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

Daniel Dunning, Michael Degraw, Vu Phan

2017-02-13

Contents

1 Introduction

Michael Degraw

1.1 Purpose

Please write here.

1.2 Scope

Please write here.

1.3 Definitions, acronyms, and abbreviations

Please write here.

1.4 References

Please write here.

1.5 Overview

Please write here.

2 Overall description

Daniel Dunning

2.1 Product perspective

`Cry` will be implemented as a stand-alone framework, with built in libraries updated as needed. No other frameworks or applications are needed to run cry: it is a standalone application. It will be used as a tool for cryptographers and developers alike, and all systems will be self-contained in the framework.

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 will be able to develope new cryptosystems
- Users will be able to develope new eavesdropping methods.
- Users will be able to develope new methods for breaking a cryptosystem.
- Using a combination of cryptosystems, eavesdropping, and breaking, users will have the ability to test systems' security.

2.2.2 Reporting

- Upon performing a test, users will receive feedback on their methods.
- Reports will be shown to suggest the security of the cryptosystem and to give helpful feedback in the area of weaknesses in the cryptosystem.

2.3 User characteristics

User of `Cry` will most likely have a medium to high level of experience with cryptosystems. This is not a requirement, it is an open application. However, `Cry` aims to aid the developing cryptosystems, and this is implies a high level of experience. If more complicated libraries are implemented, users will need a high level of experience to uderstand these methods and therefore apply them.

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

Vu Phan

3.1 External interface requirements

- 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.
- Note: the command-line arguments below will be explained later.

3.1.1 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>).
```

3.1.2 Encryption

(by *Alice*, the sender)

Input:

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

Output:

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

3.1.3 Decryption

(by *Bob*, the receiver)

Input:

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

Output:

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

3.1.4 Cryptanalysis

(by *Eve*, the eavesdropper)

Input:

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

Output:

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

3.2 Performance requirements

3.3 Design constraints

3.4 Software system attributes

3.5 Other requirements

3.6 Classes

3.6.1 `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_
```

3.6.2 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
```

3.6.3 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_
```