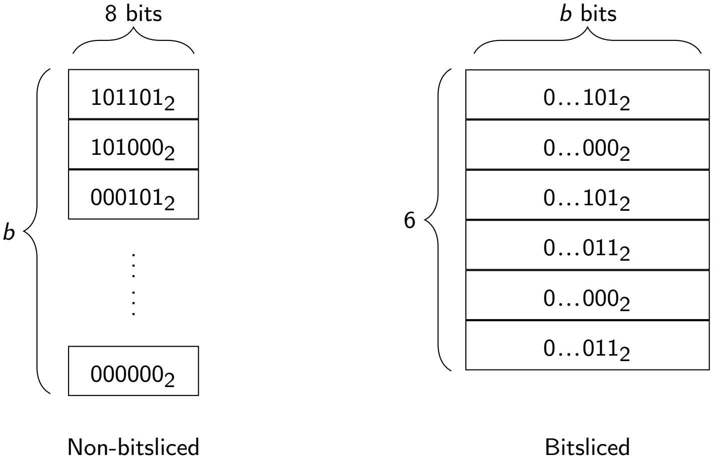
Accumulating x^is



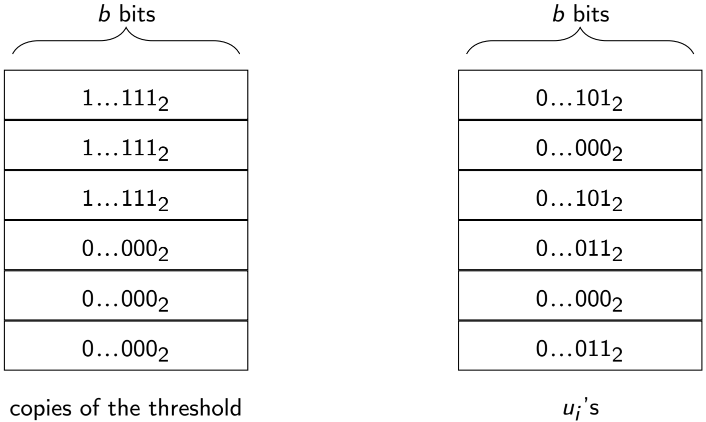
Accumulating $x^i s$



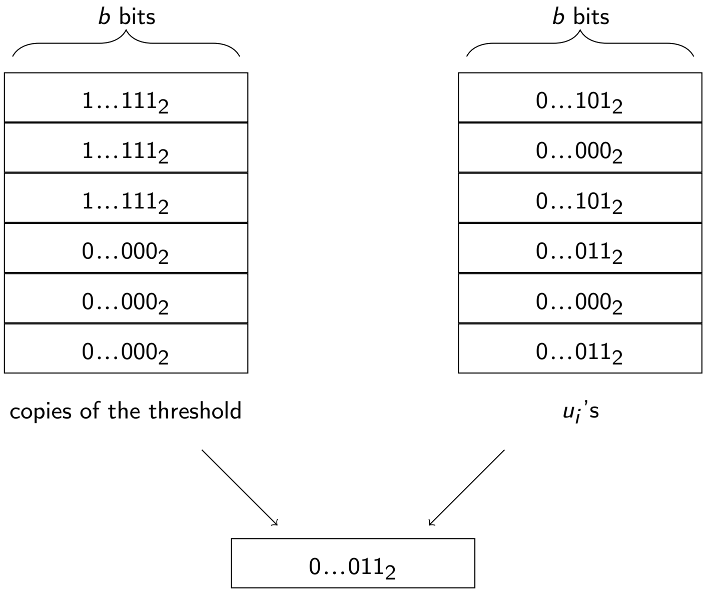
Accumulating x^is



Flipping bits



Flipping bits



Complexity

Syndrome computation

- sparse-times-dense multiplication in $\mathbb{F}_2[x]/(x^n - 1)$
- complexity: $O(wn \lg n)$

Complexity

Syndrome computation

- sparse-times-dense multiplication in $\mathbb{F}_2[x]/(x^n - 1)$
- complexity: $O(wn \lg n)$

Computing the vector u

- sparse-times-dense multiplication in $\mathbb{Z}[x]/(x^n - 1)$
- complexity: $O(wn \lg n)$

Encryption

$$H^* e = \begin{pmatrix} \mathbf{I} & & H^{(1)} \end{pmatrix} e = s$$

Encryption

$$H^* e = \begin{pmatrix} \mathbf{I} & H^{(1)} \end{pmatrix} e = s$$

$$\downarrow$$

$$\begin{pmatrix} 1 & x & \ldots & x^{n-1} & h & xh & \ldots & x^{n-1}h \end{pmatrix} \begin{pmatrix} e_0 \\ \vdots \\ e_{n-1} \\ e_n \\ \vdots \\ e_{2n-1} \end{pmatrix} = s$$

Encryption

$$H^* e = \begin{pmatrix} \mathbf{I} & & H^{(1)} \end{pmatrix} e = s$$

$$\downarrow$$

$$\begin{pmatrix} 1 & x & \dots & x^{n-1} & h & xh & \dots & x^{n-1}h \end{pmatrix} \begin{pmatrix} e_0 \\ \vdots \\ e_{n-1} \\ e_n \\ \vdots \\ e_{2n-1} \end{pmatrix} = s$$

$$\downarrow$$

$$s = e^{(0)} + he^{(1)} \in \mathbb{F}_2[x]/(x^n - 1)$$

The future of QC-MDPC McEliece/Niederreiter

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?
- better decoder?

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?
- better decoder?
- better parameters?

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?
- better decoder?
- better parameters?

Is equal weight distribution ok?

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?
- better decoder?
- better parameters?

Is equal weight distribution ok?

- at least it's close enough to the original proposal

The future of QC-MDPC McEliece/Niederreiter

How to deal with decoding failures?

- QcBits for higher security levels?
- better decoder?
- better parameters?

Is equal weight distribution ok?

- at least it's close enough to the original proposal

More research is required to build up confidence.

www.win.tue.nl/~tchou/qcbits/