RSA Public and Private Key Generation

Jason Pearson and Sam Demorest

April 22, 2015

Design Decisions

Used the in beta programming language Rust
Two separate programs reuse-ability
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 Elgorithm

Output

```
sysadmin@login02:~/rsa_keygen$ cargo run 8
     Running `target/debug/rsa_keygen 8`
Value of p = 149
Value of q = 199
Value of n = 29651
Value of totient = 29304
Value of e = 193
Value of d = 28393
Please Insert Letter To Encrypt
1c
You sent in 1c

Character: 49
Character: 99
Number before mod 25393
Encryption Time = 3947
Into decrypt: 3947
After mod in decrypt: 25393
Decryption Time = 1c
sysadmin@login02:~/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