`Cry` – Project 1 (Project Plan): Report

Daniel Dunning, Michael Degraw, Vu Phan

2017-01-30

Contents

1 Motivation

- For end-users:
  - Problem: An end-user wants to send and receive secure messages with other end-users.
  - Solution: `Cry` lets end-users establish secure communication via built-in cryptosystems.
- For cryptographers:
  - Problem: As quantum computing becomes more prevalent, we need new cryptosystems that are secure traditionally as well as in the quantum world.
    * Reason: Existing cryptosystems are secure only if their assumptions are true.
    * Example: The RSA cryptosystem's security relies on the assumption that integer factorization is hard. It is currenly unknown whether a polynomial-time factorizing algorithm

    exists for traditional binary computers. A polynomial-time factorizing algorithm does, however, exist for quantum computers.
    * Example: The DES cryptosystem's security relied on the assumption that it was not practical to attack by brute force its key space (approximately 72 quadrillion keys). In 1998, the EFF (Electronic Frontier Foundation) built Deep Crack at a cost of approximately 250,000 USD (a very affordable fee), which successfully brute forced DES in 56 hours.
    * Example: The AES cryptosystem's security also relies on the assumption that it is not practical to attack by brute force its key space. This assumption should prove to be correct for the forseable future. AES when implemented correctly is not susceptible to brute force attacks, nor is it susceptible to any known cryptanalysis techniques.
  - Solution: `Cry` allows cryptographers to easily prototype, test, and benchmark their new cryptosystems.

References:

- Wikipedia [2017c]
- Wikipedia [2017a]
- Wikipedia [2017e]

2 Overview

2.1 Example Use Case

Situation:

- *Alice* (sender) wants to confidentially send a message to *Bob* (receiver).
- *Eve* (eavesdropper) wants to know that message.

Procedure:

1. Each person downloads the binary file `cry` of the `Cry` cryptographic framework.

2. *Bob* publishes his choice of cryptosystem: RSA (Rivest, Shamir, Adleman).

3. *Bob* generates his keys:

```
$ cry getkeys --cryptosystem=rsa
The public & private keys are 825 & 637 (took 1 second).
```

4. *Bob* publishes his public key (and hides his private key).

5. *Alice* obtains *Bob*'s published public key.

6. *Alice* encrypts her message (say, her phone number):

```
$ cry encrypt --cryptosystem=rsa \
> --public-key=825 --plaintext=4692301804
The ciphertext is 1110003333 (took 1 second).
```

7. *Alice* publishes the encrypted message.

8. *Bob* obtains *Alice*'s published encrypted message.

9. *Bob* easily decrypts the message with his private key:

```
$ cry decrypt --cryptosystem=rsa \
> --private-key=637 --ciphertext=1110003333
The plaintext is 4692301804 (took 1 second).
```

10. *Eve* struggles to eavesdrop the message without *Bob*'s private key:

```
$ cry eavesdrop --cryptosystem=rsa \
> --public-key=825 --ciphertext=1110003333
The plaintext is 4692301804 (took 1 century).
```

2.2 Functionality

In the previous example:

- `Cry` is the cryptographic framework.
- RSA is a cryptosystem implemented in `Cry`.
- The key-generation, encryption, decryption, and eavesdropping algorithms are specific to RSA.

In general, with `Cry`:

- an end-user can use an implemented cryptosystem to confidentially send and receive messages with others.
- a cryptographer can:
  - prototype her own cryptosystems where the cryptographic algorithms are either newly defined or reused from different existing cryptosystems.
  - test her cryptosystems for security and performance.

References:

- Wikipedia [2017b]
- Wikipedia [2017d]

3 Expectation

- Usability:
  - For end-users: `Cry` enables end-users to establish secure communication via simple command-line options.
  - For cryptographers:
    - Being a modular object-oriented framework, `Cry` lets cryptographers easily define new cryptosystems.
    - `Cry` also reports the security measures of these cryptosystems.
- Limitation of `Cry` compared to the existing `Charm` cryptographic framework:
  - `Charm` is used for anaylizing cryptosystems implemented in Python.
  - `Charm` has advanced functionalities that we will be unable to provide in `Cry` due to time constraints.
  - `Cry` will have a similar base for sandboxing and prototyping cryptosystems.

References:

- Akinyele et al. [2013]

References

Joseph A. Akinyele, Christina Garman, Ian Miers, Matthew W. Pagano, Michael Rushanan, Matthew Green, and Aviel D. Rubin. Charm: a framework for rapidly prototyping cryptosystems. *Journal of Cryptographic Engineering*, 3(2):111–128, 2013. ISSN 2190-8508. doi: 10.1007/s13389-013-0057-3. URL http://dx.doi.org/10.1007/s13389-013-0057-3.

Wikipedia. Advanced Encryption Standard — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Advanced%20Encryption%20Standard&oldid=760970688, 2017a. [Online; accessed 29-January-2017].

Wikipedia. Cryptosystem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Cryptosystem&oldid=708378805, 2017b. [Online; accessed 29-January-2017].

Wikipedia. EFF DES cracker — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=EFF%20DES%20cracker&oldid=733831486, 2017c. [Online; accessed 29-January-2017].

Wikipedia. RSA (cryptosystem) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=RSA%20(cryptosystem)&oldid=762416482, 2017d. [Online; accessed 29-January-2017].

Wikipedia. Shor's algorithm — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Shor's%20algorithm&oldid=759584308, 2017e. [Online; accessed 29-January-2017].