Implementing RSA and DES Cryptographic Algorithms in an Academic Environment

Using the Python Programming Language

Zach Gilmer
Missouri State University
113 N. Fort St.
Nixa, MO 65714
Gilmer000@live.missouristate.edu

James Hibben
Missouri State University
419 W. Loren St.
Springfield, MO 65807
Hibben87@live.missouristate.edu

ABSTRACT

Implementing a cryptographic algorithm is generally considered to be a difficult task that should be left to the experts. While studying computer security, however, it is important to understand the complexity behind them, as even implementing a simple version of any particular algorithm is a challenge. As such, we elected to implement a simple (and insecure) RSA asymmetric-key algorithm and a DES symmetric-key algorithm.

Categories and Subject Descriptors

H.4 [**Cryptography**]: Computer Science; D.2.8 [**Software Engineering**]: Academic

Keywords

RSA, DES

1. BACKGROUND AND REVIEW

As we move farther into the Digital Age, computer security is becoming more and more important for ensuring the the safety and validity of our data, whether we store it locally or remotely. In large part because of this, it is equally important that we, as computer scientists, have a better understanding of digital cryptography; the amount of work that must go into each algorithm is staggering. Though the algorithms that we implemented for this project are very basic and cannot be considered secure, their key generation algorithms remain incredibly complicated and can be challenging to implement without help.

In writing the code for our algorithms, we referenced Tutorials Point's Data Encryption Standard tutorial for our DES implementation and gist.github.com/avalonalex/2122098 as a base for our RSA algorithm; in each implementation, some of the code was modified or removed, but the output is the same.

2. IMPLEMENTING DES

DES or data encryption standard is a symmetric key block cipher. It uses a 64 bit key and only 56 of the bits are used for key generation. DES is now considered insecure, mainly due to the small key size used. As such it is no longer considered a standard for use.

2.1 Key generation

DES creates sixteen 48-bit keys used for each round of encryption from the 56-bit cipher key. The 64-bit key is put through a permuted choice to gain the 56-bit key used for sub-key generation. Then the key is split and bits are chosen from each half for the 48 bit key. Then bits in each side are shifted a specified amount for each round, and a new key is selected.

2.2 Encryption and Decryption

The 64 bit block is split into two sides and each 32-bit side is expanded to match the 48-bit sub-key length. Each half then goes through an XOR operation with the sub-key. Then the output is split into 6-bit sections and run through substitutions that replace each 6-bits with corresponding 4-bit replacements. The output from the substitutions is then run through a final fixed permutation. The process is repeated in reverse with the same key for decryption.

3. IMPLEMENTING RSA

While RSA is a simple asymmetric-key algorithm to implement, the complexity and computations necessary to run also make it a time-consuming algorithm to use; even generating keys can take an extraordinary amount of time. The algorithms used to generate keys run in $O(n)$ time, but are typically used in conjunction with large-magnitude keys; key sizes on the order of 10^{100} are not uncommon, and often take several seconds to generate on modern processors. Encryption and decryption, however, are significantly more time-consuming when increasing the size of the input; part of the encryption and decryption process runs in $O(n^2)$ time.

3.1 Key Generation

When generating RSA keys, three keys in total are needed: Public keys (n, e), used for encryption, and the private key d, used for decryption. n is usually generated first by randomly generating prime integers p and q; these are multiplied together, then applied using Euler's totient function $\phi((p-1)(q-1))$; this totient is then used as a base from which the next prime integers, e and d are generated. Both e and d must be less than the totient, and one is selected to have the additional requirement of being coprime to the totient; for our implementation, we selected e as the coprime. Once e has been generated, then d can be generated using the modular multiplicative inverse function:

$$e \equiv d^{-1} \pmod{\phi(n)}$$

.

3.2 Encryption and Decryption

In theory- and even some implementations, given the appropriate libraries- encryption and decryption work very simply; the ciphertext is generated by using the public key (n, e) of the recipient on the original plaintext: $c \equiv m^e \mod n$, where m is the padded plaintext. Similarly, decryption is achieved by $m \equiv c^e \mod n$. However, in practice, encryption and decryption are likely to involve breaking apart the text into smaller chunks before encryption, padding any chunk that is smaller than the block size (usually just the last chunk). As the algorithm primarily deals with modifying integers, it is also likely that, for languages that do not implicitly treat characters as integers (such as Python) that the text will need to be converted prior to applying the RSA algorithm.

4. CONCLUSIONS

This project was very educational in regards to exposing some of the challenges of writing cryptographic algorithms. One of the main challenges that we ran into was in attempting to cut out some of the excess from our RSA implementation, as it was not long before we realized that some of the design choices were made due to minor differences in behavior that can determine the success or failure of the algorithm. We found that one of the most important aspects of implementing any security algorithm- aside from ensuring the algorithm's secrecy- is a willingness to deeply study the language to learn the nuances of similar methods. It can easily be these small differences that result in a more secure system.

5. REFERENCES

[1] J. Changra and D. Prima. Implementation of sub key generation algorithms. *International Journal of Advance Research in Computer Science and Management Studies*, 2(4):164âĂŞ168, Apr 2014.