CPE 593- Applied Data Structure & Algorithms

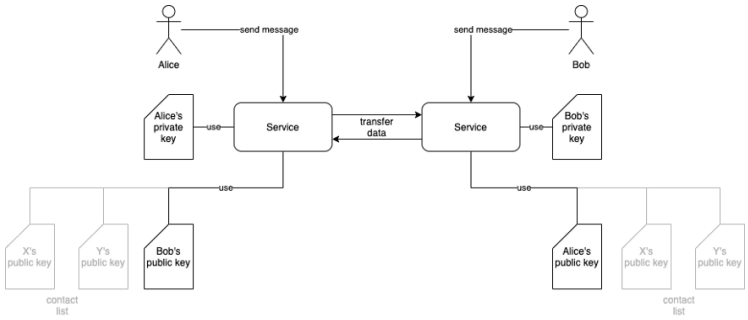
Group Members: Amanda Ly, Ghaith Arar, Neel Haria

Topic: Implement RSA using a biginteger library. It doesn't have to be production quality, but you would need to implement Diffie-Hellman key exchange and use AES-256 (you don't have to write that) to exchange secret messages.

GitHub: [CPE-593_FinalProject-RSA]

Abstract: RSA is perhaps the most widely used encryption public- key (PK) cryptography. There is a high probability that this document was delivered to the reader, at some point, using this algorithm. However, for any cryptography scheme, security is the main incentive; and to implement a secure RSA scheme very large prime numbers are required; therefore the algorithm is computationally intensive. This project will look into various implementations of the RSA algorithms, then it will implement the RSA in an efficient and elegant fashion using C++. The implementation will not only try to implement secure algorithms, but it will also find a very efficient implementation.

Brief Background on RSA:



Under RSA encryption, messages are encrypted with a public key and can only be decrypted by the private key. Each RSA user has a key pair consisting of their public and private keys. RSA encryption is often used in combination with other encryption schemes, or for digital signatures which can prove the authenticity and integrity of a message. It isn't generally used to encrypt entire messages or files, because it is less efficient and more resource-heavy than symmetric-key encryption. To make things more efficient, a file will generally be encrypted with a symmetric-key algorithm, and then the symmetric key will be encrypted with RSA encryption. Under this process, only an entity that has access to the RSA private key will be able to decrypt the symmetric key.

Aspects of RSA Encryption:

- Trapdoor functions: makes it easy to compute in one direction but incredibly hard to reverse
- Generating primes: allows PKs to be shared without endangering the message or revealing the private key
- Carmichael's totient function: used to generate public and private key

Requirements: Here is the minimum requirement to build the service:

1. The private key reader
2. The public key reader
3. Message encryptor
4. Signature writer
5. Message decipher
6. Signature verifier
7. HTTP listener
8. HTTP post