Implementation of a CCA2-secure variant of McEliece using generalized Srivastava codes

Pierre-Louis Cayrel, Gerhard Hoffmann, Edoardo Persichetti

CBC Workshop - DTU

09 May 2012

OVERVIEW

- Motivation
- Preliminaries
- The McEliece Cryptosystem
- McEliece using Generalized Srivastava Codes
- Transforming McEliece into a CCA2-secure scheme
- Implementation results

Part I

MOTIVATION

MOTIVATION

In a few years time big quantum computers might be reality.

MOTIVATION

In a few years time big quantum computers might be reality.

But then (Shor [14]):

Motivation

In a few years time big quantum computers might be reality.

But then (Shor [14]):

- RSA
- DSA
- ECC
- Diffie-Hellman key exchange
- and many others ... **not secure** !

Motivation

In a few years time big quantum computers might be reality.

But then (Shor [14]):

- RSA
- DSA
- ECC
- Diffie-Hellman key exchange
- and many others ... **not secure** !

Alternatives ?

MOTIVATION

In a few years time big quantum computers might be reality.

But then (Shor [14]):

- RSA
- DSA
- ECC
- Diffie-Hellman key exchange
- and many others ... **not secure** !

Alternatives ?

- Lattice-based cryptography (NTRU).
- Hash-based cryptography.
- Code-based cryptography (McEliece, Niederreiter).
- Multivariate quadratic equations cryptography.

McEliece Cryptosystem

McEliece: first cryptosystem using error correcting codes.

MCELIECE CRYPTOSYSTEM

McEliece: first cryptosystem using error correcting codes.

- Since 1978 unbroken in its original version for reasonable parameters.

McEliece Cryptosystem

McEliece: first cryptosystem using error correcting codes.

- Since 1978 unbroken in its original version for reasonable parameters.
- More efficient than e.g. RSA [12].

McEliece Cryptosystem

McEliece: first cryptosystem using error correcting codes.

- Since 1978 unbroken in its original version for reasonable parameters.
- More efficient than e.g. RSA [12].
- No known vulnerabilities against quantum computers.

MCELIECE CRYPTOSYSTEM

McEliece: first cryptosystem using error correcting codes.

- Since 1978 unbroken in its original version for reasonable parameters.
- More efficient than e.g. RSA [12].
- No known vulnerabilities against quantum computers.
- Drawback: relatively large keys (around 1 MByte).

McEliece Cryptosystem

McEliece: first cryptosystem using error correcting codes.

- Since 1978 unbroken in its original version for reasonable parameters.
- More efficient than e.g. RSA [12].
- No known vulnerabilities against quantum computers.
- Drawback: relatively large keys (around 1 MByte).

Goal: reduce the public key size.

Part II

PRELIMINARIES

Subfield Subcodes

Let $\mathcal{C}$ be a code of length n and dimension k over $\mathbb{F}_{2^u}$.

Consider a proper subfield $\mathbb{F}_{2^\lambda}$ of $\mathbb{F}_{2^u}$, where $u = \lambda m$.

Consider all codewords $c \in \mathcal{C}$ with coordinates in $\mathbb{F}_{2^\lambda}$, that is:

$$\boxed{\mathcal{C}|_{\mathbb{F}_{2^\lambda}} := \mathcal{C} \cap \mathbb{F}_{2^\lambda}^n} \, .$$

Then $\mathcal{C}|_{\mathbb{F}_{2^\lambda}}$ is called a subfield subcode.

SUBFIELD SUBCODES

Let $\mathcal{C}$ be a code of length n and dimension k over $\mathbb{F}_{2^u}$.

Consider a proper subfield $\mathbb{F}_{2^\lambda}$ of $\mathbb{F}_{2^u}$, where $u = \lambda m$.

Consider all codewords $c \in \mathcal{C}$ with coordinates in $\mathbb{F}_{2^\lambda}$, that is:

$$\boxed{\mathcal{C}|_{\mathbb{F}_{2^\lambda}} := \mathcal{C} \cap \mathbb{F}_{2^\lambda}^n} \,.$$

Then $\mathcal{C}|_{\mathbb{F}_{2^\lambda}}$ is called a subfield subcode.

Set from now on $q = 2^\lambda$, $m > 1$ the extension degree of $\mathbb{F}_{2^u}$ over $\mathbb{F}_{2^\lambda}$.

GENERALIZED REED-SOLOMON CODES

Let $x_0, \ldots, x_{n-1}$ be distinct (non-zero) elements of $\mathbb{F}_{q^m}$.

Let $y_0, \ldots, y_{n-1}$ be non-zero elements of $\mathbb{F}_{q^m}$.

A Generalized Reed-Solomon $GRS_k(x, y)$ code over $\mathbb{F}_{q^m}$ is the linear code having the parity-check matrix H_{GRS} defined by

$$H_{GRS} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_0 & x_1 & \ldots & x_{n-1} \\ x_0^2 & x_1^2 & \ldots & x_{n-1}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{n-k-1} & x_1^{n-k-1} & \ldots & x_{n-1}^{n-k-1} \end{bmatrix} \begin{bmatrix} y_0 & 0 & \ldots & 0 \\ 0 & y_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & y_{n-1} \end{bmatrix} \quad (1)$$

$$= vdm(x) \cdot diag(y).$$

GENERALIZED REED-SOLOMON CODES

Let $x_0, \ldots, x_{n-1}$ be distinct (non-zero) elements of $\mathbb{F}_{q^m}$.

Let $y_0, \ldots, y_{n-1}$ be non-zero elements of $\mathbb{F}_{q^m}$.

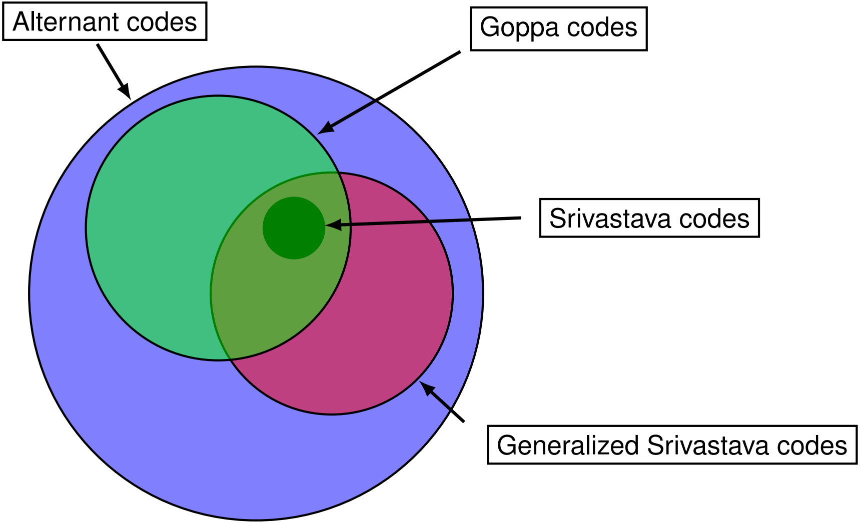
A Generalized Reed-Solomon $GRS_k(x, y)$ code over $\mathbb{F}_{q^m}$ is the linear code having the parity-check matrix H_{GRS} defined by

$$H_{GRS} = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_0 & x_1 & \ldots & x_{n-1} \\ x_0^2 & x_1^2 & \ldots & x_{n-1}^2 \\ \vdots & \vdots & \vdots & \vdots \\ x_0^{n-k-1} & x_1^{n-k-1} & \ldots & x_{n-1}^{n-k-1} \end{bmatrix} \begin{bmatrix} y_0 & 0 & \ldots & 0 \\ 0 & y_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & y_{n-1} \end{bmatrix} \quad (1)$$

$$= vdm(x) \cdot diag(y).$$

Many important codes in code-based cryptography are subfield subcodes of GRS codes:

Subfield Subcodes of GRS Codes



ALTERNANT CODES $\mathcal{A}(x, y)$

An alternant code $\mathcal{A}(x, y)$ is defined as

$$\boxed{\mathcal{A}(x, y) = GRS_k(x, y) \ \cap \ \mathbb{F}_q^n},$$

where $x, y \in \mathbb{F}_{q^m}^n$ as before.

Because $\mathcal{A}(x, y)$ is a subcode of $GRS_k(x, y)$, an alternant decoder can use the parity check matrix H_{GRS} of (1), i.e.

$$H_{GRS} = vdm(x) \cdot diag(y).$$

Part III

McEliece and variants

THE INGREDIENTS

Irreducible binary $[n, k, d]$ Goppa code $\Gamma(L, g)$

- Minimum distance $d \geq 2\tau + 1$, $\tau = \deg g(X)$

THE INGREDIENTS

Irreducible binary $[n, k, d]$ Goppa code $\Gamma(L, g)$

- Minimum distance $d \geq 2\tau + 1$, $\tau = \deg g(X)$

Secret:

- Generator matrix $G \in \mathbb{F}_2^{k \times n}$ of $\Gamma(L, g)$
- Random binary non-singular matrix $S \in \mathbb{F}_2^{k \times k}$.
- Random permutation matrix $P \in \mathbb{F}_2^{n \times n}$.
- Private key: $(S, \mathcal{D}_{\Gamma(L,g)}, P)$
  - $\mathcal{D}_{\Gamma(L,g)}$ efficient decoder for $\Gamma(L, g)$

THE INGREDIENTS

Irreducible binary $[n, k, d]$ Goppa code $\Gamma(L, g)$

- Minimum distance $d \geq 2\tau + 1$, $\tau = \deg g(X)$

Secret:

- Generator matrix $G \in \mathbb{F}_2^{k \times n}$ of $\Gamma(L, g)$
- Random binary non-singular matrix $S \in \mathbb{F}_2^{k \times k}$.
- Random permutation matrix $P \in \mathbb{F}_2^{n \times n}$.
- Private key: $(S, \mathcal{D}_{\Gamma(L,g)}, P)$
  - $\mathcal{D}_{\Gamma(L,g)}$ efficient decoder for $\Gamma(L, g)$

Public:

- The equivalent generator matrix $\hat{G} = SGP \in \mathbb{F}_{2^m}^{k \times n}$ for $\Gamma(L, g)$
- Public key: $\hat{G}$

HOW IT WORKS

Encryption:

- Choose $e \in \mathbb{F}_2^n$ of weight τ.
- $c = m\hat{G} + e$

HOW IT WORKS

Encryption:

- Choose $e \in \mathbb{F}_2^n$ of weight τ.
- $c = m\hat{G} + e$

Decryption:

- $cP^{-1} = mSG + eP^{-1}$
- $mSG = \mathcal{D}_{\Gamma(L,g)}(cP^{-1})$

- Let $J \subseteq \{1, \ldots, n\}$ be a set such that $G_{.J}$ is invertible.
- $m = (mSG)_J(G_{.J})^{-1}S^{-1}$

GOPPA CODES

Let $g(X) \in \mathbb{F}_{q^m}[X]$ be a polynomial of degree $\tau = n - k$.
(Goppa polynomial).

Fix $L = (L_0, \ldots, L_{n-1}), L_i \in \mathbb{F}_{q^m}$, such that $g(L_i) \neq 0$ for all i.

The Goppa code $\Gamma(L, g)$ is then defined as

$$\boxed{\Gamma(L, g) := \mathcal{A}(L, y)},$$

where $y_i = g(L_i)^{-1}$.

GOPPA CODES

Let $g(X) \in \mathbb{F}_{q^m}[X]$ be a polynomial of degree $\tau = n - k$. (Goppa polynomial).

Fix $L = (L_0, \ldots, L_{n-1}), L_i \in \mathbb{F}_{q^m}$, such that $g(L_i) \neq 0$ for all i.

The Goppa code $\Gamma(L, g)$ is then defined as

$$\boxed{\Gamma(L, g) := \mathcal{A}(L, y)},$$

where $y_i = g(L_i)^{-1}$.

Under certain conditions, Goppa codes allow for a compact representation in quasi-dyadic form [7].

DYADIC MATRICES

Defined by a signature $h := (h_0, \ldots, h_{n-1})$

DYADIC MATRICES

Defined by a signature $h := (h_0, \ldots, h_{n-1})$

If n is a power of two, the matrix is highly regular, for example

$n = 2^3, h = (A, B, C, D, E, F, G, H)$ we have:

$$\Delta(h) := (h_{i \oplus j}) := \begin{bmatrix} A & B & C & D & E & F & G & H \\ B & A & D & C & F & E & H & G \\ C & D & A & B & G & H & E & F \\ D & C & B & A & H & G & F & E \\ E & F & G & H & A & B & C & D \\ F & E & H & G & B & A & D & C \\ G & H & E & F & C & D & A & B \\ H & G & F & E & D & C & B & A \end{bmatrix}$$

DYADIC MATRICES

Defined by a signature $h := (h_0, \ldots, h_{n-1})$

If n is a power of two, the matrix is highly regular, for example

$n = 2^3, h = (A, B, C, D, E, F, G, H)$ we have:

$$\Delta(h) := (h_{i \oplus j}) := \begin{bmatrix} A & B & C & D & E & F & G & H \\ B & A & D & C & F & E & H & G \\ C & D & A & B & G & H & E & F \\ D & C & B & A & H & G & F & E \\ E & F & G & H & A & B & C & D \\ F & E & H & G & B & A & D & C \\ G & H & E & F & C & D & A & B \\ H & G & F & E & D & C & B & A \end{bmatrix}$$

Quasi-dyadic matrix: block matrix with dyadic submatrices.

GOPPA CODES IN QUASI-DYADIC FORM

Generator matrix G for a quasi-dyadic Goppa code (n code length, k dimension, ℓ block size):

Only the signatures (orange) will be kept in memory.

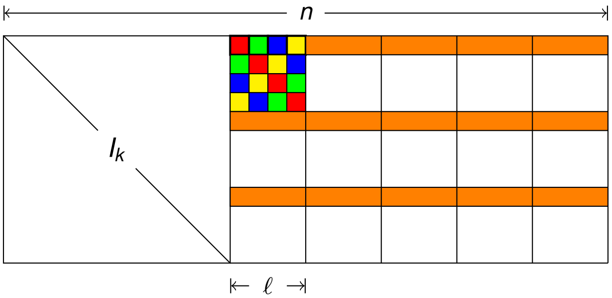


FIGURE: $k \times n$ generator matrix G

A MODERN MCELIECE VARIANT

Classical:

- **Secret** random permutation matrix P and invertible matrix S
- **Public** a generator for the code

A Modern McEliece Variant

Classical:

- **Secret** random permutation matrix P and invertible matrix S
- **Public** a generator for the code

New:

- **Secret** the support L and generator polynomial g
- **Public** a generator for the subcode
  - **No** random permutation matrix P
  - **No** invertible matrix S
  - G in systematic form

MODERN MCELIECE: HOW IT WORKS

Encryption:

- Choose $e \in \mathbb{F}_{q^m}^n$ of weight τ.
- $c = mG + e$.

Modern McEliece: How it works

Encryption:

- Choose $e \in \mathbb{F}_{q^m}^n$ of weight τ.
- $c = mG + e$.

Decryption:

- Use secret L and $g(X)$ to obtain a parity-check matrix H.
- Compute syndrome $\sigma = Hc^T = He^T$.
- Decode into error vector e.
- Read m from the first k positions of $c - e$.

FOPT

Quasi-dyadic structure of G: is it still secure ?

FOPT

Quasi-dyadic structure of G: is it still secure ?

Algebraic Cryptanalysis of McEliece Variants with Compact Keys
(J.-C. **F**augère and A. **O**tmani and L. **P**erret and J.-P. **T**illich) [2]

FOPT

Quasi-dyadic structure of G: is it still secure ?

Algebraic Cryptanalysis of McEliece Variants with Compact Keys
(J.-C. **F**augère and A. **O**tmani and L. **P**erret and J.-P. **T**illich) [2]

The attack has been applied successfully against almost all the challenges proposed.

FOPT

Quasi-dyadic structure of G: is it still secure ?

Algebraic Cryptanalysis of McEliece Variants with Compact Keys
(J.-C. **F**augère and A. **O**tmani and L. **P**erret and J.-P. **T**illich) [2]

The attack has been applied successfully against almost all the challenges proposed.

It fails in the binary case.

- It can not exploit the structure of the code.
- Computed Gröbner basis is trivial.

FOPT: Idea of the Attack in a nutshell

Let $G = (g_{i,j})$ a $k \times n$ public generator matrix (see Fig.(1)) of a Goppa code, formed by $\ell \times \ell$ blocks, i.e. $k = k_0\ell, n = n_0\ell$.

Goppa codes are alternant codes. Hence there exists a parity-check matrix H like H_{GRS} (see (1)), i.e.

$$H = \{y_j x_j^i\}.$$

Expanding the equation $H \cdot G^T = 0$, we define the system

System of equations

$$g_{i,0} Y_0 X_0^j + \cdots + g_{i,n-1} Y_{n-1} X_{n-1}^j = 0 \qquad (2)$$

where $i = 0, \ldots, k-1$, $\ell = 0, \ldots, \ell - 1$. [2, 8]

FOPT: Idea of the Attack In a nutshell
Continued

Solving this system breaks the scheme!

FOPT: Idea of the Attack In a nutshell
Continued

Solving this system breaks the scheme!

One can show (again [2, 8]) that there are

- $n_0 - 1$ unknowns Y_i
- $n_0 - m$ linear equations involving only the Y_i ($j = 0$ in (2)), where m is the extension degree.
- Hence we have $n_0 - 1 - (n_0 - m) = m - 1$ "free" variables.

Once these are determined, the system is easy to solve.

FOPT: Idea of the Attack In a nutshell
Continued

Solving this system breaks the scheme!

One can show (again [2, 8]) that there are

- $n_0 - 1$ unknowns Y_i
- $n_0 - m$ linear equations involving only the Y_i ($j = 0$ in (2)), where m is the extension degree.
- Hence we have $n_0 - 1 - (n_0 - m) = m - 1$ “free” variables.

Once these are determined, the system is easy to solve.

The authors in [2] report that this number m should not be smaller than 20. This would result in large extension fields and large keys.

How can we get out of this dilemma? Is it possible to generalize this approach?

Part IV

McEliece using Generalized Srivastava Codes

GENERALIZED SRIVASTAVA CODES

Fix $m, n, s, t \in \mathbb{N}$ and a prime power q.

Let $\alpha = (\alpha_1, \dots, \alpha_n)$, $w = (w_1, \dots, w_s)$ be $n + s$ distinct elements of $\mathbb{F}_{q^m}$.

Let $(z_1, \dots, z_n)$ be nonzero elements of $\mathbb{F}_{q^m}$.

The Generalized Srivastava (GSRV) code of order st and length n is defined by a parity-check matrix of the form:

$$H = \begin{pmatrix} H_1 \\ H_2 \\ \vdots \\ H_s \end{pmatrix} \in \mathbb{F}_{q^m}^{st \times n} \qquad (3)$$

Generalized Srivastava Codes
Continued

Each block is defined as:

$$H_i = \begin{pmatrix} \frac{z_1}{\alpha_1 - w_i} & \cdots & \frac{z_n}{\alpha_n - w_i} \\ \frac{z_1}{(\alpha_1 - w_i)^2} & \cdots & \frac{z_n}{(\alpha_n - w_i)^2} \\ \vdots & \vdots & \vdots \\ \frac{z_1}{(\alpha_1 - w_i)^t} & \cdots & \frac{z_n}{(\alpha_n - w_i)^t} \end{pmatrix}.$$

- The exponent t is a parameter of the code.
- **Note:** if $t = 1$, we have a Goppa code [5].

Generalized Srivastava Codes
Properties

For a Generalized Srivastava Code we have:

- Length $n \leq q^m - s$.
- Dimension $k \geq n - mst$,
- Minimum distance $d \geq st + 1$.

Generalized Srivastava Codes
Properties

For a Generalized Srivastava Code we have:

- Length $n \leq q^m - s$.
- Dimension $k \geq n - mst$,
- Minimum distance $d \geq st + 1$.

But now one can show [8] that we do not have $m - 1$ free variables, but rather a solution space of

$$\boxed{mt - 1}$$

free variables.

Generalized Srivastava Codes
Properties

For a Generalized Srivastava Code we have:

- Length $n \leq q^m - s$.
- Dimension $k \geq n - mst$,
- Minimum distance $d \geq st + 1$.

But now one can show [8] that we do not have $m - 1$ free variables, but rather a solution space of

$$\boxed{mt - 1}$$

free variables.

Advantage: increase t instead of m.
No need to use high extension degrees.

Keys stay reasonable small, while making FOPT less effective.

Part V

Transformation of McEliece into a CCA2-secure scheme

CCA2: ADAPTIVE CHOSEN-CIPHERTEXT ATTACK

Interactive form of chosen-ciphertext attack:

- Attacker sends a number of ciphertexts to be decrypted.
- Uses the results of these decryptions to select subsequent ciphertexts.

CCA2: ADAPTIVE CHOSEN-CIPHERTEXT ATTACK

Interactive form of chosen-ciphertext attack:

- Attacker sends a number of ciphertexts to be decrypted.
- Uses the results of these decryptions to select subsequent ciphertexts.

Goal: gradually reveal information

- about an encrypted message, or
- about the decryption key itself.

FUJISAKI-OKAMOTO TRANSFORMATION

HYBRID ENCRYPTION

$$\mathcal{E}_{pk}^{hy}(m,\sigma) = \mathcal{E}_{pk}^{asym}(\sigma,\mathcal{K}(\sigma,m)) \mid\mid \mathcal{E}_{\mathcal{H}(\sigma)}^{sym}(m)$$

$\mathcal{E}^{asym}$	asymmetric encryption scheme
$\mathcal{E}^{sym}$	symmetric encryption scheme
pk	public key
$\mathcal{H}$	any hash function (fixed-length output, key for symmetric scheme)
$\mathcal{K}$	another hash function (variable-length output, input for asymmetric scheme)
σ	a certain randomness
m	message to encrypt/decrypt

APPLYING FUJISAKI-OKAMOTO

$\mathcal{E}^{asym}$	McEliece.
$\mathcal{E}^{sym}$	A one-time pad.
pk	Public key for McEliece.

Applying Fujisaki-Okamoto

$\mathcal{E}^{asym}$	McEliece.
$\mathcal{E}^{sym}$	A one-time pad.
pk	Public key for McEliece.

Then $\boxed{\mathcal{E}^{asym}_{pk}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\sigma G + \mathcal{K}(\sigma, m)}$.

$\mathcal{K}(\sigma, m)$ needs to be an error vector.

APPLYING FUJISAKI-OKAMOTO

$\mathcal{E}^{asym}$	McEliece.
$\mathcal{E}^{sym}$	A one-time pad.
pk	Public key for McEliece.

Then $\boxed{\mathcal{E}_{pk}^{asym}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\sigma G + \mathcal{K}(\sigma, m)}$.

$\mathcal{K}(\sigma, m)$ needs to be an error vector.

Transform $\mathcal{K}(\sigma, m)$ into an error vector of fixed weight ?
In principle: Possible - Use of Constant-Weight Encoding function.

Applying Fujisaki-Okamoto

$\mathcal{E}^{asym}$	McEliece.
$\mathcal{E}^{sym}$	A one-time pad.
pk	Public key for McEliece.

Then $\boxed{\mathcal{E}^{asym}_{pk}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\sigma G + \mathcal{K}(\sigma, m)}$.

$\mathcal{K}(\sigma, m)$ needs to be an error vector.

Transform $\mathcal{K}(\sigma, m)$ into an error vector of fixed weight ?
In principle: Possible - Use of Constant-Weight Encoding function.

Could we avoid this complication?

Applying Fujisaki-Okamoto
Continued

Using a technique introduced on lattices [6] we can swap the role of the message and the randomness in the encryption process.

Then $\boxed{\mathcal{E}_{pk}^{asym}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\mathcal{K}(\sigma, m)G + \sigma}$.

Applying Fujisaki-Okamoto
Continued

Using a technique introduced on lattices [6] we can swap the role of the message and the randomness in the encryption process.

Then $\boxed{\mathcal{E}_{pk}^{asym}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\mathcal{K}(\sigma, m)G + \sigma}$.

We need:

- Length of $\boxed{\mathcal{K}(\sigma, m) = \text{rank } k \text{ of } G}$.
- σ would be a random error vector of fixed weight.
- Looks good so far: no transformation of $\mathcal{K}(\sigma, m)$.

Applying Fujisaki-Okamoto
Continued

Using a technique introduced on lattices [6] we can swap the role of the message and the randomness in the encryption process.

Then $\boxed{\mathcal{E}_{pk}^{asym}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\mathcal{K}(\sigma, m)G + \sigma}$.

We need:

- Length of $\boxed{\mathcal{K}(\sigma, m) = \text{rank } k \text{ of } G}$.
- σ would be a random error vector of fixed weight.
- Looks good so far: no transformation of $\mathcal{K}(\sigma, m)$.

But what about $\mathcal{H}$?

- We need $\mathcal{H}(\sigma)$ to have the length k of the message.

APPLYING FUJISAKI-OKAMOTO
CONTINUED

Using a technique introduced on lattices [6] we can swap the role of the message and the randomness in the encryption process.

Then $\boxed{\mathcal{E}_{pk}^{asym}(\sigma, \mathcal{K}(\sigma, m))}$ would read as $\boxed{\mathcal{K}(\sigma, m)G + \sigma}$.

We need:

- Length of $\boxed{\mathcal{K}(\sigma, m) = \text{rank } k \text{ of } G}$.
- σ would be a random error vector of fixed weight.
- Looks good so far: no transformation of $\mathcal{K}(\sigma, m)$.

But what about $\mathcal{H}$?

- We need $\mathcal{H}(\sigma)$ to have the length k of the message.

There is a cool hash function doing the job of $\mathcal{K}$ and $\mathcal{H}$ plus generating σ.

KECCAK - SPONGE CONSTRUCTION

Keccak hash function(s):

- Finalist of NIST SHA-3 contest.
- Very fast.
- Extremely flexible: can be used as hash function (arbitrary-length output) and as stream cipher.

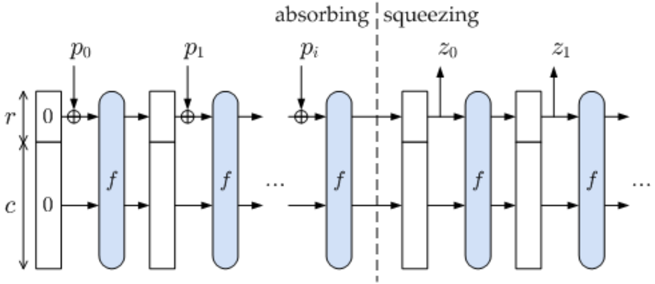


FIGURE: The sponge construction [3]

Applying Fujisaki-Okamoto
Continued

Let's summarize:

$\mathcal{E}^{asym}$	McEliece.
$\mathcal{E}^{sym}$	A one-time pad.
pk	Public key for McEliece.
$\mathcal{H}$	Keccak.
$\mathcal{K}$	Keccak.
σ	Error vector of fixed weight.
m	Message to encrypt/decrypt.

FUJISAKI-OKAMOTO TRANSFORMATION: ENCODING

To encrypt a message m, we do:

Step 1	Generate a random error vector σ of weight w.
Step 2	Set $r := \mathcal{K}(\sigma, m)$.
Step 3	Encrypt r with McEliece: $c_1 = \mathcal{K}(\sigma, m)G \oplus \sigma = rG \oplus \sigma$.
Step 4	Hash σ to be used as the key for a one-time pad.
Step 5	Symmetric encryption: $c_2 = \mathcal{H}(\sigma) \oplus m$.
Step 6	Final ciphertext: $c = c_1 \| c_2$.

FUJISAKI-OKAMOTO TRANSFORMATION: DECODING

To decrypt a ciphertext c, we do:

Step 1	Upon receiving c, parse it into the parts c_1 and c_2.
Step 2	Decrypt c_1 using McEliece. Let $\hat{\sigma} := \mathcal{D}^{asym}_{sk}(c_1)$. [1]
Step 3	Decrypt c_2 using $\mathcal{H}(\hat{\sigma})$ as key. Let $\hat{m} := \mathcal{D}^{sym}_{\mathcal{H}(\hat{\sigma})} = c_2 \oplus \mathcal{H}(\hat{\sigma})$.
Step 4	Hash $\hat{\sigma}$ and $\hat{m}$ using $\mathcal{K}$. Let $\hat{r} := \mathcal{K}(\hat{\sigma}, \hat{m})$.
Step 5	Check if c_1 equals $\mathcal{E}^{asym}_{pk}(\hat{\sigma}, \hat{r}) = \hat{r}G + \hat{\sigma}$.
Step 6	In case it is true, output $m = \hat{m}$. Otherwise reject.

[1] We assume that decoding actually works.

Part VI

Results of implementations in C

Platforms

Intel(R) Core(TM)2 Duo CPU E8400@3.00GHz

AVR Atmel Atmega 256A3

- 8-bit microcontroller, running at 32 MHz
- 256 KByte flash memory
- 16 KByte SRAM memory

- Chosen to have a comparison with recent results of S. Heyse [4] at PQCrypto 2011.

PLAIN MCELIECE
USING GENERALIZED SRIVASTAVA CODES

TABLE: Results on Intel(R) Core(TM)2 Duo CPU E8400@3.00GHz

Base Field	m	n	k	s	t	Errors	Enc.$_{[ms]}$	Dec.$_{[ms]}$	E. & D.$_{[ms]}$	Sec.
$\mathbb{F}_{2^5}$	2	512	256	2^4	8	64	0.092	1.234	1.320	2^{80}
$\mathbb{F}_{2^4}$	3	768	432	2^4	7	56	0.179	1.578	1.753	2^{80}
$\mathbb{F}_{2^5}$	2	768	416	2^4	11	88	0.188	2.491	2.677	2^{112}
$\mathbb{F}_{2^5}$	2	992	416	2^5	9	144	0.287	5.486	5.779	2^{128}

- m = extension degree
- n = code length
- k = code dimension
- s = block size
- t = exponent
- Enc. = encoding operation
- Dec. = decoding operation
- E & D. = encoding/decoding cycle

CCA2-SECURE MCELIECE
USING GENERALIZED SRIVASTAVA CODES

TABLE: Results on Intel(R) Core(TM)2 Duo CPU E8400@3.00GHz

Base Field	m	n	k	s	t	Errors	Enc.$_{[ms]}$	Dec.$_{[ms]}$	Sec.
$\mathbb{F}_{2^5}$	2	512	256	2^4	8	64	0.114	1.382	2^{80}
$\mathbb{F}_{2^4}$	3	768	432	2^4	7	56	0.213	1.814	2^{80}
$\mathbb{F}_{2^5}$	2	768	416	2^4	11	88	0.216	2.721	2^{112}
$\mathbb{F}_{2^5}$	2	992	416	2^5	9	144	0.323	5.914	2^{128}

- m = extension degree
- n = code length
- k = code dimension
- s = block size
- t = exponent
- Enc. = encoding operation
- Dec. = decoding operation
- E & D. = encoding/decoding cycle

PARAMETERS ON THE AVR

For the AVR, we used a setting with security of 2^{80} bit operations:

Extension field	$\mathbb{F}_{2^{12}}$
Base field	$\mathbb{F}_{2^4}$
Extension degree m	3
Code length n	768
Code dimension k	432
Block size s	16
Exponent t	7
Number of errors w	56

MEMORY REQUIREMENTS

G	9.072 bytes
H	172.032 bytes

TABLE: Memory requirements for public/private keys G resp. H

For our implementation, we wasted some memory:

- G really needs 9.072×4 bits = 4.536 bytes.
- H really needs 86.016×12 bits = 129.024 bytes.

- The ATxmega256A3 has 256 KByte flash memory.
- The device has been chosen mainly for comparison with [4].
- Using a device with 192 KByte possible.

Encoding: Plain McEliece

using Generalized Srivastava Codes

	Cycles	Comment
Start	3.587	Device initialization.
Init PRNG	139.250	Seed Keccak. Absorbing phase.
Init e	317.003	Generate error vector $e \in \mathbb{F}_{2^4}^n$ with weight 56. Squeezing phase of Keccak.
Init m	4.313	Load the message m
mG	3.418.292	Encode message $m \in \mathbb{F}_{2^4}^k$.
$mG + e$	8.818	Add code and error vector.
	3.891.263	Cycles for encoding operation (122 ms).

Table: Time for encoding operation $mG + e = c$.

DECODING: PLAIN MCELIECE
USING GENERALIZED SRIVASTAVA CODES

	Cycles	Comment
$cH^T = s$	6.910.742	Compute syndrome $s \in \mathbb{F}_{2^{12}}^{n-k}$.
$\sigma(X), \omega(X)$	955.597	Solve the key equation to obtain the error locator $\sigma(X)$ and error evaluator $\omega(X)$.
$\sigma(X)$	2.061.066	Compute roots of $\sigma(X)$, i.e. the error positions.
$\omega(X)$	611.898	Evaluate $\omega(X)$ at error positions to obtain error vector $e \in \mathbb{F}_{2^4}^{n}$.
$c + e$	8.641	Correct the ciphertext.
	10.547.944	Cycles for decoding operation (330 ms).

TABLE: Time for decoding operation.

Encoding: CCA2-secure McEliece
using Generalized Srivastava Codes

	Cycles	Comment
Start	3.591	Device initialization.
Init PRNG	139.253	Seed Keccak. Absorbing phase.
Init σ	322.109	Generate error vector $e \in \mathbb{F}_{2^4}^n$ with weight 56. Squeezing phase of Keccak.
Init m	1.019	Load the message m
$r = \mathcal{K}(\sigma, m)$	281.285	Hash (σ, m) to obtain message r.
rG	3.426.700	Encode message $r \in \mathbb{F}_{2^4}^k$.
$\mathcal{H}(\sigma)$	137.704	Hash the error vector σ.
$\mathcal{H}(\sigma) + m$	1.814	Add $H(\sigma)$ and m to obtain c_2.
$rG + \sigma$	1.244	Add code and error vector to obtain c_1.
	4.314.719	Cycles for encoding operation (135 ms).

Table: Time for encoding operation $mG + e = c$.

DECODING: CCA2-SECURE MCELIECE
USING GENERALIZED SRIVASTAVA CODES

	Cycles	Comment
$c_1 H^T = s$	7.029.844	Compute syndrome $s \in \mathbb{F}_{2^{12}}^{n-k}$.
$\sigma(X), \omega(X)$	954.522	Solve the key equation to obtain the error locator $\sigma(X)$ and error evaluator $\omega(X)$.
$\sigma(X)$	2.031.514	Compute roots of $\sigma(X)$, i.e. the error positions.
$\omega(X)$	611.944	Evaluate $\omega(X)$ at error positions to obtain error vector $\hat{\sigma} \in \mathbb{F}_{2^4}^{n}$.
$\mathcal{H}(\hat{\sigma})$	147.822	Hash decoded error vector $\hat{\sigma}$.

TABLE: Time for decoding operation (part 1).

Decoding continued: CCA2-secure McEliece
using Generalized Srivastava Codes

	Cycles	Comment
$\mathcal{H}(\hat{\sigma}) + c_2 = \hat{m}$	1.585	Add $H(\hat{\sigma})$ and c_2 to obtain $\hat{m}$.
$\hat{r} = \mathcal{K}(\hat{\sigma}, \hat{m})$	282.066	Hash $(\hat{\sigma}, \hat{m})$ to obtain message $\hat{r}$.
$\hat{r}G$	3.426.721	Encode message $\hat{r} \in \mathbb{F}_{2^4}^k$.
$\hat{r}G + \hat{\sigma} = \hat{c}_1$	1.113	Correct the ciphertext to obtain $\hat{c}_1$.
$c_1 = \hat{c}_1$	9.207	Check if $c_1 = \hat{c}_1$.
	14.496.338	Cycles for decoding operation (453 ms).

Table: Time for decoding operation (part 2).

Summary and future work

- McEliece using Generalized Srivastava Codes is a viable alternative to binary quasi-dyadic Goppa codes.
- It is possible to transform McEliece into a CCA2-secure scheme with only a small loss in runtime.
- The implementation is fast and does not need a constant-weight encoding function.
- Depending on the interest, we might give assembler implementations on AVR and MMIX.

- Finally, we like to thank Paulo S. L. M. Barreto (et al.) to provide us insight into their library SBCrypt [1].
- It served as invaluable starting point for the C implementation.

Thanks!

REFERENCES

P. S. L. M. Barreto, R. Misoczki, and L. B. Villas Boas.
SBCRYPT - Syndrome-Based Cryptography Library.
Private communication.

J.-C. Faugère, A. Otmani, L. Perret, and J.-P. Tillich.
Algebraic Cryptanalysis of McEliece Variants with Compact Keys.
In *EUROCRYPT*, pages 279–298, 2010.

G. Bertoni and J. Daemen and M. Peeters and G. Van Assche.
The Keccak sponge function family.
http://keccak.noekeon.org/.

S. Heyse.
Implementation of McEliece based on Quasi-Dyadic Goppa Codes for Embedded Devices.
2011.
Accepted paper at PQCrypto 2011. To appear.

F. J. MacWilliams and N. J. A. Sloane.
The Theory of Error-Correcting Codes, volume 16.
North-Holland Mathematical Library, 1977.

D. Micciancio.
Improving lattice based cryptosystems using the hermite normal form.
In *CaLC*, pages 126–145, 2001.

R. Misoczki and P. S. L. M. Barreto.
Compact McEliece keys from Goppa codes.
In *Selected Areas in Cryptography – SAC'2009*, volume 5867 of *LNCS*, pages 276–392. Springer, 2009.

E. Persichetti.
Compact McEliece keys based on Quasi-Dyadic Srivastava codes.
Cryptology ePrint Archive, Report 2011/179, 2011.

REFERENCES
CONTINUED

P. S. L. M. Barreto.
Post-Quantum Cryptography, 2009.
PQC-4.pdf. Private communication.

E. Fujisaki and T. Okamoto.
Secure Integration of Asymmetric and Symmetric Encryption Schemes.
In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 537–554, London, UK, 1999. Springer-Verlag.

S. Heyse.
Implementation of McEliece based on Quasi-Dyadic Goppa Codes for Embedded Devices.
2011.
Accepted paper at PQCrypto 2011. To appear.

S. Heyse.
Implementational Aspects of Code-based Cryptography for Embedded Systems, 2011.
http://www.win.tue.nl/cccc/cbc/slides/Stefan-Heyse.pdf.

F. J. MacWilliams and N. J. A. Sloane.
The Theory of Error-Correcting Codes, volume 16.
North-Holland Mathematical Library, 1977.

P. W. Shor.
Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer.
SIAM Journal on Computing, 26:1484–1509, 1995.

Wikipedia.
CCA2, 2011.
http://en.wikipedia.org/wiki/Adaptive_chosen_ciphertext_attack.