LEDAkem/LEDApkc

Marco Baldi[▲], Alessandro Barenghi[■], Franco Chiaraluce[▲], Gerardo Pelosi[■], Paolo Santini[▲]

[▲]Università Politecnica delle Marche [■]Politecnico di Milano

April 13, 2018

Outline

Two proposals

- LEDAkem (Low-dEnsity parity-check coDe-bAsed key encapsulation mechanism)
  - IND-CPA key encapsulation mechanism, built on Niederreiter cryptosystem
- LEDApkc (Low-dEnsity parity-check coDe-bAsed public-key cryptosystem)
  - IND-CCA2 public-key cryptosystem, built on McEliece + Kobara-Imai Conversion

Underlying hard problems

General binary code decoding problem

- Given a $k \times n$ *random* binary matrix $\mathbf{G}$ and a n-bit vector $\tilde{c} = c + e, wt(e) < t$, find c. Proven to be NP-Complete.

Syndrome decoding problem

- Given an $r \times n$ *random* binary matrix $\mathbf{H}$ and a r-bit vector s, find the (unique) n bit vector e s.t. $\mathbf{H}e^T = s, wt(e) < t$. Proven to be NP-Complete.

Quasi-Cyclic Low-Density Parity-Check codes (QC-LDPC)

- Proposed in 2008 as a code family to instantiate McEliece/Niederreiter
- Low-Density Parity-Check: Secret code representation is a sparse matrix
  - + Small size for private keys
  - + Efficient representation/arithmetics during decoding
  - – Parameter design must not allow to guess codewords
- Quasi-cyclic: $\mathbf{H}$ and $\mathbf{G}$ constituent blocks are circulant, hence fully defined by their first row
  - + Smaller public keys
  - + Reduction in arithmetic complexity in encoding/keygen

LEDAkem

Key Generation

1. Generate a random $r \times n$ binary block circulant matrix $\mathbf{H} = [\mathbf{H}_0, \ldots, \mathbf{H}_{n_0-1}]$ made of n_0 circulant blocks, each with column weight $d_v \ll n$, $n = n_0 p$, p prime
2. Generate a random, non-singular, $n \times n$ binary block circulant matrix $\mathbf{Q}$ made of $n_0 \times n_0$ circulant blocks, with total column weight $m \ll n$
3. Store private key: $\mathbf{H}, \mathbf{Q}$
4. Compute $\mathbf{L} = \mathbf{HQ} = [\mathbf{L}_0, \ldots, \mathbf{L}_{n_0-1}]$
5. Store public key: $\mathbf{M} = (\mathbf{L}_{n_0-1})^{-1}[\mathbf{L}_0, \ldots, \mathbf{L}_{n_0-2}]$

LEDAkem

Key Encapsulation

1. Generate a random n-bit error vector $\mathbf{e}$ with weight t
2. Compute the ciphertext (syndrome) $\mathbf{s} = \mathbf{M}\mathbf{e}^T$
3. Derive the shared secret $\mathbf{x} = \mathrm{KDF}(\mathbf{e})$

Key Decapsulation

1. Obtain $\mathbf{e}$ as $\textsc{Q-Decoder}(\mathbf{s}, \mathbf{H}, \mathbf{Q})$
    - $\textsc{Q-Decoder}$ exploits the fact that the parity matrix is built as $\mathbf{HQ}$
2. Derive the shared secret $\mathbf{x} = \mathrm{KDF}(\mathbf{e})$

LEDApkc

- Built as a McEliece cryptosystem based on QC-LDPC codes
- Employs conversion by Kobara and Imai to achieve IND-CCA2 and allow using a systematic generator matrix $\mathbf{G}$
  - + Reduces the size of the public key
  - + Speeds up the encryption process overall (K-I conversion is less computationally expensive than encoding with a non-systematic $\mathbf{G}$)
- Decoding done via efficient syndrome decoding taking into account the matrix $\mathbf{Q}$ (reuse decoder from LEDAkem)
  - + Saves object code size/silicon area in implementations

Parameter sizing

Parameter design strategy

- Prevent message recovery attacks.
  - Choice of the number of errors t, code size n and rate $\frac{k}{n}$ such that ISD of the public code is not feasible.
- Prevent key recovery ("structural") attacks.
  - Density of **HQ** sufficiently high that retrieving a low-weight codeword of the dual code is not feasible.
- Provide a good DFR (hinder reaction attacks against LEDApkc).
  - n large enough to provide a satisfactory DFR ($\leq 10^{-8}$).
- Parameter design was done conservatively, targeting 2^{λ}, $\lambda \in \{128, 192, 256\}$, taking into account attackers provided with quantum computers.
- Ephemeral keys for LEDAkem, keys reusable up to $10^4 DFR^{-1}$ for LEDApkc.

Proposed parameters for LEDAkem/LEDApkc

λ	n_0	p	d_v	m	t	DFR	Size Kpub (B)	Size Kpri (B)	Size Kpri (at rest) (B)
128	2	27,779	17	7	224	$\approx 8.3 \cdot 10^{-9}$	3,480	668	24
	3	18,701	19	7	141	$\lesssim 10^{-9}$	4,688	844	24
	4	17,027	21	7	112	$\lesssim 10^{-9}$	6,408	1,036	24
192	2	57,557	17	11	349	$\lesssim 10^{-9}$	7,200	972	32
	3	41,507	19	11	220	$\lesssim 10^{-9}$	10,384	1,196	32
	4	35,027	17	13	175	$\lesssim 10^{-9}$	13,152	1,364	32
256	2	99,053	19	13	474	$\lesssim 5.8 \cdot 10^{-8}$	12,384	1,244	40
	3	72,019	19	15	301	$\lesssim 5.8 \cdot 10^{-8}$	18,016	1,548	40
	4	60,509	23	13	239	$\lesssim 5.8 \cdot 10^{-8}$	22,704	1,772	40

Efficient implementation

Circulant matrix representation/arithmetics

- Represent circulant blocks as elements of $\mathbb{F}_2[x]/\langle x^p + 1\rangle$
  - Reduces both time and space complexity for arithmetics
  - Bit packed representation for dense polynomials, sparse for sparse ones
- High sparsity of $\mathbf{H}$ and $\mathbf{Q}$ yields small (cache friendly) working set

Removed non-singularity check for $\mathbf{Q}$

- $ord_2(p) = p - 1$, $\mathrm{Perm}(wt(\mathbf{Q}))$ is odd and $< p \Rightarrow \mathbf{Q}$ is non-singular

Possible further optimizations

- Sub-quadratic polynomial multiplication
- Good fit for x86-64/Aarch64 ISA extensions (e.g. CLMUL/vector units).

Running times for LEDAkem

Portable C99 implementation, on `x86-64 nocona` gcc target (no HW `popcnt,pclmul*`)

Category	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)
1	2	45.91 ($\pm$ 0.95)	1.94 ($\pm$ 0.09)	21.69 ($\pm$ 1.39)
	3	24.70 ($\pm$ 0.44)	2.13 ($\pm$ 0.09)	25.34 ($\pm$ 2.00)
	4	22.55 ($\pm$ 0.30)	2.72 ($\pm$ 0.12)	27.24 ($\pm$ 1.77)
2–3	2	215.35 ($\pm$ 3.42)	8.61 ($\pm$ 0.28)	61.74 ($\pm$ 4.95)
	3	118.93 ($\pm$ 1.57)	9.09 ($\pm$ 0.23)	54.12 ($\pm$ 1.79)
	4	90.74 ($\pm$ 1.12)	9.83 ($\pm$ 0.20)	56.79 ($\pm$ 2.21)
4–5	2	651.58 ($\pm$ 5.81)	24.18 ($\pm$ 0.61)	109.85 ($\pm$ 6.75)
	3	354.45 ($\pm$ 5.72)	25.95 ($\pm$ 0.91)	112.36 ($\pm$ 3.48)
	4	257.84 ($\pm$ 2.97)	27.44 ($\pm$ 0.38)	149.93 ($\pm$ 4.65)

Thanks for the attention

Questions?

https://www.ledacrypt.org