**Outline:**

* Abstract

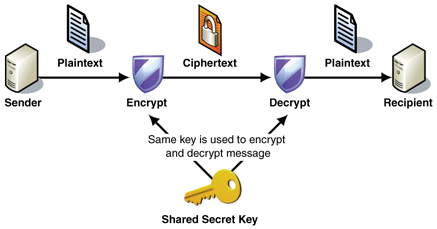
To overcome the problems faced in the symmetric key algorithms, people chose asymmetric key algorithms for communication. Communications with asymmetric algorithms will allow us to transfer information without the key exchange. Public-key cryptography requires two separate keys, one of which is secret, and one of them public. Public-key cryptography is used widely and it is an approach used by many of the encryption algorithms and cryptosystems. It supports such standards as safe Internet transport layer (TLS), PGP and GPG. RSA and the Diffie-Hellman key exchange most widely used public key distribution systems, while the Digital Signature Algorithm is the system most widely used digital signature system.

In this report we are mainly concentrating on some asymmetric algorithms which are mostly used. They are **RSA cryptosystem** and **ElGamal Cryptosystem**.

* Introduction

Cryptography is a fascinating subject. The spread of computers and communication systems require private sector to protect the information in digital form, and to provide security services. Further, Cryptography has a great importance in most technology today, not only in computers but also in credit cards, government and it remains the standard means for securing electronic commerce for many financial institutions around the world.

Cryptography is the science of encoding data, typically using a key, so that people without the key cannot read the data. Many different algorithms are used to encrypt data and they are either symmetric or asymmetric. In symmetric key cryptography, the best known algorithm is the Data Encryption Standard ([DES](http://searchsecurity.techtarget.com/definition/Data-Encryption-Standard)). DES, which was developed at IBM in 1977, was a popular standers algorithm for years, until Triple DES and AES began to replace it. Whereas, symmetric key algorithms require a shared secret to be exchanged between the communicating parties to have a secured communication see Figure 1. The security of the symmetric key algorithm depends on the secrecy of the key. So, the problem with symmetric keys is how to securely get the secret keys to each end of the exchange and keep them secure after that. For this reason, an asymmetric key system is now often used that is known as the public key infrastructure ([PKI](http://searchsecurity.techtarget.com/definition/PKI)). Public-key encryption does not require that you openly exchange a secret key with the recipient of an encrypted message. [[1](#_ENREF_1)]



**Figure 1. The process of symmetric encryption**

Public-key cryptography also referred to as asymmetric encryption, uses a pair of cryptographic keys, a public key and a private key see Figure 2. The private key is kept secret and defines the associated decryption transformation, while the public key defines an encryption transformation and can be distributed openly, thereby negating the need to transmit a secret key in advance. The keys are related mathematically, allowing the sender of a message to encrypt his message using the recipient's public key. The message can then only be decrypted using the recipient's private key. Since encryption transformation is public knowledge, public-key encryption alone does not provide data origin authentication or data integrity. Public-key decryption by using private key can provides authentication guarantees in entity authentication and authenticated key establishment protocols. So, public-key encryption provides both privacy and confidentiality. [[2](#_ENREF_2)]



**Figure 2. The process of asymmetric encryption**

Public-key encryption schemes are simple, elegant and relatively efficient digital signature mechanisms. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart. However they are typically substantially slower than symmetric-key encryption algorithms. For this reason, public-key encryption is most commonly used in practice for the transport of keys subsequently used for bulk data encryption by symmetric algorithms and other applications including data integrity and authentication, and for encrypting small data items such as credit card numbers and PINs. [[2](#_ENREF_2)]

Public-key cryptography being discovered in the mid-1970s and it does not have an extensive history as symmetric-key encryption, but the most familiar public-key algorithms are RSA and Diffie–Hellman key exchange, while the Digital Signature Algorithm is the most widely used digital signature system.

There are several public key algorithms. The first one proposed is the knapsack cryptosystem but it is insecure and an easily understood system. After the knapsack, RSA algorithm is known as the gold standard of public key cryptography that we will focus on it in this report. Then, The Diffie–Hellman key exchange was published in 1976 by Whitfield Diffie and MartinHellman and it provided a common secret key between the communicating parties over an insecure channel.

Also there are more public key algorithms such as El-Gamal Cryptosystem and Elliptic Curve Cryptosystem (ECC),[[3](#_ENREF_3)] but in this report we are mainly focused on some asymmetric algorithms which are mostly used. They are RSA cryptosystem and ElGamal Cryptosystem.

* Literature
* **Detailed RSA Algorithm**

RSA algorithm is used for public key cryptography and digital signatures. It is the most commonly used public key encryption algorithm. The security of the RSA algorithm is based on the difficulty of factoring large numbers n = pq, where p and q are large prime numbers. The RSA algorithm keys are often of length a power of two, like 512, 1024, or 2048 bits, RSA to be secure, the key length has to be at least 1024-bits [[4](#_ENREF_4)] [[5](#_ENREF_5)].

This section will provide the history of RSA, describes the RSA encryption scheme, its security, and provides some Practical Considerations.

* + - **History**

RSA invented by **R**ivest, **S**hamir and **A**dleman in 1977, RSA stands for the first letter in each of its inventors' last names. It was first published in the paper A Method for Obtaining Digital Signatures and Public-Key Cryptosystems, but the National Security Agency (NSA) did not like the idea of the distribution the cryptography source code internationally, and requested to be stopped. However, they continued distribution because the NSA failed to provide a legal basis for their request. [[5](#_ENREF_5)]

* + - **How RSA System works**

RSA system involves three steps: key generation, encryption and decryption. First step: **key generation**, the mathematical details of the algorithm used in obtaining the public and private keys are explained in following concepts.

1. Take two large primes, p and q, and compute their product n = pq, where n is called the modulus.
2. Choose a number, e which is less than n and relatively prime to (p-1)(q-1), that means e and (p-1)(q-1) have no common factors except 1.
3. Find another number d such that (ed - 1) is divisible by (p-1)(q-1). Whereas the values e and d are called the public and private exponents, respectively.

The public key is the pair (n, e); the private key is (n, d). The factors p and q may be destroyed or kept with the private key.

It is currently difficult to obtain the private key d from the public key (n, e). However if one could factor n into p and q, then one could obtain the private key d. Thus the security of the RSA system is based on the assumption that factoring is difficult. The discovery of an easy method of factoring would "break" RSA.

Second step: encryption, the cyphertext C is found by the equation "C = Me mod n" where M is the original message and e is the public or encryption exponent.

Third step: decryption, the message M can be found form the cyphertext C by the equation "M = Cd mod n" where d is the private or decryption exponent.[[5](#_ENREF_5)]

The RSA system can be used for public key cryptography (encryption) and digital signatures as we said before, there is slightly different between them.

**Encryption**

In RSA, encryption keys are public, while the decryption keys are private, so no one can decrypt an encrypted message except the person who has the private key. Everyone has their own public and private keys. The keys must be made in such a way that the private key may not be easily construed from the public encryption key.[[4](#_ENREF_4)]

**Digital signatures**

The recipient may need to verify that the message sent in fact originated from the sender (sign), just did not come from there (authentication). This is done by using the decryption key of the sender, and anyone can use the corresponding public key encryption to verify the signature. Therefore, Signatures cannot be forged and after signed the message no one can deny the site at a later time. [[4](#_ENREF_4)]

* + - **Attacks on RSA Cryptosystem**

The RSA function is a one-way trapdoor function. It is not known to be easily invertible without the trapdoor, d. General attacks on RSA involve trying to compute d, or some other way to get to M (plaintext). The difficulties and the main security of RSA algorithm is based on prime factorization problem.[[6](#_ENREF_6)]

There has been much research and many publications on attacks on the RSA Cryptosystem. We can categorize attacks on RSA into five categories as below, and we will present a summary of them.

1. **Factoring Large Integers.**

Given (C, *e*, N), Marvin can do a brute force search to find M. But factoring N involves a much smaller search space and thus, more efficient to do. Factoring is a well-researched problem. Pomerance [[7](#_ENREF_7)] has an interesting introduction into what clever methods can be done. For example, how to factor 8051? Naïve approach is trying all primes up to square root of 8051, or almost 90. Clever trick is to find a square of primes that subtract to the number, example 90^2 – 7^2 = 8051. Thus (90 – 7) \* (90 + 7) = 8051; 83 & 97 are the factors. However, this trick is only easy if factors are close to square root of the number. There are only a very small set of numbers where this is true. But much more sophisticated methods are known. Currently the general number field sieve is the best known running time with, reported by Boneh [[6](#_ENREF_6)], a time on n-bit integers of exp((c + o(1))n^(1/3) log^(2/3) n) for some c < 2.

To be complete, Peter Shor [[8](#_ENREF_8)]has shown a quantum computer has a polynomial time algorithm for factoring integers. But engineering a quantum computer still has a way to go, so RSA is safe for now.

RSA security depends on the computation to be unreasonable for 1) the factoring of N, and 2) computing the *e*th roots modulo N. The first is fairly well researched and complexity is understood. However, the second is an open question. It has not yet been proven that taking the eth roots modulo N is at least as hard as factoring N. This means RSA security has not been proven to be at least as hard as factoring. But it has withstood time and a large amount of research, which is encouraging that it will stay secure well into the future.

1. **Elementary attacks.**

These attacks show blatant misuse of RSA. There are many such attacks exist, but in this report we will illustrate two of them:

* **Common Modulus**: Fixing N. One blatant misuse of RSA would be using the same modulus N for all users. So instead, have a central authority issue out unique e,d pairs to each user. This sort of works, in the fact that C1 = M^e1 mod N can only be decrypted with corresponding pair d1. So a different user with e2,d2 doesn’t appear able to decrypt. But from the property that knowing (e,d,N) you can factor N, then the game is over. Any user can use their ei,di pair to factor N. Then, given any other user’s ej and the factors of N, dj can be computed. RSA key pairs should never use the same N twice. [[6](#_ENREF_6)]
* Signature forgery can be done with a technique called **blinding**. If Alice ever “blindly” signs M, then Marvin might be able to forge Alice’s signature. If Marvin sends evil plaintext M for Alice to sign, it’s easy to assume Alice would reject and not sign it. But what if Marvin sends a random, harmless looking M? Depending on the implementation, Alice might be fooled and sign it. Marvin then can take advantageous of this is by generating M’ = r^e M mod N, for some random r. If Alice provides a signature, S’ on M’, then Marvin can compute Alice’s signature S for M by S = S’/r mod N. [[6](#_ENREF_6)]

1. **Low private exponent attacks (Using Chinese Remainder Theorem)**

Having a low private exponent d will reduce decryption and signature computing costs. However, too low of a d is insecure. Boneh [[6](#_ENREF_6)] describes approximations using fractions can allow Marvin to solve for a small d that is d < 1/3 N^(1/4). If N is 1024 bits, then d should be at least 256 bits long order to avoid this attack.

1. **Low public exponent attacks.**

Having a low public exponent will reduce encryption and signature validation computing costs. However, too low of an *e* is also insecure. Today’s standard *e* is set at 2^(16) + 1. This is a large enough value to avoid attacks and needs only 17 mod multiplications for M^*e* mod N using repeated squares. But if a very small *e* is used instead, it can be subject to attacks such as Hastad’s Broadcast Attack. If the same M is encrypted with many users (*e*,N) keys and broadcasted out, Marvin can collect each and compute M. If all users have the same *e*, then Marvin needs to collect at least *e* messages. Restating Boneh [[6](#_ENREF_6)], take example if *e*=3:

C1 = M^3 mod N1, C2 = M^3 mod N2, C3 = M^3 mod N3

M < {N1, N2, N3} thus M^3 < N1N2N3

Using Chinese Remainder Theorem (CRT) C1C2C3 = M^3 mod N1N2N3, thus taking cube root of C1C2C3 gives M.

Stronger attacks are also known on a small *e*. If you pad M in the above scenario to make it unique for each message, then the broadcast attack fails. But Hastad shows if the padding scheme is a public, fixed polynomial function it doesn’t defend from the attack. Franklin and Reiter found an attack on two related messages encrypted with same modulus in time quadratic to *e*. And Coppersmith took it farther to show an attack on same messages that used a short, random pad (1/9th the size of M). So using a small *e* is not wise.

To defend against all above low public exponent attacks, large *e* such as the standard 2^(16) + 1 should be used. A good randomized pad also helps make random M’s to remove relationship amongst messages.

1. **Attacks on the implementation.**

Research [[9](#_ENREF_9)] has shown that it is possible to recover the private keys of RSA without directly breaking RSA. This type of attack is known as "timing attack" an attacker notes runtime encryption algorithm and informed and thus justify the extracted parameter involved in secret operations. While it is generally agreed that the RSA is secure from direct attack, not so well known and often overlooked weakness of RSA to timing attacks.

In 1996, Kocher was the first one who discuss about timing attacks. Timing attacks are a form of "side channel attack" where an attacker gains information from the implementation of a coding rather than from any inherent weakness in the mathematical properties of the system. The operation in RSA that involves the private key is thus the modular exponentiation *M* = *Cd* mod *N,* where *N* is the RSA modulus, *C* is the text to decrypt or sign, and *d* is the private key. The attacker wants to find *d*. For a timing attack, the attacker must have a target system compute *Cd* mod *N* for several carefully selected values of *C*. By precisely measuring the amount of time required and analyzing the timing variations, the attacker can recover the private key *d* one bit at a time until know the entire exponent. [[9](#_ENREF_9)]

For example the “repeated squaring algorithm” does a round of computation for each bit of *d*. If the bit is 1, then an additional multiplication mod N is performed. Thus, analyzing the timing/power to determine the operations can give away *d*. To prevent, fix the algorithm to provide the same timing and power for each bit of *d* regardless of 0 or 1.[[6](#_ENREF_6)]

If using the Chinese Remainder Theorem (CRT) to speed up computation, a random fault could expose secrete key *d* on a signature. With CRT to compute M^*d* mod N, you instead work with M^*d* mod *q* and M^*d* mod *p* to reduce the computation time by working with smaller modulus. However, if a random fault happens in only one of the equations, then Marvin can figure out one of the factors of N. If a fault happened in M^*d* mod *q*, then for the final produced C, Marvin would observe C^*e* does not equal M mod N. Now C^*e* = M mod *q* is also not true, but C^*e* = M mod *p* is. Thus gcd(N, C^*e* – M) exposes a factor of N… To prevent, use random padding so Marvin doesn’t know M. Or, double check the signature to observe faults before releasing. [[6](#_ENREF_6)]

Timing attacks illustrate that attackers do not necessarily play by the presumed rules and that they tend to attack the weakest link in a system. The best way to protect against such attacks is to think like an attacker and find these links before attacker does.

* **Detailed El-Gamal cryptosystems**
  + - **History**
    - **Steps in El-Gamal Algorithm**
    - **El-Gamal Signature Scheme**
    - **Applications of El-Gamal Cryptosystem**
    - **ElGamal Cryptosystem Performance**
* Comparison ElGamal Cryptosystem with RSA
* Implementations
* Conclusion
* Member Contributions

Every member of the group searched for papers as sources for our project. Then we divided tasks between members. Maryam discussed the part of RSA Cryptosystem, while Abraham takes the part of El-Gamal Cryptosystem.

* References

[1] J. E. Sarlabous, "Introduction to cryptography," 2000.

[2] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, "Applied Cryptography," ed: CRC Press, New York, 1997.

[3] M. Stamp, *Information security: principles and practice*: John Wiley & Sons, 2011.

[4] E. Milanov, "The RSA Algorithm," 3 June 2009.

[5] D. Wright, "The RSA Algorithm," April 6th, 2007.

[6] D. Boneh, R. Rivest, A. Shamir, and L. Adleman, "Twenty years of attacks on the RSA cryptosystem," *Notices of the AMS,* vol. 46(2)*,* p. 203-213, 1999.

[7] C. Pomerance, "A tale of two sieves," *Biscuits of Number Theory,* vol. 85, 2008.

[8] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM journal on computing,* vol. 26(5)*,* p. 1484-1509, 1997.

[9] W. H. Wong, "Timing attacks on RSA: revealing your secrets through the fourth dimension," *Crossroads,* vol. 11(3)*,* p. 5-5, 2005.