Classic McEliece Key Generation on RAM constrained devices

Rainer Urian
rainer.urian@infineon.com

Raphael Schermann
raphael.schermann@student.tugraz.at

November 19, 2022

Abstract

Classic McEliece is a code based encryption scheme and candidate of the NIST post quantum contest. Implementing Classic McEliece on smart card chips is a challenge, because those chips have only a very limited amount of RAM. Decryption is not an issue because the cryptogram size is short and the decryption algorithm can be implemented using very few RAM. However key generation is a concern, because a large binary matrix must be inverted.

In this paper, we show how key generation can be done on smart card chips with very little RAM resources. This is accomplished by modifying the key generation algorithm and splitting it in a security critical part and a non security critical part. The security critical part can be implemented on the smart card controller. The non critical part contains the matrix inversion and will be done on a connected host.

Keywords— Classic McEliece, post quantum cryptography, smart card chips, key generation

1 Introduction

The Classic McEliece code based encryption scheme [1] is a candidate of the 4th round in the NIST post quantum contest [4]. It is one of the most trusted post quantum key agreement scheme today and it is also recommended by the German BSI [2]. Thus, for highest security demands, it would be very desirable if Classic McEliece could be used on standard security controllers, like the ones used in smart cards.

The Classic McEliece encryption scheme has very short cryptogram sizes between 128 and 240 bytes. This is smaller than RSA and even smaller than many other post quantum KEM schemes. Furthermore, decryption is very performant. The private key size is moderate and can easily be stored in Non-Volatile Memory (NVM) of a smart card controller. Although the public key size is very big, this may not be a big issue for many use cases. First, it will even completely fit on a smart card controller with big

NVM storage (i.e. 2 Mbyte). Second, there exist techniques to generate the public key on demand form the private key and some moderately sized auxiliary matrix. Third, in many use cases it is actually not necessary at all to store the public key on the chip. In such cases, one gets the public key by a host computer connected to the security chip or from a certification authority service.

The crucial issue is the generation of the public key on the security chip. For the generation a big binary matrix must be inverted. For instance, the medium size parameter set of submission [1] uses a matrix of size $1248 \times 1248 = 194688$ bytes. For performance reasons, this matrix must be stored in a RAM with adequate speed. However, RAM is expensive and standard smart card controller rarely have RAM bigger than 64k Bytes.

1.1 Our contribution

In this work we show a method how the key generation for Classic McEliece can be implemented on a security controller with less than 64k RAM. The basic idea is rather trivial: the expensive matrix inversion algorithm is outsourced to the host system connected to the security controller.

We assume the situation shown in Figure 1. This means, we have a small security controller ("Chip") with less than 64kb of RAM which is connected to a standard computer ("Host") with at least 512kb RAM. Such a system could be, for instance, a smart card connected to a terminal or a Secure Enclave (SE) chip integrated on a single die together with a more powerful host processor. The crucial security assumption is that only the Chip is trusted. Neither the Host nor the communication link will be assumed to be trusted.

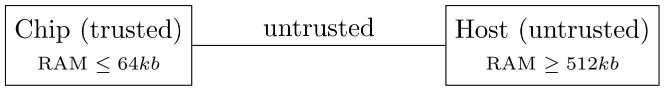

Figure 1: Trusted chip connected to an untrusted host

The method described in this paper is based on a patent application from the author and filed by Infineon Technologies AG.

1.2 Related work

Our work is mainly based on the Classic McEliece NIST submission [1]. An optimised version for the Arm Cortex M4 processor is given by Ming-Shing Chen and Tung Chou [3].

Johannes Roth, Vangelis Karatsiolis and Juliane Krmer [5] show how NVM storage space for Classic McEliece key generation can be optimised by using the LUP matrix decomposition. This method is also used in [3]. However, we do not make any used of it.

Also interesting is a somewhat older paper from 2010. Falko Strenzke [7] shows a McEliece implementation on a 16 bit controller by Infineon Technologies AG with 504 kByte of NVM and 12 kByte of RAM. He uses the original McEliece method and not the Niederreiter variant as used in the Classic McEliece submission. He describes implementations for encryption and decryption but not for key generation. The code size he uses is $n - k = 550$, which is no longer considered as appropriately secure.

1.3 Notation

For an integer m, we write $\mathbb{F}_{2^m}$ for the finite (Galois) field with 2^m elements. As we are doing explicit arithmetic, we assume that a fixed minimal polynomial f of degree m is given. An element of $\mathbb{F}_{2^m}$ is represented by a polynomial of degree less than m. We chose an vector space isomorphism $\iota : \mathbb{F}_{2^m} \simeq \mathbb{F}_2^m$ to interpret the coefficients of this polynomial as an m-dimensional vector over $\mathbb{F}_2$. This isomorphism can be extended to vectors or matrices over $\mathbb{F}_{2^m}$ in a natural way. If it is clear from the context, we always use this isomorphism implicitly without mentioning ι. We always use capital size letters to denote matrices over $\mathbb{F}_2$ (or over $\mathbb{F}_{2^m}$). We write $\mathbb{I}_n$ for the $n \times n$ identity matrix over $\mathbb{F}_2$. A linear isomorphism $\mathbb{F}_2^n \mapsto \mathbb{F}_2^n$ is written as $\mathrm{GL}_k(\mathbb{F}_2)$.

If A is a matrix, then the following notation is used to refer to its elements:

$A[i, j]$ denotes the element of A from row i and column j

$A[i, \cdot]$ denotes the vector consisting of elements from row i of A

$A[\cdot, j]$ denotes the vector consisting of elements from column j of A

If a is a vector then we write $a[i]$ or a_i for the i-th component of a.
Index numeration for a vector or matrix always start at 1.

1.4 Outline of this paper

The next section gives a short recap of the Classic McEliece algorithm from an higher abstraction level. Section 3 is the core contribution of this paper. It shows our proposed McEliece key generation method. The last chapter provides the conclusion and proposes a future work outlook.

2 Classic McEliece

First we present an overview of the Classic McEliece protocol and show its RAM usage on an optimised Arm Cortex M4 implementation.

2.1 Classic McEliece algorithm

First we present the Classic McEliece protocol. The presentation is on an abstract level and based on chapter 2 of [1].

System parameters

The system parameters of the Classic McEliece KEM consists of the following data:

m: Degree of finite field $\mathbb{F}_{2^m}$

t: Degree of the Goppa polynomial $g \in \mathbb{F}_{2^m}[x]$

k: Codesize

n: Calculated from the previous numbers as $n - k = mt$

Key generation

The key generation algorithm performs the following steps:

1. Generate private key Γ:
   (a) Generate uniformly at random a monic irreducible polynomial $g \in \mathbb{F}_{2^m}[x]$ of degree t. This is called the *Goppa Polynomial*.
   (b) Select uniformly at random n distinct elements $(\alpha_1, \dots, \alpha_n)$ from $\mathbb{F}_{2^m}$.

   The private key is then $\Gamma = \{g, (\alpha_1, \dots, \alpha_n)\}$.
2. Generate public key T:
   (a) Generate a matrix $H \in \mathbb{F}_{2^m}^{(t \times n)}$ by $H[i,j] = \alpha_j^{i-1}/g(\alpha_j)$ for $1 \leq i \leq n-k, 1 \leq j \leq n$, and interpret this matrix via the isomorphism ι as an element of $\mathbb{F}_2^{(n-k)\times n}$.
   (b) Write $H = (H_0|H_1)$, where
   $H_0[\cdot, j] = H[\cdot, j]$ with $1 \leq j \leq n-k$
   $H_1[\cdot, j] = H[\cdot, j + (n-k)]$ with $1 \leq j \leq k$
   (c) Check if H_0 is regular. If this is not the case, restart the algorithm at step 1.
   (d) Compute $S = H_0^{-1}$.
   (e) Calculate the public key $T = SH_1$.

Encryption

To encrypt a plaintext message m, the following steps have to be performed

1. Encode the message msg as a vector $e \in \mathbb{F}_2^n$ of Hamming-weight t.
2. Compute the cipher text $c_0 = Te$.

Decryption

To decrypt the cipher text $c_0 \in \mathbb{F}_2^{n-k}$, the following steps have to be performed

1. Extend c_0 to $v = (c_0, 0, \dots, 0) \in \mathbb{F}_2^n$ by appending k zeros.
2. With a syndrome decoder find the unique codeword c in the Goppa code defined by Γ that is at distance $\leq t$ from v. There are different choices for syndrome decoders. The NIST Classic McEliece proposal uses the Berlekamp-Massey algorithm.
3. Set $e = v + c$.
4. Check that e has Hamming-weight t and that $c_0 = He$. (Note that we use the non-systematic matrix H here).
5. Decode e to the message msg.

2.2 RAM footprint on Arm Cortex M4

The Github page[1] for the code to the paper [3] shows the measured stack RAM consumption of an optimised Arm Cortex M4 implementation. One can see that decryption has a RAM usage which fits into security controllers with less than 48 kByte RAM. However, key generation uses too much RAM for such devices.

[1] `https://github.com/pqcryptotw/mceliece-arm-m4`

params	key gen.	enc.	dec.
mceliece348864f	170880	1420	18500
mceliece460896f	400056	2004	34956
mceliece6688128f	581896	3996	35716
mceliece6960119f	510340	3948	35764
mceliece8192128f	588736	4116	36100

Table 1: Stack usage in bytes

3 Split key generation

Our solution to the public key generation on RAM constrained controllers is rather simple. Instead of performing the matrix inversion on the Chip by outsourcing it to the connected Host.

We do not make any security assumptions on the host. This means, the host may be totally untrusted. Therefore it is very important that no information about the private key matrix $H = (H_0|H_1)$ is leaked to the Host. This is achieved by first sending only a blinded version of the matrix H_0 to the Host. The host then performs matrix inversion on that blinded matrix and sends the blinded inverse matrix back to the Chip. By unblinding, the Chip can finally reconstruct the matrix S and construct the public key $T = SH_1$ from it.

3.1 Basic split key generation

The detailed protocol of our basic split key algorithm is shown in Figure 2.

Chip		**Host**
Generate private key Γ		
$R \xleftarrow{\$} \mathrm{GL}_{n-k}(\mathbb{F}_2)$		
$H_0[i,j] \leftarrow \alpha_j^{i-1}/g(\alpha_j)$		
$\tilde{H} \leftarrow RH_0$	$\xrightarrow{\tilde{H}}$	
		check $\tilde{H} \in \mathrm{GL}_{n-k}(\mathbb{F}_2)$
		$\tilde{S} \leftarrow \tilde{H}^{-1}$
	$\xleftarrow{\tilde{S}}$	
$S \leftarrow \tilde{S}R$		
check $SH_0 = \mathbb{I}_{n-k}$		
$H_1[i,j] \leftarrow \alpha_{j+n-k}^{i-1}/g(\alpha_{j+n-k})$		
$T \leftarrow SH_1$	$\xrightarrow{T}$	

Figure 2: Split key generation

1. The Chip

   (a) generates the private key Γ as in the Classic McEliece protocol in section 2.1, step 1 and stores it in NVM.
   (b) chooses uniformly at random an invertible matrix $R \in \mathrm{GL}_{n-k}(\mathbb{F}_2)$ and stores it in NVM.
   (c) calculates the matrix H_0 as $H_0[i,j] = \alpha_j^{i-1}/g(\alpha_j)$ for $1 \le i \le t, 1 \le j \le n-k$ and interprets this matrix via the isomorphism ι as an element of $\mathbb{F}_2^{(n-k)\times n}$.
   (d) blinds H_0 by multiplication with R.
   (e) sends the blinded matrix $\tilde{H} = RH_0$ to the host.

2. The Host
   (a) calculates $\tilde{S} = \tilde{H}^{-1}$. If the inverse does not exist it aborts and restarts the protocol at step 1.
   (b) sends $\tilde{S}$ to the Chip.
3. The Chip
   (a) unblinds the matrix inverse $\tilde{S}$ by calculating $S = \tilde{S}R$.
   (b) checks that the Host has correctly performed the inversion by checking that SH_0 is the identity matrix. If the verification fails, the Chip aborts the protocol and returns fail. Otherwise it marks the private key as valid.
   (c) reconstructs the public key. It calculates $H[i,j] = \alpha_{j+n-k}^{i-1}/g(\alpha_{j+n-k})$ for $1 \le i \le n-k$ and $1 \le j \le k$ and then multiplies with S to get the public key $T = SH_1$.

Please note that steps 1(c) - 1(e) can be done interleaved column-by-column for $\tilde{H}$ which makes temporarily storing $\tilde{H}$ in NVM unnecessary. Similarly, steps 2(b) - 3(a) can be performed interleaved row-by-row to construct S. The public key T in step 3(c) can be constructed row-by-row or column-by-column.

3.2 Correctness and security

For correctness we show that the split algorithm produces the same result as the key generation algorithm from section 2.1. The following equalities show that the matrix $S = \tilde{S}R$ produced by the split protocol is the same as the matrix $S = H_0^{-1}$ produced by protocol 2.1:

$$\tilde{S}R = \tilde{H}^{-1}R = (RH_0)^{-1}R = H_0^{-1}R^{-1}R = H_0^{-1}$$

Therefore, both algorithms produce the same systematic form of the public key. For security we must show the following two properties:

1. The blinded matrix $\tilde{H}$ does not reveal any exploitable information of the private matrix H.
2. The Chip will detect an incorrectly calculated $\tilde{S}$.

To show the first property, we note that the attacker can only exploit $\tilde{H}$ if H_0 is invertible. Otherwise the Chip would abort the protocol and choose a different H_0. The matrix $\tilde{H}$ is a random isomorphism of the invertible matrix H_0. It follows that RH_0 is indistinguishable from a random matrix of $\mathrm{GL}_{n-k}(\mathbb{F}_2)$. This shows that revealing of $\tilde{H}$ does not affect security.

To show the second property, the Chip verifies that S is the inverse of $\tilde{H}_0$ in step 3b of the split key generation protocol. Either the inverse does not exist or it is unique and therefore this check is sufficient.

3.3 RAM footprint for our key generation

As we do not have a real and optimized split key generation implementation at this time, we provide a theoretical analysis. We analyze the RAM consumption of public key generation first. The matrix R can be generated and directly stored into NVM. The matrix H_0' can be generated on-the-fly. Therefore the matrix $\tilde{H}$ can be dynamically created and send to the host column-by-column. The matrix S can be received row-by-row from the host and directly stores into NVM. This shows that RAM consumption of public key generation can be neglected.

Let's now analyse private key generation. A private key consists of two parts:

- the random permutation of elements $(\alpha_1, \ldots, \alpha_n)$ from the field $\mathbb{F}_{2^m}$
- The Goppa polynomial g

As done in [5], we will use a Fischer Yates shuffle algorithm instead of a Beneš network to perform the permutation. We assume that field elements are stored in 16 bit. Then the RAM consumption of the Fischer Yates shuffle algorithm for the fields can be calculated as $2 * 2^m$ bytes. For $m = 13$ this equates to 16384 bytes.

The implementation [3] uses a Gaussian elimination to construct the Goppa polynomial as the minimum polynomial of a random element of the extension field $\mathbb{F}_{2^{mt}}/\mathbb{F}_{2^m}$. This costs approximately $2t(t+1)$ bytes of RAM. For $t = 128$ this equates to 33024 bytes. This is a good margin below $48k$ and therefore not an issue. But we can even do better. Instead of Gaussian elimination, we can use the Berlekamp-Massey algorithm to calculate the minimum polynomial as has been shown by Victor Shoup [6]. A short analysis of this algorithm shows that it needs approximately $12 * t$ bytes of RAM, which equates to 1536 bytes for $t = 128$.

3.4 Optimised split key generation

The split key generation protocol requires the generation of a random invertible matrix $R \in \mathrm{GL}_{n-k}(\mathbb{F}_2)$. The obvious way is to generate a uniform random matrix $R \in \mathbb{F}_2^{(n-k)\times(n-k)}$ and try again if R is not invertible (i.e. the host aborts the protocol). But a random binary matrix will be invertible with a probability about 0.29 only. The probability can be calculated by the following well known result:

Fact 1. *Let A be a random matrix from $\mathbb{F}_2^{s\times t}$ (or $\mathbb{F}_2^{t\times s}$), where $s \geq t$. Then the probability that A has full column (or row) rank c is*

$$\prod_{k=1}^{t}(1 - 2^{-s+k-1})$$

The split key algorithm will fail if the matrix $\tilde{H}$ is not invertible. $\tilde{H}$ is the matrix product of R and H_0 and both matrices can be considered as independent random variables. By Fact 1, the probability that H_0 or R is invertible is about 0.29 for each. Therefore the probability that $\tilde{H}$ is invertible is only $0.29^2 = 0.084$. This means, on average approximately 12 trials are necessary until a proper key is found. This is

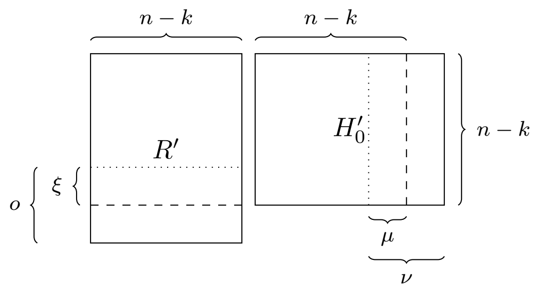

Figure 3: Matrices and parameters for optimised split key generation

not acceptable. Therefore we now show methods to significantly reduce the failure probability.

To begin with, we use an variant of the semi-systematic form of [1] for H_0. In the semi-systematic form, the matrix H_0 will be extended by a few additional columns. The resulting rectangular matrix will have full row rank $n-k$ with high probability. In a similar way, we extend the random matrix R by a few additional random rows such that the resulting rectangular matrix will have full column rank $n-k$ with high probability. It follows that the multiplied matrix $\tilde{H} = RH_0$ will have full $n-k$ rank with high probability. The host selects from this extended matrix an invertible sub-matrix $\tilde{H}_0$ of dimension $(n-k)\times(n-k)$. It inverts that matrix and sends the inverted matrix together with the sub-matrix extraction information to the chip. Now the detailed protocol follows.

Chip		**Host**
Generate private key Γ		
$R' \overset{\$}{\leftarrow} \mathbb{F}_2^{(n-k+o-\xi)\times(n-k)}$		
$H'_0[i,j] \leftarrow \alpha_j^{i-1}/g(\alpha_j)$		
$\tilde{H} \leftarrow R'H'_0$	$\xrightarrow{\tilde{H}}$	
		calculate σ_i, π_j
		$\tilde{H}'[i,j] \leftarrow \tilde{H}[\sigma_i, \pi_j]$
	$\xleftarrow{(\tilde{S},\pi,\sigma)}$	$\tilde{S} \leftarrow \tilde{H}'^{-1}$
$\alpha_i \leftarrow \alpha_{\pi_i}$		
$R[i,j] \leftarrow R'[\sigma_i, j]$		
$S \leftarrow \tilde{S}R$		
$H_0[i,j] \leftarrow \alpha_j^{i-1}/g(\alpha_j)$		
check $SH_0 = \mathbb{I}_{n-k}$		
$T \leftarrow SH_1$	$\xrightarrow{T}$	

Figure 4: Optimised split key generation

1. The Chip
   (a) generates the private key Γ as in section 3, step 1.
   (b) calculates a random matrix $R' \in_R \mathbb{F}_2^{(n-k+o-\xi)\times(n-k)}$.
   (c) calculates the matrix $H_0' \in \mathbb{F}_2^{(n-k)\times(n-k+\nu-\mu)}$ as $H_0'[i,j] = \alpha_j^{i-1}/g(\alpha_j)$ for $1 \le i \le n-k$ and $1 \le j \le n-k+\nu-\mu$.
   (d) blinds H_0' by multiplication with R', i.e. calculates $\tilde{H} = R'H_0'$.
   (e) sends the matrix $\tilde{H}$ to the Host.
2. The Host
   (a) chooses a column permutation π of the last ν columns of $\tilde{H}$, such that the first $n-k$ columns have full column rank. If that is not possible, it restarts the protocol.
   (b) choses o rows from the last ξ rows, such that the resulting $n-k$ rows have full rank $n-k$. The selection of the rows is protocolled by σ. If that is not possible, it restarts the protocol.
   (c) calculates the invertible sub matrix $\tilde{H}'$ consisting of the first $n-k$ permuted columns and the $n-k$ selected rows of $\tilde{H}$.
   (d) inverts $\tilde{S} = \tilde{H}'^{-1}$.
   (e) sends $\tilde{S}, \sigma, \pi$ to the Chip.
3. The Chip
   (a) checks if π is a permutation of $\{n-k-\mu, \dots, n-k+\nu-\mu\}$. If this is not the case, it aborts the algorithm. Otherwise, it updates the permutation α_i of its private key by calculating $\alpha_i \leftarrow \alpha_{\pi_i}$, for $n-k-\mu \le i \le n-k+\nu-\mu$.
   (b) checks if σ is a selection of ξ elements from $\{n-k-\xi, \dots, n-k+o-\xi\}$. If this is not the case, it aborts the algorithm.
   (c) calculates the random matrix $R[\sigma_i, j] = R'[i,j]$ for $1 \le i, j \le n-k$.
   (d) unblinds the matrix $\tilde{S}$ by calculating $S = \tilde{S}R$.
   (e) calculates $H_0[i,j] = \alpha_j^i/g(\alpha_j)$ for $1 \le i, j \le n-k$.
   (f) checks if the Host has correctly performed the inversion by checking that SH_0 is the identity matrix $\mathbb{I}_{n-k}$. If the verification fails, the Chip aborts the protocol. Otherwise it marks the private key as valid.
   (g) reconstructs the public key by calculating $H_1[i,j] = \alpha_{j+n-k}^{i-1}/g(\alpha_{j+n-k})$ for $1 \le i \le n-k$ and $1 \le j \le k$ and then the public key $T = SH_1$.

Please note that steps 1(b) - 1(e) can be done interleaved column-by-column for $\tilde{H}$ which makes storing $\tilde{H}$ in NVM unnecessary. Similarly, steps 2(e) and 3(c) can be performed interleaved row-by-row to construct S. The public key T in step 3(f) can be constructed row-by-row or column-by-column.

By using Fact 1, we calculate the success probability of the protocol, i.e. the probability that the matrix $\tilde{H}$ has rank $n-k$. If we assume that the matrices R' and H_0' are statistically independent, then the probability that the matrix $\tilde{H} = R'H_0'$ has rank $n-k$ can be calculated as the product of the four following probabilities:

- probability that a $(n-k) \times (n-k-\mu)$ random matrix has rank $n-k-\mu$

- probability that a $\mu \times (\nu - \mu)$ random matrix has rank $\nu - \mu$
- probability that a $(n - k - \xi) \times (n - k)$ random matrix has rank $n - k - \xi$
- probability that a $(o - \xi) \times o$ random matrix has rank $o - \xi$

One can check that for all semi-systematic McEliece parameters from the NIST submission and with $o = 64$ and $\xi = 32$, the success probability is greater than $1 - 10^{-9}$.

3.5 Correctness and security of the optimised algorithm

Any admissible column permutation π defines a new private key H, where the first $n - k$ sub-matrix H_0 is invertible. Similarly, any admissible row selection σ defines a new random matrix R which is invertible. It follows that we are again in the setting of the protocol from Figure 2. This shows correctness of the protocol.

For security, we first show that any choice of σ and π leads to equivalent encryption schemes. Any admissible selection function σ corresponds to an invertible random matrix R_σ. All those matrices are constructed from independent random rows. Thus, it doesn't matter which rows are chosen and all those matrices are equivalent from a security perspective.

Now we show that prescribing the additional permutation π gives the attacker no advantage. Let a McEliece public key T be given. Any other admissible permutation π must permute the columns of $(\mathbb{I}_{n-k}|T)$ such that the first $n - k$ columns have full rank. It follows that the permutation π induces a $n \times n$ permutation matrix P and a $(n-k) \times (n-k)$ invertible matrix S, such that the resulting public key T' is determined by $(\mathbb{I}_{n-k}|T') = S(\mathbb{I}_{n-k}|T)P$.

Any encryption equation $c = Te$ for the unprimed system can be transferred to an encryption equation in the primed system by using $c' = S^{-1}c$ and $e' = Pe$. This shows that both systems are security equivalent and therefore the security is independent from the attacker's choice of π.

It remains to show that the attacker does not gain additional information by receiving the $\nu - \mu$ additional rows of $\tilde{H}$. As $\tilde{H}$ has row rank $n - k$, we can chose $n - k$ linearly independent rows. The remaining $\nu - \mu$ rows of $\tilde{H}$ will then be random linear combinations of those $n - k$ independent rows. Therefore, those additional rows will not give the attacker any additional information.

3.6 Pseudo random matrix R

It will be preferable to use a pseudo random number generator to generate R and R'. This avoids storing the whole matrix R' in NVM. Only a short seed value has to be stored from which R and R' can be generated on-the-fly.

3.7 Semi trusted host

Although the host is never trusted to protect the private key, it may be trusted to correctly produce and store the public key. This will be called a semi-trusted host. In that situation the chip doesn't have to construct and store the matrix S. Instead, the chip constructs a non-systematic public key $\tilde{T}$ by using the random matrix R and the π, σ reconstruction information from the host. The host must then bring the whole public key in systematic form by using the matrix S. This protocol is shown below in Figure 5.

Chip		**Host**
Generate private key Γ		
$R' \xleftarrow{\$} \mathbb{F}_2^{(n-k+o-\xi)\times(n-k)}$		
$H_0'[i,j] \leftarrow \alpha_j^{i-1}/g(\alpha_j)$		
$\tilde{H} \leftarrow R'H_0'$	$\xrightarrow{\tilde{H}}$	
		calculate σ_i, π_j
		$\tilde{H}'[i,j] \leftarrow \tilde{H}[\sigma_i, \pi_j]$
	$\xleftarrow{(\tilde{S},\pi,\sigma)}$	$\tilde{S} \leftarrow \tilde{H}'^{-1}$
$\alpha_i \leftarrow \alpha_{\pi_i}$		
$R[i,j] \leftarrow R'[\sigma_i, j]$		
$\tilde{T} \leftarrow RH_1$	$\xrightarrow{\tilde{T}}$	
		$T \leftarrow \tilde{S}\tilde{T}$

Figure 5: Split key generation for semi trusted host

1. The Chip
   (a) generates the private key Γ as in section 3, step 1.
   (b) calculates a random matrix $R' \in_R \mathbb{F}_2^{(n-k+o-\xi)\times(n-k)}$.
   (c) calculates the matrix $H_0' \in \mathbb{F}_2^{(n-k)\times(n-k+\nu-\mu)}$ as $H_0'[i,j] = \alpha_i^j/g(\alpha_i)$ for $1 \leq i \leq n-k$ and $1 \leq j \leq n-k+\nu-\mu$.
   (d) blinds H_0' by multiplication with R', i.e. calculates $\tilde{H} = R'H_0'$.
   (e) sends the matrix $\tilde{H}$ to the Host.
2. The Host
   (a) chooses a column permutation π of the last ν columns of $\tilde{H}$, such that the first $n-k$ columns have full column rank. If that is not possible, it restarts the protocol.
   (b) choses o rows from the last ξ rows, such that the resulting $n-k$ rows have full rank $n-k$. The selection of the rows is protocolled by σ. If that is not possible, it restarts the protocol.
   (c) calculates the invertible submatrix $\tilde{H}'$ consisting of the first $n-k$ permuted columns and the $n-k$ selected rows of $\tilde{H}$.
   (d) inverts $\tilde{S} = \tilde{H}'^{-1}$.
   (e) sends σ, π to the Chip.
3. The Chip
   (a) calculates the random matrix $R[\sigma_i, j] = R'[i,j]$ for $1 \leq i, j \leq n-k$.
   (b) reconstructs the public key by calculating $H_1[i,j] = \alpha_{j+n-k}^{i-1}/g(\alpha_{j+n-k})$ for $1 \leq i \leq n-k$ and $1 \leq j \leq k$
   (c) calculates the non-systematic public key $\tilde{T} = RH_1$ and sends $\tilde{T}$ to the host.

4. The Host
    (a) reconstructs the public key by calculating $T = \tilde{S}\tilde{T}$.

Note that steps 3(a) - 3(c) can be done without using NVM and with very limited RAM usage, because R and H_1 can both be calculated on-the-fly.

4 Conclusion and future work

We have shown how keys for the Classic McEliece Post Quantum KEM can be generated on a security chip with very limited RAM resources. We have shown how to do this by outsourcing the RAM-expensive matrix inversion to the attached host. The presentation of this method was focussed on functionality and crypto-analytic security. A useful future work would be to provide countermeasures against possible side channel and fault attacks.

A proof-of-concept implementation based on the Classic McEliece NIST submission is provided for a standard PC platform. The purpose of this implementation is to show the functional separation of Chip and Host part only. It does not have a focus on performance and security. A useful future work would be to provide an implementation on a standard state of the art smart card controller.

References

[1] Classic McEliece: conservative code-based cryptography. Round 3 submission to the NIST Post-Quantum Cryptography Standardization Project, 2020. `https://classic.mceliece.org/`.

[2] BSI. TR-02102-1. https://www.bsi.bund.de, 2021.

[3] Ming-Shing Chen and Tung Chou. Classic mceliece on the arm cortex-m4, 2021. `https://tungchou.github.io/papers/cm-m4.pdf`.

[4] NIST. Post-Quantum Cryptography, Round 4 Submissions. https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4, 2022.

[5] Johannes Roth, Vangelis Karatsiolis, and Juliane Krämer. Classic mceliece implementation with low memory footprint. In *19th Smart Card Research and Advanced Application Conference (CARDIS 2020)*, 2020.

[6] Victor Shoup. Fast construction of irreducible polynomials over finite fields. *Journal of Symbolic Computation*, 17(5):371–391, 1994.

[7] Falko Strenzke. A smart card implementation of the mceliece pkc. In Pierangela Samarati, Michael Tunstall, Joachim Posegga, Konstantinos Markantonakis, and Damien Sauveron, editors, *Information Security Theory and Practices. Security and Privacy of Pervasive Systems and Smart Devices*, pages 47–59, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.