User Documentation for RSA Key Generation, Encryption, and Decryption Modules

This documentation provides detailed guidance on using three Haskell modules for RSA key generation, message encryption, and message decryption. The modules covered are:

1. `GenerateKeys.hs`
2. `EncryptMessage.hs`
3. `DecryptMessage.hs`

1. GenerateKeys.hs

This module is responsible for generating RSA public and private keys. The main functions are designed to generate large prime numbers, calculate the keys, and check their safety against Wiener's attack.

Functions

`millerRabinPrimality :: Integer -> IO Bool`

Determines if a given number is prime using the Miller-Rabin primality test.

- **Parameters:**
  - `n`: Integer to be tested for primality.
- **Returns:**
  - `True` if `n` is prime, `False` otherwise.

`generatePrimes :: IO (Integer, Integer)`

Generates two distinct large prime numbers.

`generatePrime :: IO Integer`

Generates a single large prime number within a specified range.

`calculateKeys :: IO (Integer, Integer, Integer)`

Calculates the public and private keys by generating two primes, computing n, ϕ, e, and d.

`calculateNAndPhi :: Integer -> Integer -> (Integer, Integer)`

Calculates n (the modulus) and ϕ (the totient) given two primes p and q.

`calculateE :: Integer -> Integer`

Finds a suitable public exponent e that is coprime with ϕ.

`calculateD :: Integer -> Integer -> Integer`

Calculates the private exponent d using the Extended Euclidean Algorithm.

`extendedEuclidean :: Integer -> Integer -> (Integer, Integer, Integer)`

Performs the Extended Euclidean Algorithm to find the greatest common divisor and coefficients.

`isSafeAgainstWiener :: Integer -> Integer -> Bool`

Checks if the keys are safe against Wiener's attack.

`squareRoot :: Integer -> Integer`

Calculates the integer square root of a number.

Main Program

The `main` function generates the keys, prints the public and private keys, and checks their safety against Wiener's attack.

Example Usage

Compile and run `GenerateKeys.hs`:

Output:

Generating keys...

Keys generated

Your public key (n, e):
(22000746718227297823159580811702284418487261876889141219101359894 7, 3)

Your private key (n, d):
(22000746718227297823159580811702284418487261876889141219101359894 7,
14667164478818198548773053874468126247998196334538076522576109898 7)

Your keys are safe against Wiener's attack

2. EncryptMessage.hs

This module encrypts a message using a given RSA public key.

Functions

`encryptWithPublicKey :: Integer -> Integer -> Integer -> Integer`

Encrypts a message *m* using the public key components *n* and *e*.

`asciiStringToInt :: String -> Integer`

Converts an ASCII string to an integer.

Main Program

The `main` function prompts the user for a public key and a message, then encrypts the message and displays the encrypted message as an integer.

Example Usage

Enter your public key (n e):

220007467182272978231595808117022844184872618768891412191013598947 3

Enter the message:

"Ahoj ako sa dnes mas?"

Encrypted message:
24225208958865814749064867162654809109452882237648659499172173003

3. DecryptMessage.hs

This module decrypts a message using a given RSA private key.

Functions

`exponentWithMod :: Integer -> Integer -> Integer -> Integer`

Performs modular exponentiation.

`decryptWithPrivateKey :: Integer -> Integer -> Integer -> Integer`

Decrypts a message *c* using the private key components *n* and *d*.

`intToAsciiString :: Integer -> String`

Converts an integer back to an ASCII string.

Main Program

The `main` function prompts the user for a private key and an encrypted message, then decrypts the message and displays the original message.

Example Usage

Enter your private key (n d):

220007467182272978231595808117022844184872618768891412191013598947
146671644788181985487730538744681262479981963345380765225761098987

Enter the encrypted message (as an integer):

242252089588658147490648671626548091094528822376486594991721730003

Decrypted message: "Ahoj ako sa dnes mas?"