Code-based cryptography III

Goppa codes: definition and usage

Tanja Lange
with some slides by Tung Chou and Christiane Peters

Eindhoven University of Technology

SAC – Post-quantum cryptography

Binary Goppa code

Let $q = 2^m$. A binary Goppa code is often defined by

- a list $L = (a_1, \ldots, a_n)$ of n distinct elements in $\mathbb{F}_q$, called the support.
- a square-free polynomial $g(x) \in \mathbb{F}_q[x]$ of degree t such that $g(a) \neq 0$ for all $a \in L$. $g(x)$ is called the Goppa polynomial.
- E.g. choose $g(x)$ irreducible over $\mathbb{F}_q$.

The corresponding binary Goppa code $\Gamma(L, g)$ is

$$\left\{ \mathbf{c} \in \mathbb{F}_2^n \,\middle|\, S(\mathbf{c}) = \frac{c_1}{x - a_1} + \frac{c_2}{x - a_2} + \cdots + \frac{c_n}{x - a_n} \equiv 0 \bmod g(x) \right\}$$

- This code is linear $S(\mathbf{b} + \mathbf{c}) = S(\mathbf{b}) + S(\mathbf{c})$ and has length n.
- What can we say about the dimension and minimum distance?

Dimension of $\Gamma(L, g)$

- $g(a_i) \neq 0$ implies $\gcd(x - a_i, g(x)) = 1$, thus get polynomials

$$(x - a_i)^{-1} \equiv f_i(x) \equiv \sum_{j=0}^{t-1} f_{i,j} x^j \bmod g(x)$$

  via XGCD. All this is over $\mathbb{F}_q = \mathbb{F}_{2^m}$.
- In this form, $S(\mathbf{c}) \equiv 0 \bmod g(x)$ means

$$\sum_{i=1}^{n} c_i \left(\sum_{j=0}^{t-1} f_{i,j} x^j \right) = \sum_{j=0}^{t-1} \left(\sum_{i=1}^{n} c_i f_{i,j} \right) x^j = 0,$$

  meaning that for each $0 \leq j \leq t - 1$:

$$\sum_{i=1}^{n} c_i f_{i,j} = 0.$$

- These are t conditions over $\mathbb{F}_q$, so tm conditions over $\mathbb{F}_2$. Giving an $tm \times n$ parity check matrix over $\mathbb{F}_2$.
- Some rows might be linearly dependent, so $k \geq n - tm$.

Nice parity check matrix

Assume $g(x) = \sum_{i=0}^{t} g_i x^i$ monic, i.e., $g_t = 1$.

$$H = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ g_{t-1} & 1 & 0 & \dots & 0 \\ g_{t-2} & g_{t-1} & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & g_3 & \dots & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & \cdots & 1 \\ a_1 & a_2 & a_3 & \cdots & a_n \\ a_1^2 & a_2^2 & a_3^2 & \cdots & a_n^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_1^{t-1} & a_2^{t-1} & a_3^{t-1} & \cdots & a_n^{t-1} \end{pmatrix}$$

$$\cdot \begin{pmatrix} \frac{1}{g(a_1)} & 0 & 0 & \dots & 0 \\ 0 & \frac{1}{g(a_2)} & 0 & \dots & 0 \\ 0 & 0 & \frac{1}{g(a_3)} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \frac{1}{g(a_n)} \end{pmatrix}$$

Reminder: How to hide nice code?

- Do not reveal matrix H related to nice-to-decode code.
- Pick a random invertible $(n-k) \times (n-k)$ matrix S and random $n \times n$ *permutation matrix* P. Put

$$K = SHP.$$

- K is the public key and S and P together with a decoding algorithm for H form the private key.
- For suitable codes K looks like random matrix.
- How to decode syndrome $\mathbf{s} = K\mathbf{e}$?
- Computes $S^{-1}\mathbf{s} = S^{-1}(SHP)\mathbf{e} = H(P\mathbf{e})$.
- P permutes, thus $P\mathbf{e}$ has same weight as $\mathbf{e}$.
- Decode to recover $P\mathbf{e}$, then multiply by P^{-1}.

How to hide nice code?

- For Goppa code use secret polynomial $g(x)$.
- Use secret permutation of the a_i, this corresponds to secret permutation of the n positions; this replaces P.
- Use systematic form $K = (K'|I)$ for public key;
  Store only K' part.
  - This implicitly applies S.
  - No need to remember S because decoding does not use H. (see Code-based crypto IV).
  - Public key size decreased to $(n-k) \times k$.
- Private key is polynomial g and support $L = (a_1, \ldots, a_n)$.