POLITECNICO DI MILANO

Dipartimento di
Elettronica, Informazione
e Bioingegneria

Crittografia nell'era del calcolo quantistico: direzioni nella progettazione e realizzazione di crittosistemi

Cryptography in the quantum computation era: directions in designing and realizing cryptosystems

Alessandro Barenghi

De Cifris incontra Milano - 11th Settembre 2018

Outline

Quantum computing and modern cryptography

- Modern cryptography designed around problems assumed computationally hard in a a classical computation model
- Quantum computers provide a different computation model in terms of efficiency
- Coping with the advent of quantum computers requires modifications and re-designs of cryptographic algorithms

Quantum computers

Are they actually coming?

- R. P. Feynman's original idea of performing computation with a quantum device dates back to the 1981
- First prototype built in 1998 having 2 qubits (Mosca et al.)
- Current models have 49 (Intel), 50 (IBM), 72 (Google) qubits
- Strong drive towards larger QCs; current challenges
  - Improve coherence time of the qubits
  - Work at room temperature (instead of 20 mK)
  - Scale the design to more qubits

Impact on cryptography

Symmetric ciphers

- Lov Grover's algorithm allows to break a λ-bit key symmetric cipher in $\mathcal{O}(2^{\lambda/2})$: polynomial speedup
  - Enlarge the symmetric cipher keys by a factor of ≈ 2

Hashes

- Same as symmetric ciphers for 1st/2nd preimage, collisions for a λ-bit digest in $\mathcal{O}(2^{\lambda/3})$: polynomial speedup

Asymmetric ciphers

- Computing discrete logs and factoring are both polynomial-time on a quantum computer: exponential speedup!
  - Change of trapdoor functions needed for practical use

Too early to worry?

Perspectives and ongoing efforts

- “*There’s a 1 in 7 chance that a quantum computer able to break cryptographic algorithms will be build before 2026, a 1 in 2 chance by 2031*” – M. Mosca
- ETSI Working group on Quantum Safe Cryptography (WG-QSC)
- NIST PQ standardization effort
  - Began on Nov 2017, expected to take $3 - 5$ years
  - “*No silver bullet - each candidate has some disadvantage*”
  - “*Transition to new (public-key) algorithms in 10 years*”
  - 69 submissions, 6 withdrawn, 8 with unpatched attacks

Candidate trapdoors

Code based

- Decoding an erroneous word with a random block code
  - Good for PKE/KEM, well scrutinized (1978), quite fast

Lattice based

- Finding shortest vector in an (integer/poly) lattice
  - Possible to have PKE/KEM/Sig, “young” (’90s), fast

Hash based

- Finding a first preimage of a hash obtained as $\texttt{concat}(m, k)$
  - Sig only, acceptably fast, very well analyzed, large keys

Focus on code based

McEliece cryptosystem

- McEliece ('78): proposal of the general scheme
  - Pick a block code w/ efficient decoding
  - Obtain equivalent, random looking, representation of generator matrix (permute col.s, lin. comb. rows)
  - Encrypt encoding information and adding intentional errors
- Hardness: removing errors is NP-complete for a random code

Design choices

- Code family (e.g., Goppa, LDPC, MDPC, Hamming)
- Quasi-cyclic or non quasi-cyclic
- McEliece or Niederreiter trapdoor variant

Existing attacks

Information Set Decoding (ISD)

- Applies to all code-base based cryptosystems
- Exploits redundant information in the codeword: recovers message guessing the error-free locations
- First version proposed in 1962 by Prange as a general decoder
- Cryptosystem holds well to attacks; on practical code sizes:
  - Security margin exponent reduced by $\approx$ 35b since 1962
  - Reduction of $<$ 4b since 1988

Existing attacks

Structural attacks

- Devised against a specific code family
- Try to find and exploit non randomness in the public code representation to recover the secret representation
- Successful for some algebraic decoding code family choices (e.g. Wild McEliece) with exp. speedup
  - No effect on original Goppa codes picked by McEliece
- Successful against Low Density Parity Check codes: exploit low density in the private code
  - Can be thwarted increasing code density to prevent recovery

Proposals to the NIST PQ contest

- Proposals from [PoliMI, UnivPM]:
  - LEDAkem (Low dEnsity parity-check coDe-bAsed key encapsulation mechanism)
  - LEDApkc (Low-dEnsity parity-check coDe-bAsed public-key cryptosystem)
- Both proposals share the underlying trapdoor PKC obtained w/ IND-CCA2 construction
- Parameters tuned to have a computation effort equivalent to breaking AES on a classic/quantum computer

LEDAkem and LEDApkc design choices

- Quasi-Cyclic Low Density Parity Check (QC-LDPC) codes
  - Significantly reduced key size and highly efficient decoding during decryption
- LEDAkem relies on Niederreiter's variant of the McEliece cryptosystem
  - Given a random-looking parity matrix **H** and a syndrome vector $\mathbf{s} = \mathbf{H}\mathbf{e}^T$, find **e** , w/ $\mathit{weight}(\mathbf{e}) \leq t$
  - Problem proven to be NP-complete for a random matrix **H**
  - Less information encrypted w.r.t. McEliece (encoded in **e**), still enough for key encap
- Obtains the symmetric key employing the error vector with weight t as the input of a KDF

LEDAkem – Key Generation

Key Generation

1. Generate a random $r \times n$ binary block circulant matrix $\mathbf{H} = [\mathbf{H}_0, \ldots, \mathbf{H}_{n_0-1}]$ with column weight $d_v \ll n$
2. Generate a random, non-singular, $n \times n$ binary block circulant matrix $\mathbf{Q}$ with column weight $m \ll n$
3. Compute $\mathbf{L} = \mathbf{H} \times \mathbf{Q} = [\mathbf{L}_0, \ldots, \mathbf{L}_{n_0-1}]$
4. Private key: $\mathbf{H}, \mathbf{Q}$; Public Key $\mathbf{M} = (\mathbf{L}_{n_0-1})^{-1} \times \mathbf{L}$

LEDAkem – Encryption and Decryption

Session Key Encryption

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 $\mathbf{x} = \mathrm{KDF}(\mathbf{e})$

Session Key Decryption

1. Obtain $\mathbf{e}$ as $\textsc{Decode}(\mathbf{s}, \mathbf{H}, \mathbf{Q})$
2. Derive the shared secret $\mathbf{x} = \mathrm{KDF}(\mathbf{e})$

Running times for LEDAkem

Table: Running times for the reference portable implementation (ISO-C99, no architecture specific opt.s) on an Intel i5-6600

Security	n_0	KeyGen (ms)	Encrypt (ms)	Decrypt (ms)	Ephemeral KEM (ms)
AES-128	2	13.68 (± 0.45)	0.73 (± 0.08)	3.82 (± 0.21)	18.24
	3	4.19 (± 0.21)	0.49 (± 0.05)	6.50 (± 0.61)	11.19
	4	3.84 (± 0.21)	0.64 (± 0.04)	8.08 (± 0.64)	12.56
AES-192	2	45.58 (± 0.50)	2.07 (± 0.08)	10.53 (± 0.45)	58.19
	3	13.79 (± 0.38)	1.35 (± 0.09)	11.28 (± 0.67)	26.42
	4	13.76 (± 0.36)	1.89 (± 0.10)	19.50 (± 1.07)	35.15
AES-256	2	71.12 (± 1.35)	3.09 (± 0.13)	17.18 (± 0.60)	91.41
	3	38.83 (± 0.36)	3.45 (± 0.10)	23.77 (± 0.65)	66.07
	4	32.81 (± 0.40)	4.37 (± 0.16)	26.30 (± 1.09)	63.49

Key sizes

Table: Keypair size and encapsulated secret size for LEDAkem

Category	n_0	Private Key (B) At rest	Private Key (B) In memory	Public Key (B)	Shared secret (B)	Encap. secret (B)
AES-128	2	24	452	2,088	2,088	32
	3	24	604	2,256	1,128	32
	4	24	684	3,216	1,072	32
AES-192	2	32	644	3,832	3,832	48
	3	32	748	4,112	2,056	48
	4	32	924	6,144	2,048	48
AES-256	2	40	764	4,752	4,752	64
	3	40	988	7,008	3,504	64
	4	40	1,092	9,552	3,184	64

Thanks for the attention

Questions?

`https://www.ledacrypt.org`