How SAGE helps to implement Goppa Codes and McEliece PKCSs

Thomas Risse
DSI GmbH Bremen
Institute of Informatics & Automation, IIA
Faculty EEE & CS, Hochschule Bremen
University of Applied Sciences, Germany
risse@hs-bremen.de

ICIT’11, May 11th, 2011, Amman, Jordan
Introduction

Objectives

The McEliece PKCS

Goppa Codes

Decoding generalized RS-Codes

The Berlekamp-Massey Algorithm

Demonstration: Goppa-Codes in SAGE

SAGE does help to implement McEliece PKCSs
Introduction

Bad news: Once quantum computers become operational, today's widely used public key cryptosystems, PKCSs like

- Rivest-Shamir-Adleman, RSA
- Diffie-Hellman (-Merkle)
- elliptic curve cryptography, ECC
- or the Buchmann-Williams key exchange

are broken! [1], [12]

Good news: Alternatives like

- McEliece PKCS
- Goldreich-Goldwasser-Halevi, GGH lattice based PKCSs
- hash based digital signature schemes
- or multivariate PKCSs

are believed to be quantum computer resistant. [1]
Objectives

Implement McEliece PKCS on FPGAs

In order to do so, we adhere to a two-version-approach:

- we use a quick implementation in SAGE to test the implementation in VHDL
- so that we can check implementational details before hand in SAGE

Experience shows that the open source computer algebra System for Algebraic and Geometric Experimentation, SAGE is a very potent tool for this task!

Here, we will

- sketch McEliece PKCS
- sketch Goppa-Codes
- demonstrate implementation of decoding algorithms in SAGE
The McEliece PKCS

Three binary matrices specify a McEliece PKCS

- $\mathbf{G}$, the $k \times n$ generator matrix of a $[n, k, 2t + 1]$-Goppa-Code, i.e. $n = 2^m$, $k = n - mt$
- $\mathbf{S}$, a random non-singular $k \times k$ scrambler matrix
- $\mathbf{P}$, a random $n \times n$ permutation matrix

The triple $(\mathbf{S, G, P})$ is a private key with corresponding public key $\hat{\mathbf{G}} = \mathbf{SGP}$.

Plain text is partitioned into bit-blocks $\mathbf{u}$ of length $k$. Each block is Goppa encoded to $\mathbf{c} = \mathbf{u}\hat{\mathbf{G}}$ of length $n$. Each code word $\mathbf{c}$ is charged with maximal $t$ 1-bit-errors, i.e. modified to $\mathbf{y} = \mathbf{c} + \mathbf{e}$ with error vector $\mathbf{e}$.

Using the private key, the receiver decodes $\mathbf{y}$ to $\mathbf{yP}^{-1} = \mathbf{cP}^{-1} + \mathbf{eP}^{-1} = \mathbf{uSG} + \mathbf{eP}^{-1}$ per fast Goppa decoding to $\mathbf{uS}$ and per $\mathbf{uSS}^{-1}$ to $\mathbf{u}$. 

Goppa Codes

**Goppa Codes** are alternant generalized Reed-Solomon codes. Specifically, let \( F = \mathbb{GF}_2 \), \( \Phi = \mathbb{GF}(q^m) \) and \( g(x) \in \Phi[x] \) be a Goppa polynomial of degree \( t \). Let \( L = \{ \lambda_1, \ldots, \lambda_n \} \subset \mathbb{GF}(q^m) \) be the code locators with \( g(\lambda_i) \neq 0 \) for \( i = 1, 2, \ldots, n \). Then the linear subspace \( C_{\text{Goppa}}(L, g) = \{ (c_1, \ldots, c_n) \in \mathbb{GF}_2^n : \sum_{i=1}^{n} \frac{c_i}{x - \lambda_i} = 0 \mod g(x) \} \) is a \([n, k, d]\) Goppa code with \( k \geq n - mt \) and canonical parity matrix

\[
H_{g\text{RS}} = (\lambda_j^{i-1})_{i=1, j=1}^{\deg g-1, n} \text{diag}(\frac{1}{g(\lambda_1)}, \frac{1}{g(\lambda_2)}, \ldots, \frac{1}{g(\lambda_n)}).
\]

Selecting a base of \( \Phi \) and representing each entry of \( H_{g\text{RS}} \) as a column vector in \( \mathbb{F}^m \) results in the parity matrix \( H_{\text{Goppa}} \) of \( C_{\text{Goppa}} \) with canonical generator matrix \( G_{\text{Goppa}} = \ker(H_{\text{Goppa}}) \).
Decoding generalized RS-Codes

Information words $\mathbf{u}$ are encoded to code words $\mathbf{c} = \mathbf{uG}_{\text{Goppa}}$. Received words $\mathbf{y}$ have syndromes $\mathbf{s} = \mathbf{H}_{\text{Goppa}}\mathbf{y}^\top$ which are decoded to the in the Hamming metric nearest code word by solving the key equation

$$\sigma(x)s(x) = \omega(x) \mod x^{d-1}$$

with $\gcd(\sigma, \omega) = 1$ and $\deg \omega < \deg \sigma \leq (d - 1)/2$

- by a variant of the extended Euclidean algorithm [9]
- using linear recurrences
  - by the Berlekamp-Massey algorithm [9]
- in case of binary Goppa codes
  - by the Patterson algorithm [2]
The Berlekamp-Massey Algorithm

Berlekamp\textsuperscript{1}-Massey\textsuperscript{2} algorithm which computes a $N$-recurrence of (a syndrome) $b(x)$.

**Input:** polynomial $b(x) \in \mathbb{F}[x]$, $0 < N \in \mathbb{N}$

**Output:** pair of polynomials $(\sigma_N(x), \omega_N(x))$ over $\mathbb{F}$

- $\sigma_{-1}(x) = 0$; $\sigma_0(x) = 1$;
- $\omega_{-1}(x) = -x^{-1}$; $\omega_0(x) = 0$;
- $\mu = -1$; $\delta_{-1} = 1$;
- for ($i = 0$; $i < N$; $i++$)
  
  - $\delta_i = \text{coefficient of } x^i \text{ in } \sigma_i(x) b(x)$;
  - $\sigma_{i+1}(x) = \sigma_i(x) - (\delta_i/\delta_{\mu}) x^{i-\mu} \sigma_\mu(x)$;
  - $\omega_{i+1}(x) = \omega_i(x) - (\delta_i/\delta_{\mu}) x^{i-\mu} \omega_\mu(x)$;
  - if $\delta_i \neq 0$ \&\& $2 \text{ord}(\sigma_i, \omega_i) \leq 1$ then $\mu = i$;

} // where $\text{ord}(\sigma, \omega) = \max\{\deg \sigma, 1 + \deg \omega\}$

\textsuperscript{1}Elwyn R. Berlekamp: Algebraic Coding Theory; Aegan Park Press 1984, Section 7.4

The Berlekamp-Massey Algorithm, contd.

- For given input $b$, the algorithm computes the $N$-recurrence $(\sigma_n, \omega_n)$ over $\mathbb{F}$ with smallest recurrence order.
- Applied to a syndrome $b$ the algorithm returns the error locator polynomial $\sigma(x)$ and error evaluator polynomial $\omega(x)$.
- In case of a binary code we do not need to compute the error evaluator polynomial $\omega(x)$.
- We can translate the algorithm more or less literally to SAGE!

Arrays in SAGE are numbered starting with index 0. For the sake of clarity, we use actual polynomials $\omega_i$ and $\sigma_i$ instead of maintaining just two triples $\sigma_{\text{new}}, \sigma_{\text{old}}, \sigma_{\mu}$ and $\omega_{\text{new}}, \omega_{\text{old}}, \omega_{\mu}$ of polynomials which are updated in each loop.
Demonstration: Goppa-Codes in SAGE

run the SAGE worksheet www.weblearn.hs-bremen.de/risse/papers/ICIT11/Goppa_codes_BM.sws on any SAGE server, e.g.

- www.sagenb.org, the official SAGE server
- https://sage.informatik.hs-bremen.de, the SAGE server at Hochschule Bremen
SAGE does (help to) implement McEliece PKCSs

Summarizing

- SAGE is a very potent tool to implement e.g. McEliece PKCS.
- SAGE helps to develop and test e.g. an implementation in VHDL.
- SAGE helps to assess variants in this implementation:
  - whether to do inversion in $\mathbb{GF}(2^m)$ by table look up or by recursion [7] (a SAGE version is presented in [8])
  - whether to compute square roots in $\mathbb{GF}(2^m)$ – as well as in $\mathbb{GF}(2^m)[z]/g(z)$ – by an algorithm [3] or by table look up
  - whether to transform to and from normal bases in order to profit from the simple arithmetic using normal bases

SAGE compares favourably with e.g. MATLABs Communication Tool Box.
Referenzen I


Referenzen II

www.comms.scitech.sussex.ac.uk/fft/crypto/rijndael-sbox.pdf or [8]

www.weblearn.hs-bremen.de/risse/papers/Frege2010_03/

www.cs.technion.ac.il/~ronny/

[10] NN: System for Algebraic and Geometric Experimentation, SAGE
www.SAGEmath.org
