# Title

Kent Odde

October 16, 2020

# Contents

# Abstract

This is my submission for the home exam in Discrete Mathmatics, fall 2020. The assignment consists of implementing the ElGamal encryption algorithm, in a programming language of my own choosing.

Quite a bit of overhead were required to encrypt larger inputs. Because this course is concerned with the mathematics behind ElGamal and to satisfy the professors requirement for a short paper, I will not cover this part of the implementation to a detailed extent.

# Introduction

# Implementation

## Finding and choosing a generator

$$Y = g^x \bmod p \tag{1}$$

For every value of x, we should get a unique value of Y

All generators are factors of p-1

## Overhead

## Padding

PKCS#1.5

Asymmetric encryption algorithms like RSA and ElGamal, requires a padding scheme in order to increase the security. A common padding scheme for RSA is PKCS#1.5. As it was hard to find sources for padding schemes commonly used with ElGamal, my choice also fell on PKCS for this implementation.

A visualization of the scheme can be seen in figure XX.

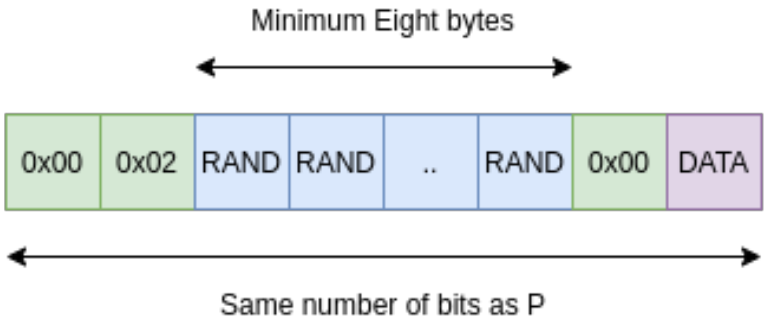


Figure 1: PKCS padding scheme

## Block Handling

ElGamal is not a common choice for encrypting larger texts, however after conversations with the lecturer, I understood that this was a requirement for this assignment. To keep the overhead to a minimum ECB(Electronic Codebook) was chosen. An illustration of ECB can be seen in figure XX. If the plaintext is larger than the number of bits of P, we divide it up into blocks, encrypt them seperately, and concatenate them.

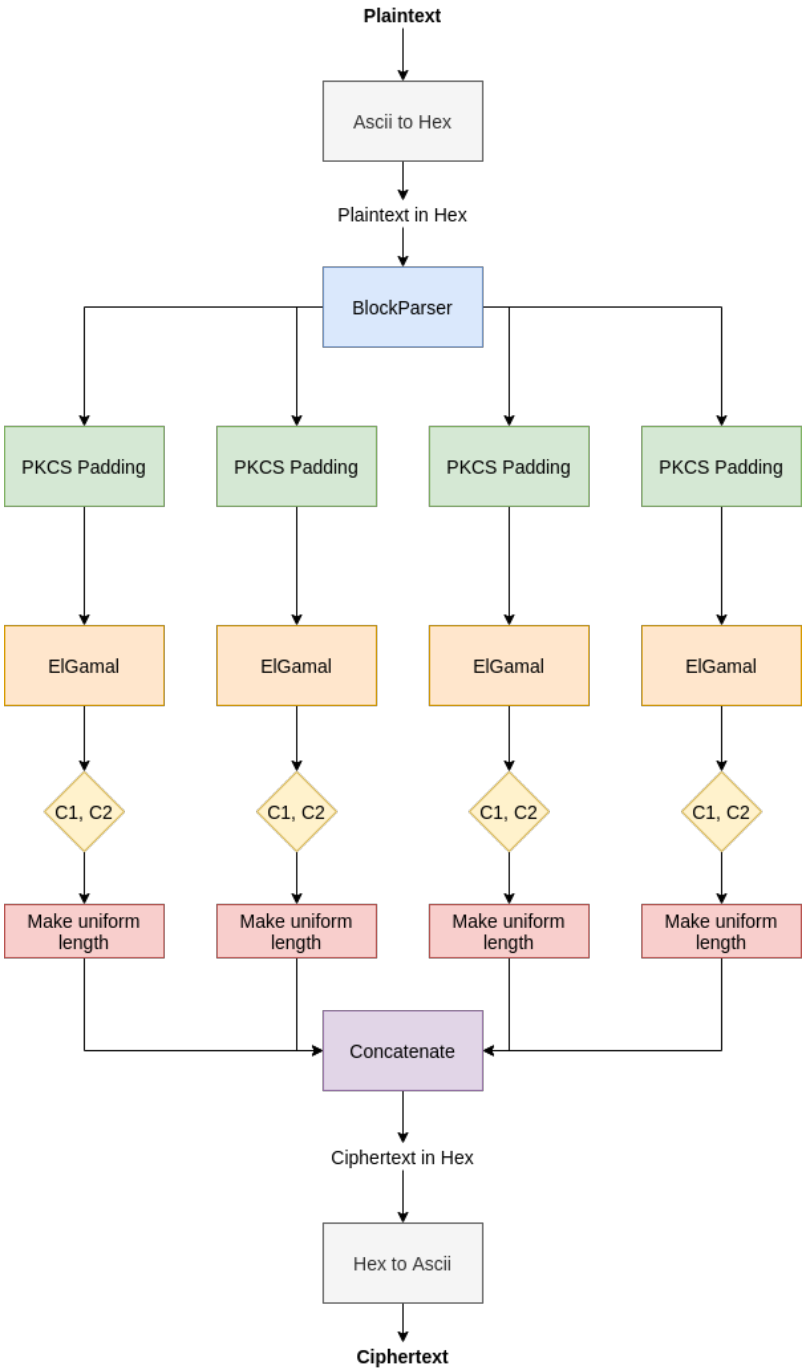


Figure 2: PKCS padding scheme

An extra complication with ElGamal is that the algorithm produces two numbers, in which the size of bits may vary. In order to be able to parse the concatenated strings later, I decided to prepend zeroes to c1 and c2, until their length were at 512 bits.

## Conclusion

Figure 3: ..

## Appendices