`Cry` – Project 2
(Software Requirements Specification): Report

Daniel Dunning, Michael Degraw, Vu Phan

2017-02-13

Contents

1 Introduction

(by Michael Degraw)

1.1 Purpose

This document outlines the requirements that `Team Crybabies` will meet in the development of `Cry`. The target audience is cryptographers, as well as crypto-nerds who want to use `Cry` for exchanging messages.

1.2 Scope

The primary purpose of `Cry` is to give cryptographers the ability to benchmark the cryptosystems they are developing. The secondary purpose of `Cry` is to allow users to encrypt and decrypt data.

1.3 Definitions, Acronyms, and Abbreviations

- `Cry`: the cryptoframework under development.
- `Team Crybabies`: the team responsible for developing `Cry`.
- Cryptographers: the intended audience of `Cry`.

1.4 References

- **GMP (GNU Multiple Precision arithmetic library)**: `https://gmplib.org/`
- **Msieve (General Number Field Sieve integer factorization library)**: `https://github.com/radii/msieve`

1.5 Overview

An overall description of `Cry` can be found in section 2. Descriptions of different interfaces and `C++` function prototypes can be found in section 3.

2 Overall Description

(by Daniel Dunning)

2.1 Product Perspective

`Cry` will be implemented as a cryptoframework with built-in cryptosystems updated as needed. No other frameworks are needed to run `Cry`; it is standalone. It will be used as a tool for cryptographers and developers alike, and all cryptosystems will be self-contained in `Cry`.

2.1.1 User Interface

`Cry` will be used as a command line application, being accessed with the command `cry`.

2.2 Product Functions

The essential functions of `Cry` can be broken into two separate parts: testing and reporting.

2.2.1 Testing

- Users shall be able to develop new cryptosystems by defining key-generation, encryption, decryption, and cryptanalysis algorithms.
- Users shall be able to test their cryptosystems' security.

2.2.2 Reporting

- Upon performing a test, a user will receive a report on the cryptosystem.
- The report will gauge the security of the cryptosystem and indicate its weaknesses.

2.3 User Characteristics

Users of `Cry` will most likely have at least a medium level of experience with cryptography. To use advanced features of `Cry`, users will need a high level of cryptographic experience.

2.4 Constraints

`Cry` will need certain base requirements:

- basic memory and CPU availability
- command line permissions

In addition to baseline requirements, parallel operation and interfacing with other applications may become necessary if future library additions dictate such.

2.5 Assumptions and Dependencies

Few assumptions are needed as `Cry` will run on all operating systems and is a standalone framework. The single assumption to be made, as mentioned in constraints, is that certain permissions may be needed from the command line. These are assumed to be available for all users.

3 Specific Requirements

(by Vu Phan)

3.1 Interface

- The end-users are *Alice*, *Bob*, and *Eve*.
- *Alice* wants to send a confidential message to *Bob*.
- *Eve* wants to eavesdrop that message.
- These end-users invoke their downloaded `Cry` binaries using command-line shells.

3.2 Performance

Minimum hardware:

RAM	4 GB
CPU	1.5 GHz

3.2.1 Key Generation

(by *Bob*, the receiver)

Input:

```
$ cry generatekeys -cryptosystem=<cryptosystem>
```

Output:

```
The public & private keys are <public key> & <private key>
(took <key-generation time>).
```

Requirements:

- `<cryptosystem>` is the name of an available cryptosystem, such as `rsa`
- `<public key>` and `<private key>` are integers
- `<key-generation time>` (how long the command takes) shall be less than 1 minute (with the aforementioned minimum hardware)

3.2.2 Encryption

(by *Alice*, the sender)

Input:

```
$ cry encrypt -cryptosystem=<cryptosystem> \
> -publickey=<public key> -plaintext=<plaintext>
```

Output:

```
The ciphertext is <ciphertext> (took <encryption time>).
```

Requirements:

- `<plaintext>` is an obviously meaningful string, such as "I hate *Eve*."
- `<ciphertext>` is an apparently meaningless string, such as "sdofAOVI29347"
- `<encryption time>` shall be less than 1 minute

3.2.3 Decryption

(by *Bob*, the receiver)

Input:

```
$ cry decrypt -cryptosystem=<cryptosystem> \
> -privatekey=<private key> -ciphertext=<ciphertext>
```

Output:

```
The plaintext is <plaintext> (took <decryption time>).
```

Requirements:

- `<decryption time>` shall be less than 1 minute

3.2.4 Cryptanalysis

(by *Eve*, the eavesdropper)

Input:

```
$ cry cryptanalyze -cryptosystem=<cryptosystem> \
> -publickey=<public key> -ciphertext=<ciphertext>
```

Output:

```
The plaintext is <plaintext> (took <cryptanalysis time>).
```

Requirements:

- `<cryptanalysis time>` shall be more than 1 day

3.3 Classes

3.3.1 `cryptosystem/`

`cryptosystem.h`

```
#ifndef CRYPTOSYSTEM_CRYPTOSYSTEM_H_
#define CRYPTOSYSTEM_CRYPTOSYSTEM_H_

////////////////////////////////////////////////////////////

enum EnumeratedCryptosystem {rsa}; // more to come

////////////////////////////////////////////////////////////

using IntPtr = mpz_t; // GNU Multiple Precision Integer Type
using Key = IntPtr;
using Text = IntPtr;

////////////////////////////////////////////////////////////

class Cryptosystem {
public:
  virtual void generateKeys(Key publicKey, Key privateKey); // set these

  virtual void encrypt(Text ciphertext, // set this
    const Text plaintext, const Key publicKey);

  virtual void decrypt(Text plaintext, // set this
    const Text ciphertext, const Key privateKey);

  virtual void cryptanalyze(Text plaintext, // set this
    const Text ciphertext, const Key publicKey);
};

////////////////////////////////////////////////////////////
```

```
#endif // CRYPTOSYSTEM_CRYPTOSYSTEM_H_
```

rsa.h

```
#ifndef CRYPTOSYSTEM_RSA_H_
#define CRYPTOSYSTEM_RSA_H_

////////////////////////////////////////////////////////////

#include "cryptosystem.h"

////////////////////////////////////////////////////////////

class Rsa : public Cryptosystem {
public:
  void generateKeys(Key publicKey, Key privateKey); // set these

  void encrypt(Text ciphertext, // set this
    const Text plaintext, const Key publicKey);

  void decrypt(Text plaintext, // set this
    const Text ciphertext, const Key privateKey);

  void cryptanalyze(Text plaintext, // set this
    const Text ciphertext, const Key publicKey);
};

////////////////////////////////////////////////////////////

#endif // CRYPTOSYSTEM_RSA_H_
```

3.3.2 party/

party.h

```
#ifndef PARTY_PARTY_H_
#define PARTY_PARTY_H_

////////////////////////////////////////////////////////////

#include "../cryptosystem/cryptosystem.h"

////////////////////////////////////////////////////////////

class Party {
public:
  Cryptosystem cryptosystem;
```

```
  Party(EnumeratedCryptosystem enumeratedCryptosystem);
};

////////////////////////////////////////////////////////////

#endif // PARTY_PARTY_H_
```

sender.h

```
#ifndef PARTY_SENDER
#define PARTY_SENDER

////////////////////////////////////////////////////////////

#include "party.h"

////////////////////////////////////////////////////////////

class Sender : public Party {
public:
  Text ciphertext;

private:
  Text plaintext;

  void encrypt(const Key publicKey);
    // {cryptosystem.encrypt(ciphertext, plaintext, publicKey)}
};

////////////////////////////////////////////////////////////

#endif // PARTY_SENDER
```

receiver.h

```
#ifndef PARTY_RECEIVER
#define PARTY_RECEIVER

////////////////////////////////////////////////////////////

#include "party.h"

////////////////////////////////////////////////////////////

class Receiver : public Party {
public:
  Key publicKey;
```

```
private:
  Key privateKey;
  Text plaintext;

  void generateKeys();
    // {cryptosystem.generateKeys(publicKey, privateKey)}

  void decrypt(const Text ciphertext);
    // {cryptosystem.decrypt(plaintext, ciphertext, privateKey)}
};

//////////////////////////////////////////////////////////

#endif // PARTY_RECEIVER
```

eavesdropper.h

```
#ifndef PARTY_EAVESDROPPER
#define PARTY_EAVESDROPPER

//////////////////////////////////////////////////////////

#include "party.h"

//////////////////////////////////////////////////////////

class Eavesdropper : public Party {
private:
  Text plaintext;

  void cryptanalyze(const Text ciphertext, const Key publicKey);
    // {cryptosystem.cryptanalyze(plaintext, ciphertext, publicKey)}
};

//////////////////////////////////////////////////////////

#endif // PARTY_EAVESDROPPER
```

3.3.3 cryptoframework.h

```
#ifndef CRYPTOFRAMEWORK_H_
#define CRYPTOFRAMEWORK_H_

//////////////////////////////////////////////////////////

#include "party/sender.h"
```

```
#include "party/receiver.h"
#include "party/eavesdropper.h"

////////////////////////////////////////////////////////

class Cryptoframework {
public:
  Sender sender;
  Receiver receiver;
  Eavesdropper eavesdropper;

  Cryptoframework(EnumeratedCryptosystem enumeratedCryptosystem);

  void testKeyGeneration();
    // {receiver.generateKeys()}

  void testEncryption();
    // {sender.encrypt(receiver.publicKey)}

  void testDecryption();
    // {receiver.decrypt(sender.ciphertext)}

  void testCryptanalysis();
    // {eavesdropper.cryptanalyze(sender.ciphertext, receiver.publicKey)}
};

////////////////////////////////////////////////////////

#endif // CRYPTOFRAMEWORK_H_
```