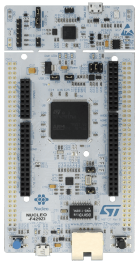
Classic McEliece Implementation with Low Memory Footprint

Johannes Roth, MTG AG (presenter)
Evangelos Karatsiolis, MTG AG
Juliane Krämer, Technische Universität Darmstadt

Classic McEliece on Embedded Devices

Parameter Set	Public Key (B)
mceliece348864	261,120
mceliece460896	524,160
mceliece6688128	1,044,992
mceliece6960119	1,047,319
mceliece8192128	1,357,824



STM32 NUCLEO-F429ZI

- ARM Cortex-M4
- RAM: 256 KB (192 KB + 64 KB)
- Flash: 2048 KB

Classic McEliece on Embedded Devices (2)

- Address the *handling* of the public key, s.t.
  - Encapsulation is more memory-efficient
  - Key pair generation is more memory-efficient
- Goal: never hold the (full) public key in memory
- Stream it!
  1. Stream from the private key
  2. Streaming encapsulation
- Work is based on round-2 reference code

Classic McEliece Key Pair Generation

1. Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2. Select a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
3. Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
4. Form an $mt \times n$ matrix $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + \cdots + c_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, \ldots, c_{m-1}$.
5. Reduce $\hat{H}$ to systematic form $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. If this fails, go back to Step 1.
6. Generate a uniform random n-bit string s.
7. Put $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ and output (s, Γ) as private key and T as public key.

Classic McEliece Key Pair Generation (2)

1. Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2. Select a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
3. Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
4. Form an $mt \times n$ matrix $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + \cdots + c_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, \ldots, c_{m-1}$.
5. Reduce $\hat{H}$ to systematic form $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. If this fails, go back to Step 1.
6. Generate a uniform random n-bit string s.
7. Put $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ and output (s, Γ) as private key and T as public key.

Private Key: $(\Gamma = (g, \alpha_1, \alpha_2, \cdots, \alpha_n), s)$

Classic McEliece Key Pair Generation (3)

1. Generate a uniform random monic irreducible polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t.
2. Select a uniform random sequence $(\alpha_1, \alpha_2, \ldots, \alpha_n)$ of n distinct elements of $\mathbb{F}_q$.
3. Compute the $t \times n$ matrix $\tilde{H} = \{h_{i,j}\}$ over $\mathbb{F}_q$, where $h_{i,j} = \alpha_j^{i-1}/g(\alpha_j)$ for $i = 1, \ldots, t$ and $j = 1, \ldots, n$.
4. Form an $mt \times n$ matrix $\hat{H}$ over $\mathbb{F}_2$ by replacing each entry $c_0 + c_1 z + \cdots + c_{m-1} z^{m-1}$ of $\tilde{H}$ with a column of t bits $c_0, c_1, \ldots, c_{m-1}$.
5. Reduce $\hat{H}$ to systematic form $(I_{n-k} \mid T)$, where I_{n-k} is an $(n-k) \times (n-k)$ identity matrix. If this fails, go back to Step 1.
6. Generate a uniform random n-bit string s.
7. Put $\Gamma = (g, \alpha_1, \alpha_2, \ldots, \alpha_n)$ and output (s, Γ) as private key and T as public key.

Public Key: $T \in \mathbb{F}_2^{(n-k) \times k}$, part of the parity check matrix $H = (I_{n-k} \mid T)$

Streaming the Public Key from the Private Key

- How to stream the public key to another party?

Streaming the Public Key from the Private Key (2)

- Approach in Classic McEliece submission
  - Perform Gaussian elimination on a parity-check matrix
    - Computes $H = (I_{n-k} \mid T)$ from $\hat{H} \in \mathbb{F}_2^{(n-k)\times n}$
    - Requires memory to hold $\hat{H}$

- Our Approach
  - Compute H by $H = S\hat{H}$
    - S is the inverse of the leftmost $n-k$ columns of $\hat{H}$
    - $S \in \mathbb{F}_2^{(n-k)\times(n-k)}$ is much smaller than $\hat{H}$
  - Computing the public key T can be done in smaller chunks
    - Note: $\hat{H}$ can be computed on-the-fly from the private key.

Streaming the Public Key from the Private Key (3)

Public Key Retrieval / Streaming

1. For $i = n - k + 1$ to n:
   2. Compute c as the ith column of $\hat{H}$ (from the private key)
   3. Compute the product Sc
   4. Send Sc and release the buffers that contain Sc and c

- Can also be done in row-major order
  - ... need to recompute $\hat{H}$ for every row

Streaming the Public Key from the Private Key (4)

- Significant computational overhead for the public key retrieval

Parameter Set	Public Key Retrieval	
	Cycles	s
mceliece348864	667,392,425	3.97
mceliece460896	2,250,917,383	13.40
mceliece6688128	5,820,127,974	34.64
mceliece8192128	7,558,882,087	44.99

Extended Private Key

- Add the matrix S to the private key
  - i.e. $(S, \Gamma = (g, \alpha_1, \alpha_2, \cdots, \alpha_n), s)$
- In exchange do not store the public key at all
- This obviously reduces the key pair size
- But how to compute S memory-efficiently?
  - Obvious approach: Gaussian elimination
  - Requires two matrices of the size of S

Extended Private Key Generation

Memory-efficient inversion of S^{-1}

1. Set S^{-1} as the leftmost $n - k$ columns of $\hat{H}$
2. Perform the LU decomposition $PS^{-1} = LU$
3. Invert L and U
4. Compute the product $U^{-1}L^{-1}$
5. Undo permutation to obtain $S = U^{-1}L^{-1}P$

- Each step can be done in-place
  - but the permutation matrix has to be stored separately (our implementation: $2(n - k)$ bytes)

Streaming Encapsulation

- Public key is used to compute the syndrome
  - $s = He = (I_{n-k} \mid T)e$, with $e \in \mathbb{F}_2^n$ a random weight-t vector
- While a public key is streamed in, all received bytes can be consumed to update s
- Example: Column-major order
  1. Generate a random error vector $e \in \mathbb{F}_2^n$
  2. Set $s := e$
  3. For $i := 1$ to k:
     4. Receive public key column c
     5. Compute $s := s + e_i c$
- A similar approach has already been described for the original McEliece scheme[1]
  - We have not seen it mentioned for Classic McEliece yet, though

[1] *Strenzke, F.: Solutions for the Storage Problem of McEliece Public and Private Keys on Memory-Constrained Platforms. In: Gollmann, D., Freiling, F.C. (eds.) Information Security. pp. 120–135. Springer Berlin Heidelberg (2012)*

Impact on Memory Requirements

Key Pair Generation / Extended Private Key Generation

- Dominated by $\hat{H}$ / S
- For reference implementation:
  - Gaussian Elimination of $\hat{H}$ to obtain H
  - $n(n-k)/8$ bytes
- For our implementation:
  - Almost-in-place inversion of matrix S^{-1}
  - $(n-k)(n-k)/8 + 2(n-k)$ bytes

Parameter Set	n	k	Ref. Impl.	Our Impl.	Difference	Ratio
mceliece348864	3488	2720	334,848	75,264	259,584	0.22
mceliece460896	4608	3360	718,848	197,184	521,664	0.27
mceliece6688128	6688	5024	1,391,104	349,440	1,041,664	0.25
mceliece8192128	8192	6528	1,703,936	349,440	1,354,496	0.21

Impact on Memory Requirements (2)

Note, this approach can also be used to stream the public key to flash memory

- Greatly reduced memory requirements of extended private key generation
- Circumvents the overhead of repeatedly retrieving the public key
- But: Consider limited write cycles of the flash (short-lived ephemeral keys)

Impact on Memory Requirements (3)

Streaming Encapsulation

- Buffer $e \in \mathbb{F}_2^n$ and the resulting syndrome $s = He \in \mathbb{F}_2^{n-k}$
- Buffer one chunk of the public key
  - Our implementation chooses to buffer chunks of 8 columns, i.e. $n - k$ bytes
  - Assumes public key is sent in column-major order
  - In principle, any order and any size for the chunks is possible

Parameter Set	n	k	Memory Overhead
mceliece348864	3488	2720	1300
mceliece460896	4608	3360	1980
mceliece6688128	6688	5024	2708
mceliece8192128	8192	6528	2896

Meaning of the Results

- Classic McEliece can be deployed with less memory than one might think, considering the public key size
  - Especially true if only the encapsulation operation is performed on a device
- The streaming encapsulation might also mitigate some DoS attacks
  - No need to allocate space for the public key
- We demonstrate the practical relevance with a proof of concept TLS implementation

TLS Proof of Concept Prototype in mbedTLS

- mbedTLS Library
  - TLS 1.2
- Use Classic McEliece as PKE to encrypt the TLS premaster secret
  - Encapsulate AES key, use AES key to encrypt TLS premaster secret
  - Ephemeral Classic McEliece key
  - The parameter set is mceliece348864
- Use SPHINCS$^+$-256f signatures
  - 49,216 bytes
  - For signing the ephemeral public key and the end-entity certificate

TLS Proof of Concept Prototype in mbedTLS (2)

- The board can handle both the server and the client side
- For completeness, we give measurements
  - however, speed was not the goal of this work
- In the following, two scenarios are considered:
  - The board is the TLS client and connects to a TLS server
  - The board is the TLS server and a TLS client connects to it
- The other party is a much faster x86 machine (Intel i5-8400)

TLS Proof of Concept Prototype in mbedTLS (3)

- Average timings for the board as a server
  - Total handshake time: 126.3 s
  - Extended private key gen: 10.83 s
  - Public key retrieval: 3.97 s
  - Decapsulation: 0.99 s
  - SPHINCS$^+$ sign: 109.71 s
- Average timings for the board as a client
  - Total handshake time: 5.83 s
  - Encapsulation: 0.018 s
  - SPHINCS$^+$ verify (x2): 5.18 s
- Not listed timings include: Network overhead, computations on x86 machine