The Russian Attack

The goal of this challenge is to break the "original" Niederreiter encryption scheme, which is based on the Generalized Reed-Solomon (GRS) codes. The parameters, public key, and the challenge ciphertext of the Niederreiter encryption scheme can be found in the file `Challenge.txt.bz2`. You should decrypt the challenge ciphertext, and recover the plaintext, which is of the form `SharifCTF{flag}`, where `flag` is a 16-byte hexadecimal number.

To aid you in solving this challenge, several useful functions are provided in the file `Niederreiter-Implementation.gap`, which is written for the GAP system[1]. The file is heavily commented to make it easier to read (hopefully!).

Inside `Challenge.txt.bz2`, you find:

1. Parameters q, n, and k of the Niederreiter encryption scheme;
2. Matrix $\boldsymbol{M}$, the public key of the Niederreiter encryption scheme. $\boldsymbol{M}$ is a $k \times n$ matrix over $\mathbb{F}_q$ (the finite field of prime order q).
3. The ciphertext $\boldsymbol{ctxt} = [\ell, \boldsymbol{c}]$, where ℓ is the length of the plaintext prior to padding[2], and $\boldsymbol{c} \in \mathbb{F}_q^n$ is the actual ciphertext.

If you want to review the Niederreiter encryption scheme, read Appendix A. The GRS codes are described in Appendix B.

[1] GAP stands for Groups, Algorithms, Programming. GAP is an open-source system for computational discrete algebra. It provides many useful packages, including GUAVA, which is a GAP package for computing with error-correcting codes. GUAVA provides functionality required to solve this challenge. Such functionality is missing in most other mathematics software system, such as SageMath (though SageMath provides an interface for GAP.)

[2] The Plaintext is first encoded as a vector over $\mathbb{F}_q$. Here, $\ell \leq k$ is the length of this vector. Before encryption, the plaintext is left-padded with $k - \ell$ random elements from $\mathbb{F}_q$ to construct a vector in $\mathbb{F}_q^k$.

A Original Niederreiter encryption scheme

A.1 Parameter Generation

Let q be a prime, n be a natural number, and $k \in \{0, \ldots, n\}$. In what follows, $\mathbb{F}_q$ denotes the finite field of prime order q.

1. Pick n random and distinct elements $\alpha_1, \ldots, \alpha_n$ from $\mathbb{F}_q$. Let $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$.
2. Let $\boldsymbol{G}$ be the generator matrix of $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha})$. (See Appendix B.)
3. Let H be a random, invertible $k \times k$ matrix over $\mathbb{F}_q$.

Matrix $\boldsymbol{M} = \boldsymbol{H} \cdot \boldsymbol{G}$ is the public key, and G is the private key.

A.2 Encryption

To encrypt a message msg:

1. Encode msg as a row vector $\boldsymbol{v} \in \mathbb{F}_q^k$. If necessary, add random padding to the left, such that the vector $\boldsymbol{v}$ is of the required length k.
2. Generate a random row vector $\boldsymbol{e} \in \mathbb{F}_q^k$ with Hamming weight t (see Appendix B).
3. The ciphertext is $\boldsymbol{ctxt} = \boldsymbol{v} \cdot \boldsymbol{M} + \boldsymbol{e}$.

A.3 Decryption

To decrypt a ciphertext $ctxt$:

1. Use a GRS decoding algorithm, such as Gao, to remove the error: $\boldsymbol{c} = \mathrm{Decode}(\boldsymbol{ctxt}) = \boldsymbol{v} \cdot \boldsymbol{M}$. (See Appendix B.)
2. Let $\boldsymbol{P}$ be any right inverse of $\boldsymbol{M}$. Then, $\boldsymbol{v} = \boldsymbol{c} \cdot \boldsymbol{P}$.
3. Decode $\boldsymbol{ctxt}$ to recover msg.

B Generalized Reed-Solomon (GRS) codes

Let $\mathbb{F}_q$ be the finite field of prime order q, and $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_n)$ be n distinct elements in this field. For $k \in \{0, \ldots, n\}$, define the unweighted[3] Generalized Reed-Solomon (GRS) codes as follows:

$$\mathrm{GRS}_{n,k}(\boldsymbol{\alpha}) = \{f(\alpha_1), \ldots, f(\alpha_n) | f(x) \in \mathbb{F}_q[x]_k\} \quad ,$$

[3] In its most general case, a GRS code can be weighted, where weights are defined as a vector $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{F}_q^*$ (i.e., the multiplicative group of $\mathbb{F}_q$). For simplicity, we only consider the unweighted case.

where $\mathbb{F}_q[x]_k$ denotes the set of polynomials in $\mathbb{F}_q[x]$ whose degree is (strictly) less than k. Notice that $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha})$ can correct up to $t = \lfloor \frac{n-k}{2} \rfloor$ errors.

The (canonical) generator matrix of $\mathrm{GRS}_{n,k}$ is defined by the following $k \cdot n$ matrix over $\mathbb{F}_q$:

$$\boldsymbol{G} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ \alpha_1 & \alpha_2 & \cdots & \alpha_n \\ \vdots & \vdots & \vdots & \vdots \\ \alpha_1^{k-1} & \alpha_2^{k-1} & \cdots & \alpha_n^{k-1} \end{bmatrix} . \tag{1}$$

If $\boldsymbol{x} \in \mathbb{F}_q^k$ is a row vector, then $\boldsymbol{x} \cdot G$ is a codeword in $\mathrm{GRS}_{n,k}(\boldsymbol{\alpha})$. Let $\boldsymbol{e} \in \mathbb{F}_q^n$ be a row vector of Hamming weight at most t. There are decoding algorithms (such as the one due to Gao [1]) which, given $\boldsymbol{v} = \boldsymbol{x} \cdot \boldsymbol{G} + \boldsymbol{e}$, can remove $\boldsymbol{e}$ (the *error*), and recover find $\boldsymbol{x} \cdot \boldsymbol{G}$ (the codeword).

For further information, see [2].

References

[1] Gao, Shuhong. "A New Algorithm for Decoding Reed-Solomon Codes." *Communications, Information and Network Security*, pp. 55–68, 2003.

[2] Hall, Jonathan I. "Chapter 5. Generalized Reed-Solomon Codes," 2012. Available online from `http://users.math.msu.edu/users/jhall/classes/codenotes/grs.pdf`.