Classical Cryptography

Tobias Innleggen, Andráas Zsolt Süutő, Azim Kazimli

Tuesday 18th March, 2025

Abstract

This abstract still needs to be rewritten so this is a placeholder: This is project explores the domain of classical cryptography by implementing and analyzing a selection of historical ciphers using the Haskell programming language. Our primary objective is to develop functional encoders and decoders for three classical ciphers: the Caesar cipher (or substitution cipher), the Vigenère cipher, and the Playfair cipher, with the potential to expand this list based on time availability. Beyond implementation, we aim to design an automated cipher recognition system capable of identifying these encryption methods from ciphertext samples. Additionally, we will construct tools to automatically crack the implemented ciphers, leveraging cryptanalytic techniques to expose their vulnerabilities. Should time permit, we will apply these tools to solve cryptography assignments provided in the project description. This work seeks to deepen our understanding of classical cryptographic systems while showcasing Haskell's suitability for such implementations and analyses.

Contents

1 Introduction

Here we talk about theory mostly, maybe cite some things like [Lam94].

2 Implementing classical cryptography ciphers

Some more theory, some more citing like [Lam94].

2.1 Substitution cipher

Explain code of cipher implementation in codeblocks, unhide important things from below

```
{-# LANGUAGE TypeSynonymInstances #-}
{-# LANGUAGE FlexibleInstances #-}
{-# LANGUAGE InstanceSigs #-}
module Substitution where
```

2.2 Vignere cipher

Explain code of cipher implementation in codeblocks, unhide important things from below

```
module Vignere where
```

2.3 Playfair cipher

Explain code of cipher implementation in codeblocks, unhide important things from below

```
module Playfair where
```

3 Cipher recognition

This section is about cypher recognition.

4 Attacks

In this section we perform attacks on the ciphers implemented in 2.

```
module Main where

import Vignere
import Substitution
import Playfair
```

```
import PlayfairBreaker

main :: IO ()
main = do
  putStrLn "Hello!"
```

We can show the output with

```
Hello!
Output, statistics, timings, encoded decoded results whatever:
[1,2,3,4,5,6,7,8,9,10]
[100,100,300,300,500,500,700,700,900,900]
[1,3,0,1,1,2,8,0,6,4]
[100,300,42,100,100,100,700,42,500,300]
GoodBye
```

5 Tests

We now use the library QuickCheck to randomly generate input for the ciphers, and test whether they work correctly. We can also use QuickCheck to test attacks maybe?

```
module Tests where

import Vignere
import Substitution
import Playfair

import Test.Hspec
import Test.QuickCheck

main :: IO ()
main = hspec $ do
  describe "Basics" $ do
    it "somenumbers should be the same as [1..10]" $
      somenumbers `shouldBe` [1..10]
    it "if n > - then funnyfunction n > 0" $
      property (\n -> n > 0 ==> funnyfunction n > 0)
    it "myreverse: using it twice gives back the same list" $
      property $ \str -> myreverse (myreverse str) == (str::String)
```

6 Conclusion

Here we concluded what has to be concluded.

References

[Lam94] Leslie Lamport. Latex: A document preparation system. *TUGboat*, 15(3):305–307, 1994.