RSA Public and Private Key Generation

Jason Pearson and Sam Demorest

April 20, 2015

Design Decisions

Used Beta Rust as our programming Language
Two Programs for Easy Reuse-ability
Miller-Rabin and AKS for Primality Testing
Public and Private key Generation
Encryption and Decryption of Characters

Implimentation

Miller-Rabin instead of AKS for speed
Modular exponentiation
ASCII Representation of Character for Encrypting
Used an Inverse function instead of Extended Euclidean Algorithm

Output

```
C:\Users\BuckDich\Documents\GitHub\rsa_keygen>cargo run 8
     Running `target\rsa_keygen 8`
Value of p = 151
Value of q = 157
Value of n = 23707
Value of totient = 23400
Value of e = 181
Value of d = 224821
Please Insert Letter To Encrypt
C
You sent in C

Number before mod 67
Encryption Time = 15618
Decryption Time = C

C:\Users\BuckDich\Documents\GitHub\rsa_keygen>
```

Conclusion

Able to determine large prime numbers
Able to encrypt one character and decrypt it

BigInt library not fully ready for deployment yet
No use of bit twiddling operations for optimization

Demo!

Demo Time!

References

AKS References:
http://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf
http://mathworld.wolfram.com/AKSPrimalityTest.html

Rust References:
https://doc.rust-lang.org/
http://rustbyexample.com/
https://github.com/rust-lang/rust

RSA Cryptosystem References:
http://mathworld.wolfram.com/RSAEncryption.html
https://engineering.purdue.edu/kak/compsec/NewLectures/Lecture12.pd