Progetto Di Ingegneria Informatica

Aleknovich's Cryptosystem

Mattia Campana

Project Overview

Importance of data protection

With the rise of quantum computing, traditional cryptographic method are at risk, necessitating the development of quantum-resistant cryptosystem.

Project Focus

Exploring alekhnovich's cryptosystem, whic is based on the hardness of decoding linear codes, as a viable post quantum solution.

Target

Assert the theoretical foundations, implementation, and security of the cryptosystem.

The Algorithm

Keys Generation

Gereration of random matrices S (private key) and A, followed by the computation of the public key Y.

Encryption Process

The encryption involves generating a random vectore and combining it with the message and public key.

Decryption Process

Decryption leverages the private key and erro-correcting code to recover the original message from the cyphertext.

Main

Entry point for the whole project:

- *test 0* to ensure that generate and store all data correctly.
- *test 1* to measure the efficiency of the error correcting algorithm.

index.c

```c
switch(cmd) {
        case TEST:
            test(argc > 2 ? atoi(argv[2]) : 0);
            break;

        case GENERATE:
            generate_key();
            break;

        case ENCRYPT:
            if (argc < 4) {
                print_err(argv[0], "encrypt <message> <key_a_path> <key_y_path>\n");
                return 2;
            }
            encrypt(argv[2], argv[3], argv[4]);
            break;

        case DECRYPT:
            if (argc < 4) {
                print_err(argv[0], "decrypt <nnc_path> <word_path> <key_path>\n");
                return 2;
            }
            decrypt(argv[2], argv[3], argv[4]);
            break;

        case CORRECT:
            if (argc < 4) {
                print_err(argv[0], "correct <input_path> <input_path> <input_path> ...\n");
                return 2;
            }
            correct(argv[2], argv[3], argv[4]);
            break;

        default:
            printf("Invalid command: %s\n", argv[1]);
            return 3;
    }
```

Transpose

Like many other functions I had to implement, this one also operates on a matrix based on individual bits.

index.c

```c
struct mat *matrix_transpose(struct mat *m) {
    if (m == NULL)
        return NULL;

    struct mat *t = calloc(1, sizeof(struct mat));
    if (t == NULL)
        return NULL;

    t->rows = m->cols;
    t->cols = m->rows;

    t->data = calloc(t->rows, sizeof(uint64_t *));
    if (t->data == NULL) {
        free(t);
        return NULL;
    }

    for (int i = 0; i < t->rows; i++) {
        t->data[i] = calloc(real_dim(t->cols), sizeof(uint64_t));
        if (t->data[i] == NULL) {
            free_mat(t);
            return NULL;
        }
    }

    for (int i = 0; i < m->rows; i++) {
        for (int j = 0; j < m->cols; j++) {
            int bit_pos_in_block = j % SIZE;
            int block_index = j / SIZE;

            if (m->data[i][block_index] & (1ULL << bit_pos_in_block)) {
                int transpose_block_index = i / SIZE;
                int transpose_bit_pos_in_block = i % SIZE;

                t->data[j][transpose_block_index] |= (1ULL << transpose_bit_pos_in_block);
            }
        }
    }

    return t;
}
```

Transpose

For loops used for correct bit arrangement

```c
struct mat *matrix_transpose(struct mat *m) {
    if (m == NULL)
        return NULL;

    struct mat *t = calloc(1, sizeof(struct mat));
    if (t == NULL)
        return NULL;

    t->rows = m->cols;
    t->cols = m->rows;

    t->data = calloc(t->rows, sizeof(uint64_t *));
    if (t->data == NULL) {
        free(t);
        return NULL;
    }

    for (int i = 0; i < t->rows; i++) {
        t->data[i] = calloc(real_dim(t->cols), sizeof(uint64_t));
        if (t->data[i] == NULL) {
            free_mat(t);
            return NULL;
        }
    }
```

```c
for (int i = 0; i < m->rows; i++) {
    for (int j = 0; j < m->cols; j++) {
        int bit_pos_in_block = j % SIZE;
        int block_index = j / SIZE;

        if (m->data[i][block_index] & (1ULL << bit_pos_in_block)) {
            int transpose_block_index = i / SIZE;
            int transpose_bit_pos_in_block = i % SIZE;

            t->data[j][transpose_block_index] |= (1ULL << transpose_bit_pos_in_block);
        }
    }
}
```

```c
}
```

Write Key

Function used to save the public and private keys.

In the same way work the *write_packet* where it saves elements like the encrypted or decrypted messages.

```c
index.c

void write_key( const char *path, struct mat *m) {
    FILE *file = fopen(path, "wb");
    if (file == NULL)
        return;

    fwrite(&(m->rows), sizeof(int), 1, file);
    fwrite(&(m->cols), sizeof(int), 1, file);

    for (int i = 0; i < m->rows; i++) {
        fwrite(m->data[i], sizeof(uint64_t), real_dim(m->cols), file);
    }

    fclose(file);
}
```

Output Generated

These are examples of the possible result after running the test command.

Keep in mind that by using random vectors to introduce noise, the values are not deterministic.

index.sh

```
~/build/bin ./MyProject test 0

Starting the testing process...
Generating matrices...
Computing matrix y...
Matrices generated successfully.

Checking the reading of public and private keys...
Keys read correctly.

Starting encryption and decryption check...
Packets transmitted and verified correctly.

Decryption process completed successfully.

Hamming distance from original message: 623
```

index.sh

```
~/build/bin ./MyProject test 1

Finished target/test1.bin.
Hamming distance from original message: 638

Finished target/test2.bin.
Hamming distance from original message: 539

Finished target/test3.bin.
Hamming distance from original message: 559

Trying to correct message.

After correcting code w/ three test...
Hamming distance from original message: 0
```

Future Developments

Post-Quantum Cyptography

Continued research into quantum resistant algorithms is crucial to protect against future quantum threats.

Quantum Computing

Despite the non-imminence of quantum computers for private use, the technology is improving every year.

Standardization Effort

It will require a collective effort to implement new algorithms promptly, thereby minimizing this threat.

End