Project Title:Encryptor

Abstract:The project Encryptor is an encryption system used to encrypt any text inside a file. It allows the user to select a file in a particular file location on his system and allows him to encrypt the data(text) in the file using RSA Algorithm.

The system generates a set of keys(public and private), upon which the data in the chosen file is encrypted using RSA algorithm, which provides security to the user to send the data without thinking about any intrusion.

Introduction: We are in a world where information is everything and privacy is a myth. Sharing any data on internet, just as it is without any additional security measures might lead to a privacy breach. So using an encryption standard to protect the data from a third person is very much necessary.

RSA is an perfect way to protect the data from unauthorised access. The RSA is hugely popular as its advantage speak for themselves, For instance this encryption standard is freely usable, incurs no licence fees and is not subject to patent restrictions. Added to this come relatively low storage and hardware requirements. The encryption algorithm is uncomplicated and elegant in programming, and is simple to implement.

Detailed System Description: The encryption system lets the user encrypt the data in a file, by specifying the file location.

To increase the ease of use of the system, the system is designed such a way that the user only needs to run the java program after specifying the file location.

RSA (cryptosystem):
RSA is one of the first practical public-key cryptosystems and is widely used for secure data transmission. In such a cryptosystem, the encryption key is public and differs from the decryption key which is kept secret. In RSA, this asymmetry is based on the practical difficulty of factoring the product of two large prime numbers, the factoring problem. RSA is made of the initial letters of the surnames of Ron Rivest, Adi Shamir, and Leonard Adleman, who first publicly described the algorithm in 1977. Clifford Cocks, an English mathematician working for the UK intelligence agency GCHQ, had developed an equivalent system in 1973, but it was not declassified until 1997.

A user of RSA creates and then publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message Breaking RSA encryption is known as the RSA problem; whether it is as hard as the factoring problem remains an open question.

The RSA algorithm involves four steps: key generation, key distribution, encryption and decryption.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers *e*, *d* and *n* such that with modular exponentiation for all integer *m*:

$$(m^e)^d \equiv m \ (mod\ n)$$

and that even knowing *e* and *n* or even *m* it can be extremely difficult to find *d*.

Additionally, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e \equiv m \ (mod\ n)$$

RSA involves a *public key* and a *private key.* The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The public key is represented by the integers *n* and *e*; and, the private key, by the integer *d* (although *n* is also used during the decryption process; so, it might be considered a part of the private key, too). *m* represents the message.

The keys for the RSA algorithm are generated the following way:

1. Choose two distinct prime numbers *p* and *q*.
    - For security purposes, the integers *p* and *q* should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits to make factoring harder. Prime integers can be efficiently found using a primality test.
2. Compute *n* = *pq*.
    - *n* is used as the modulus for both the public and private keys. Its length, usually expressed in bits, is the key length.
3. Compute $\lambda(n) = \mathrm{lcm}(\lambda(p), \lambda(q)) = \mathrm{lcm}(p - 1, q - 1)$, where λ is Carmichael's totient function. This value is kept private.

4. Choose an integer e such that $1 < e < \lambda(n)$ and $\gcd(e, \lambda(n)) = 1$; i.e., e and $\lambda(n)$ are coprime.
5. Determine d as $d \equiv e^{-1} \pmod{\lambda(n)}$; i.e., d is the modular multiplicative inverse of e (modulo $\lambda(n)$).
    - This is more clearly stated as: solve for d given $d \cdot e \equiv 1 \pmod{\lambda(n)}$.
    - e having a short bit-length and small Hamming weight results in more efficient encryption – most commonly $2^{16} + 1 = 65{,}537$. However, much smaller values of e (such as 3) have been shown to be less secure in some settings.
    - e is released as the public key exponent.
    - d is kept as the private key exponent.

The *public key* consists of the modulus n and the public (or encryption) exponent e. The *private key* consists of the modulus n and the private (or decryption) exponent d, which must be kept secret. p, q, and $\lambda(n)$ must also be kept secret because they can be used to calculate d.

Alternatively, as in the original RSA paper, the Euler totient function $\varphi(n) = (p - 1)(q - 1)$ can be used instead of $\lambda(n)$ for calculating the private exponent d. This works because $\varphi(n)$ is always divisible by $\lambda(n)$, and thus any d satisfying $d \cdot e \equiv 1 \pmod{\varphi(n)}$ also satisfies $d \cdot e \equiv 1 \pmod{\lambda(n)}$. However, computing d modulo $\varphi(n)$ will sometimes yield a result that is larger than necessary (i.e. $d > \lambda(n)$). Most RSA implementations will accept exponents generated using either method (if they use the private exponent d at all, rather than using the optimized decryption method based on the Chinese remainder theorem described below), but some standards like FIPS 186-4 may require that $d < \lambda(n)$. Any "oversized" private exponents not meeting that criterion may always be reduced modulo $\lambda(n)$ to obtain a smaller equivalent exponent.

Since any common factors of $(p - 1)$ and $(q - 1)$ are present in the factorisation of $n - 1 = pq - 1 = (p - 1)(q - 1) + (p - 1) + (q - 1)$, it is recommended that $(p - 1)$ and $(q - 1)$ have only very small common factors, if any besides the necessary.

Key distribution

Suppose that Bob wants to send a secret message to Alice. If they decide to use RSA, Bob must know Alice's public key to encrypt the message and, Alice must use her private key to decrypt the message. To enable Bob to send his encrypted messages, Alice transmits her public key (n, e) to Bob via a reliable, but not necessarily secret route. Alice's private key (d), is never distributed.

Encryption

After Bob obtains Alice's public key, he can send a message *M* to Alice.

To do it, he first turns *M* (strictly speaking, the un-padded plaintext) into an integer *m* (strictly speaking, the padded plaintext), such that $0 \le m < n$ by using an agreed-upon reversible protocol known as a padding scheme. He then computes the ciphertext *c*, using Alice's public key *e*, corresponding to

$$c \equiv m^e \pmod{n}$$

This can be done reasonably quickly, even for 500-bit numbers, using modular exponentiation. Bob then transmits *c* to Alice.

Decryption

Alice can recover *m* from *c* by using her private key exponent *d* by computing

$$c^d \equiv (m^e)^d \equiv m \pmod{n}$$

Given *m*, she can recover the original message *M* by reversing the padding scheme.

Requirements: Eclipse IDE

JAVA

Literature Survey: When I decided on this kind of project, I had stuck to few things that are to be followed while developing this program. Such as,

Usage of JAVA and its rich libraries.

Fast and Reliable.

Ease of access.

User Manual: The functionality of the system is pretty simple, yet very effective.

When the user runs the java file, he needs to include the path of the file, which he want to encrypt. The text to be encrypted has to be inputted in the text file.

The java program selects the file, creates the public and private keys, upon which the encryption using RSA is performed. A new file created, which contains the encrypted data of the original file. Also another text file is created, which contains the decrypted format of the encrypted data ie., the original plain text.

Conclusion: The Encryptor provides an easy to use system to a user through which he can encrypt his data to provide security from unauthorised access. It doesn't require much of an infrastructure and will be very effective.

References:

https://en.wikipedia.org/wiki/RSA_(cryptosystem)

https://www.nfon.com/en_de/solutions/resources/glossary/advanced-encryption-standard/

https://eclipse.org/

https://docs.oracle.com/javase/7/docs/api/

https://www.java.com/en/