Techniques of Asymmetric File Encryption

Alvin Li
Thomas Jefferson High School
For Science and Technology
Computer Systems Lab

Abstract

As more and more people are linking to the Internet, threats to the security and privacy of information continue to grow. To solve this growing problem, encryption programs have been created to protect privacy during a transfer of files and to make sure that sensitive files will be protected. My project is to create an asymmetric file encryption program. This means that encrypted files will need a pass-key to open that will be different from the key used to encrypt. This program could be applied practically to protect files during transfers.

Introduction and Background

Over the summer, I read a book that explained the history of encryption. Since before the times of Caesar and the Roman empire, encryption has been used to keep secrets secret. In the modern world, file encryption is now used almost everywhere. Whether you are transferring files, compressing files, or formatting databases, you must use modern file encryption techniques. There is so much private information, such as social security numbers, credit card numbers, bank-account information, or private correspondence that all needs to be protected.

In order to fully understand my project, you must know a little about encryption first. There are many categories of encryption, such as key encryption, block encryption, and stream encryption. The one that I will focus on, since it is the category of my algorithm, is key encryption. The basic concept behind key encryption is simple. The user runs the algorithm on the desired file and the file becomes encrypted with the key. In order to decrypt the file, the user must input the correct password, which translates to the correct key. This method of encryption is very powerful because the algorithm is dependent on the key, which is arbitrarily defined. Since each key is unique, each encryption is unique as well. If we used a set, standard algorithm, as soon as a hacker figured out the algorithm, he could break every code created by that algorithm. But, with a good key encryption program, knowing the algorithm does not help. You must have the correct key.

The ultimate goal of any key encryption program is to make the encrypted file impossible to decode without the key. This will force the intruder to use the brute force method, which is to try every single possible pass-key. This is highly difficult to do, because even with a super computer, the number of permutations is so great, that it would take years to finish. So, with a good key encryption program, you can create practically unbreakable codes.

Key encryption algorithms are separated into two groups as well. There are public key and symmetric key encryption programs. There are also two categories of keys, asymmetric and symmetric keys. When using symmetric keys, the key used to encrypt is the same key used to decrypt the file. So there is only one key. Symmetric key encryption uses this form of key. The public key encryption uses different keys to encrypt and decrypt. The key used to encrypt is called the public key. The private key, provided by the user or the computer the file is being transfered to, is used to encrypt the public key. When decrypting, the private key accesses the public key, which decrypts the file. The keys are said to be asymmetric.

File encryption is commercially used in almost every widely used application. Research into this subject is every extensive. Programmers are constantly inventing stronger, faster, and more effective algorithms. Some of these are very efficient, able to not only encrypt, but also compress files. The military also has a large interest in this field. Although their algorithms are generally much safer, they are slower and harder to implement.

Some of the popular encryption algorithms I have studied include RSA, DES, and AES. The RSA is a classic key encryption program created in 1977. It uses very large prime numbers and factoring as its public and private key. Since factoring multiples of large primes is near impossible, this method is very safe and easy to use. It is the most popular form of key encryption used today. The AES or Advanced Encryption Standard, also known as Rijndael, is considered the strongest algorithm to date. So far, no one has found a way to easily crack it.

The main task of my project is to create a encryption program. Specifically, the program that will take a file as an input, and create an encrypted copy of the file in an executable form. When the encrypted file is run, there will be an input asking for a pass-key. If the user inputs the correct pass-key, the file will self-decrypted and transform back to the original file. My program will use a block cipher method of encryption. This means that data will be encrypted in 16, 32, or 128 bit blocks. This method is faster than taking

every bit, as in a stream cipher. I expect to get a version of this program running and put my finished product on the Internet for download.

In writing the program, I will use aspects of some popular existing algorithms. By using a combination of methods, I hope to create an algorithm that is stronger than any.

Procedure and Development

Stage 1: Planning

I plan to work on this project in a series of iterations. Each iteration will build off the previous one by adding a function or revising an algorithm. After each iteration is complete, I will thoroughly test the program to see if everything is working. This way, I can easily pinpoint bugs in the program. There will be two goals for the final version of the program. First, the program must be able to encrypt and decrypt a file. The decrypted file must be an exact copy of the file before encryption. This will be easily determined by examining both files. Second, the encrypted file must be very difficult to crack. This could be determined by letting hackers try to decipher the code. If it is very difficult to decode, then this goal will be achieved.

The only tool I will need for this project will be a computer. The program will be written in C++. If the two criteria listed above are met, then the project can be considered a success. I plan to post a free version of the program on my website for download. This way, people can use the program to securely transfer files over the Internet without interference from outside parties.

Stage 2: First Iteration

I have used a basic RSA algorithm as the basis for my project. The theory behind RSA is simple, but the implementation is difficult.

RSA was invented in 1977. It uses the fact that products of large primes are very hard to factor to its full advantage. Lets say you have two large primes, p and q. The

public key would be the number p * q. This key is used to encrypt the message. The private key, would be either p or q. It is near impossible to derive p and q from p * q because factoring p * q is very difficult. If p* q is a 128 bits large, then you would need to try dividing p * q by 2^64 numbers to find p or q. This is because you need to try at most root n numbers to factor n. Naturally, trying to solve the cipher by brute force would take practically forever. This is what makes RSA so strong, the fact that it is pretty much unbreakable. With the private key p, you can decrypt the original message. The operations below shows how this is done.

*public key n = p * q*
private key large primes p and q
Let e be a random encryption exponent that is less than n and has no factors in common with (p - 1) or (q - 1)
*Calculate the decryption exponent d which satisfies e * d mod (p - 1) * (q - 1) = 1*
Then, the encryption function is E(m) = m ^ e mod n, for any message m
The decryption function is D(c) = c ^ d mod n, for any ciphertext c

This is the basic procedure for my RSA encryption program.

While RSA is relatively simple to code, it is a very strong encryption method. The most difficult problem I had to deal with so far is determining whether a number is a prime or not. If the number is close to a prime, meaning it has two or three divisors, it would not matter that much to the program. The only problem it would create would be in the cryptoanalysis of the ciphertext. If there is more than two divisors of n, another pair of divisors might be mistaken for the private key. This would probably just throw off the hacker for a bit.

I have tested my program in a very simple way. The first standard of my testing was to be able to convert cleartext to ciphertext and back with ease. After this was achieved, I attempted to find clearly visible patterns within the ciphertext. I use cryptoanalysis methods to locate patterns, but after much testing, I could not break the code. The indicates that the code cannot be cracked using basic cryptoanalysis techniques.

The next step in my project development will be to create a more friendly interface. Right now, since the program is text based, it cannot be easily accessed.

There were many obstacles to the development of my project. Many programming errors and miscalculations occurred. As a result, progress was very slow. Just getting the program to encrypt files was took a very long time.

The main problem that I still have not solved is decrypting the cryptext generated by my RSA implementation. When the cryptext is run through the decryption program, the resultant file is gibberish. This is a serious problem because without a working decryption component to my program, it is unusable.

Results and Conclusion

Let me discuss the results of my project. As a result of numerous programming complications, progress has been very slow. The final program is capable of generating cryptext from cleartext. The cryptext is very random, with ASCII values approximately randomly distributed throughout the encrypted text. I came to this conclusion by running tests on the encrypted data.

Also, the encrypted text is completely patternless, which is another essential component. Among the numerous ways that codes can be broken are frequency analysis of characters, and finding patterns within the data. Frequency analysis is used to break translation table ciphers. Because the frequency of some characters in the English language occur more often than others (such as the letter 'e' and 's'), if each letter is associated with only one other character, a cryptoanalyst can guess much of the translation table. This makes the code pretty much worthless. Since the frequencies of the ASCII values of the text generated by my program is are randomly distributed, the method of cryptoanalysis cannot be used. Also, there are no visible patterns the would give away the encryption method.

Due to a shortage of time, the decryption part of my program has not been completed.

References

For more current, up-to-date information on my project, visit my website at
www.tjhsst.edu/~ali
The following are website URLs of sources I used in researching about file encryption:

www.mycrypto.net/encryption/crypto_algorithms.html
This is a good website for details about existing algorithms.

http://catalog.com/sft/encrypt.html
This site explains many forms of encryption and provides a guide to how to write encryption programs.

http://www.howstuffworks.com/encryption.htm
A comprehensive guide to encryption from howstuffworks.

http://www.ssh.fi/support/cryptography/index.html
Another good website on encryption from the SSH people