RSA VS Diffie-Hellman Key Exchange

Is RSA better than Diffie-Hellman Key Exchange or is Diffie-Hellman Key Exchange better than RSA?

By Kshitij Galav & Alok Kumar (IIT Math Professor)

Table of Contents

Introduction

RSA and Diffie-Hellman Key Exchange are modern cryptosystems used to establish secure data transmission. These cryptosystems share similar objectives, however, differ from one another in their functioning. Both of these cryptosystems are based on different mathematical functions. In this paper these cryptosystems will be compared, contrasted, and evaluated on the basis of their mathematical functions' complexity, vulnerability, and performance. In essence, these parameters will be employed **to investigate which cryptosystem is better than the other**. This research holds significance as RSA and Diffie-Hellman perform similar tasks, hence making it confounding to choose one out of the two. Furthermore, with rising importance of privacy this research aids in culling the more effective cryptosystem. To answer the research question both cryptosystems are first discussed, that is their history and application. This is followed by an explanation of the mathematical functions which underlie the working of the cryptosystems. Then, both the cryptosystems are assessed on the criteria stated above. This assessment is carried out by analyzing the functions, which involves inspecting the workings of the functions [complexity of the function], ways to solve the functions [vulnerabilities], and the time taken by the functions to complete an operation [performance]. On the basis of this assessment, their scope, and application, these cryptosystems are then compared and contrasted. From this exercise, the better cryptosystem is identified.

Introduction to Cryptography

Cryptography is a technique used to ensure the safe communication or storage of data by converting this data into unreadable form which can be converted back only via a key owned by the concerned person. In the word cryptography, crypt stands for vault and graphy means writing. Cryptography first emerged thousands of years ago, with the Egyptians being the earliest known users of this technique. The Egyptians made use of various symbols to encrypt data. Earlier, this technique was restricted only among authoritative figures to safeguard their communication primarily during conflicts. However, with the passage of time cryptography has become an important aspect of everyone's lives. On the internet, today, people are constantly communicating with one another, undertaking online transactions, sharing data, etc. This makes cryptosystems crucial so as to ensure that this data cannot be stolen by individuals and organizations with malicious intent.

Mathematics in Cryptography

As seen in the previous section, cryptography involves changing data into an unreadable form [encryption]. Once this unreadable form [cipher text] reaches the concerned individual, it is changed back to the original data [decryption]. This process of encryption and decryption involves mathematics in the form of functions. In a simple function, a number is input in the function [domain], it is then acted upon by the rule of the function, and then we receive an output [range]. Similarly, in a cryptosystem like RSA data is first input in the RSA function, it is then acted upon by the function's rule, and then the output received is the encrypted data. Decryption of data involves inputting the range which we received from the initial function into the inverse of the initial function. So, the domain [which is the range of the initial function] acts as our input into the inverse of the initial function, it is then acted upon by the function's rule, and then the output [range] which we receive is the initial domain which we input into the original function. In the case of a cryptosystem like RSA encrypted data is first input into the inverse of the RSA function, it is then acted upon by the function's rule, and then the output which we get is the decrypted data.

RSA

RSA [Rivest–Shamir–Adleman] was publicly announced in 1978 by its founders Ron Rivest, Adi Shamir, and Leonard Adleman, hence RSA. RSA was one of the first asymmetric cryptosystems [a system which involves two keys: public key which is shared across the Internet and is responsible for encryption and private key which is to be kept private by the owner and is responsible for decryption] during a time when symmetric cryptosystems [a system having the same key for both encryption and decryption] were prominent. Due to the increase in vulnerabilities of a symmetric system [the key could not be shared across the internet, and the interception of the key would break the cryptosystem] RSA became popular. Despite being around for a long time, a viable method to break RSA has not been discovered and RSA still has industrial application.

Working of RSA

As seen in the previous section, RSA makes use of two separate keys. Functioning of this asymmetric system is shown:

1. User *1* chooses large prime numbers:

   Prime Numbers: p and q

2. *p* and *q* are then multiplied with one another to obtain a number *n*, termed as the modulus:

   $p \times q = n$ *[gives length of key]*

3. User *1* finds a number *e*, termed as the encryption key (exponent):

   $e \longrightarrow 1 < e < (p-1) \times (q-1)$ $[(p-1) \times (q-1) = \phi(n)]$ e should be relatively prime [integers a and b are relatively prime if 1 is the only positive integer they are divisible by] to $\phi(n)$.

4. User *1* then finds a number *d*, termed as the decryption key. *d* is found by finding the modular inverse of *e* with respect to $\phi(n)$:

   $e \times d \bmod \phi(n) = 1$

   To solve this equation Extended Euclidean Algorithm is used [method to compute the greatest common divisor of two integers, demonstrated as an example below].

5. From the variables above, user *1* determines the public and private key. Public Key:

   (e,n)

   Private Key:

   (d,n)

6. User *1* shares the Public Key with user *2*, who encrypts his message using the formula:

   $plaintext^{e} \pmod{n} = encryptedtext$

mod stands for the function modulo of the form *a mod(b)*, used to calculate the remainder when *a* is divided by *b*.

7. User *2* sends the ciphertext (*encryptedtext*) to user *1*, who decrypts it using the formula:

   $encryptedtext^{d} \ (mod(n)) = plaintext$

An example of how RSA works [small prime numbers have been used to simplify the calculations]:

1. User *1* chooses:

   $p = 11$ and $q = 5$

2. To obtain the modulus [*n*], user *1* multiplies *p* and *q*:

   $p \times q = n$

   $11 \times 5 = 55 = n$

3. To obtain $\phi(n)$, user *1* multiplies *(p-1) and (q-1)*:

   $(p-1) \times (q-1) = \phi(n)$

   $(11-1) \times (5-1) = 40 = \phi(n)$

4. User *1* obtains value of encryption key [*e*]:

   $1 < e < \phi(n), \ 1 < e < 40$

   Value should also be relatively prime to $\phi(n)$. From this:

   $e = 7$

5. User *1* obtains value of decryption key [*d*]:

   $e \times d \ mod \ \phi(n) = 1$ *-- [1]*

This is solved via the Extended Euclidean Algorithm, involving two steps: Euclidean Algorithm [method to calculate the greatest common divisor] and Back Substitution

[method where we plug in the results of the first step]. We plug the values of *e* and *n* in *[1]*:

$7 \times d \bmod 40 = 1$

Applying the Euclidean Algorithm -

a. First, *[1]* is written in the form of the Euclidean Equation [equation of the form $AX + BY$, where A and B are integers whose greatest common divisor we wish to compute):

$40X + 7Y = 1$

b. Now 40 is written in the LHS, 7 in the RHS, and manipulate the RHS to be equal to the LHS:

$40 = 7\ (5) + 5$

Same procedure is followed for 7 and 5:

$7 = 5(1) + 2$

This procedure is repeated till 1 is received in the RHS:

$5 = 2(2) + 1$

c. After the Euclidean Algorithm, the Back Substitution method is used:

$5 = 2(2) + 1$ *[received in step b]* this

is manipulated to get:

$1 = 5 - 2(2)$ -- *[2]*

Similarly, from step b:

$7 = 1(5) + 2$ is manipulated to get:

$2 = 7 - 1(5)$ -- *[3]*

Plugging *[3]* into *[2]*:

$1 = 5 - 2(7 - 1(5)),\ 1 = 3(5) - 2(7)$ -- *[4]*

Furthermore:

40 = 5(7) + 5 [from step b] is

manipulated to get:

5 = 40 - 5(7) -- [5]

Plugging *[5]* in *[4]*:

1 = 3(40 - 5(7)) - 2(7), 1 = 3(40) - 17(7) -- [6]

d. Extended Euclidean Algorithm has a rule that there should be equal number of steps in the Euclidean Algorithm and Back Substitution step. Hence, user *1* stops at *[6]*:

d = 40 - 17 = 23 [comparing with equation used to solve d]

6. From the above, user *1* determines the Public and Private Keys. Public Key:

(7,55)

Private Key:

(23,55)

7. User *1* shares the public key with user *2* who uses it to encrypt, for example, nine.

User *2* plugs it in the formula for encryption:

$[plaintext^e \quad mod(n)] \rightarrow 9^7 \ mod(55) = 4$ giving the ciphertext (*encryptedtext*). User *2* delivers the ciphertext to user *1* who decrypts it by plugging it in the formula for decryption:

$[encryptedtext^d \ mod(n)] \rightarrow 4^{23} \ mod(55) = 9,$ giving user 1 the encrypted message.

Vulnerabilities of RSA

The security of this cryptosystem relies on the difficulty of factoring large integers. During industrial implementation, RSA does not involve the usage of small prime numbers. Instead, large prime numbers like 340282363487254643170864374573732807431 are used [this is from a 128-bit encryption which is easily broken!]. With time the computational power of computers has increased, and computers have been able to brute force their way into guessing the prime numbers from the modulus. This led to an increase in the key size, with key sizes ranging from 1024 bits - 4096 bits. Forms of RSA encryption with such key lengths have not been broken yet, however, quantum computers [still under development] may be able to break these keys. There are mathematical approaches to break RSA encryption, however, they would take an infinite amount of time even on a supercomputer when trying to break RSA encryption with a large key length. One such mathematical attack is based on the AKS
Algorithm which states that:

any given number $n \geq 2$ *is prime if* $\rightarrow ((x + a)n\ ! \equiv (xn + a)\ (mod\ n))$ where a is relatively prime to n and x is to be seen as a formal symbol. AKS Algorithm is used to find out a specific position k in the Pascal Triangle for which the segment wherein the sum of k binomial coefficients with n is seen to be greater than one as:

$${}^{n}C_0\ (mod\ n) = 1$$

From this, the Greatest Common Divisor (gcd) of k and n will give us the value of p [one of the prime numbers]. The algorithm is applied as:

1. We first find the value of k which satisfies:

$$\sum_{i=0}^{k} {}^{n}C_i\ (mod\ n) > 1$$

2. Then:

$$p = gcd\ (n,k),$$

$$q = n \div p$$

3. GCD of n and k is calculated using the Euclidean Algorithm [as demonstrated in the previous section].

This method works effectively for small primes, but with large primes an infinite amount of time is required by computers.

Diffie-Hellman Key Exchange

Diffie-Hellman Key Exchange first released to the public in 1976 by Whitfield Diffie and Martin Hellman, hence Diffie-Hellman. Diffie-Hellman was the first asymmetric cryptosystem available for public use and is used to this day. Today, Diffie-Hellman is used in the security protocol SSL: the basis of data transfer across the internet. Unlike RSA, Diffie-Hellman Key Exchange cannot transfer messages. Instead, Diffie-Hellman allows for the exchange of cryptographic keys across a public communication channel. Exchange of cryptographic keys does not mean the exchange of these keys, instead public variables are transferred from one user to another and are then combined with private variables, which are kept secure by the user, via the Diffie-Hellman Algorithm. In essence, both users together derive the cryptographic key. The derived cryptographic key may then be used in a cryptographic system such as AES. For example, if you visit YouTube, before the website loads on your device a Diffie-Hellman Key Exchange is performed. After this data is securely transferred [secured via the key derived by our device and YouTube] between our device and YouTube.

Working of Diffie-Hellman Key Exchange

As seen in the previous section, Diffie-Hellman involves both users deriving a cryptographic key. Its functioning is shown:

1. If two users wish to communicate with one another, they agree on a large prime number and a base [generator]:

   $0 < generator < prime\ number$

   $0 < g < p$

2. One user selects an integer a [this is a secret and is the user's private key]. Similarly, the second user chooses his private key b.

3. Now users calculate their public keys. The first user calculates his public key:

   $A \rightarrow g^a \bmod p = A$

   Similarly, the second user calculates his public key:

   $B \rightarrow g^b \bmod p = B$

4. Both users send their public keys to one another. The users have each other's public key, however, fail to manipulate the formula for the public key:

   $g^a \bmod p$ *or* $g^b \bmod p$ to calculate the value of a or b. This is because of the discrete logarithm problem:

   $e^c \bmod l = x$

   Where, it is easy to find the value of x but is difficult to find the value of c.

   Essentially, we have a one-way function: it is easy to solve one way, but it is difficult to solve the other way around.

5. Therefore, the users use the public keys in the equation:

   $B^a \bmod p$

$A^b \bmod p$

what is interesting is that both of these equations give the same result. This is because of the property of modulo exponents:

$(g^a \bmod p)^b \bmod p = g^{ab} \bmod p$,

$(g^b \bmod p)^a \bmod p = g^{ba} \bmod p$

Regardless of order, both users reach the same result without each other's private key.

6. Result calculated by the users is the shared secret key which can be used in other cryptosystems.

An example of how Diffie-Hellman works [small prime numbers have been used to simplify the calculations] :

1. Two users, User 1 and User2, agree on:

   prime number 23 [p] , generator

   9 [g]

2. Both users then choose their private keys. User 1's private key:

   4 [a]

   User 2's private key:

   3 [b]

3. Users then calculate their public keys. User 1's public key:

   $9^4 \bmod 23 = 6$ [$g^a \bmod p = A$]

   User 2's public key:

   $9^3 \bmod 23 = 16$ [$g^b \bmod p = B$] These

   keys are then exchanged with one another.

4. Users then calculate the shared secret key. For User 1:

$$16^4 \bmod 23 = 9 \; [B^a \bmod p]$$

For User 2:

$$6^3 \bmod 23 = 9 \; [A^b \bmod p]$$

Therefore, *9* is the shared secret key.

Vulnerabilities of Diffie-Hellman Key Exchange

Strength of this cryptosystem relies on the discrete logarithm problem. The difficulty of solving for *c*:

$e^c \bmod l = x$ makes Diffie-Hellman strong. However, the Pohlig-Hellman Algorithm [method to solve the discrete logarithm problem] is a way to exploit this cryptosystem. This algorithm allows us to solve for *c:*

$e^c \bmod l = x$ revealing the private key and breaking this cryptosystem.

The algorithm is shown:

1. We have an equation:

   $\beta = a^x \bmod p$ where $\beta = 12$, $a = 7$, $0 \leq x < p-1$, and p (a prime):

   $12 = 7^x \bmod 41$ -- *[1]*

   Here we are solving for x.

2. We find the prime factors of $p - 1$:

   $41 - 1 = 40$

   The prime factors of *40*:

   $2^3 \times 5$

   2 *and* 5 Let prime factors be q.

3. We find one x for each q.

4. For $q = 2$:

   $x = 2^0 x_0 + 2^1 x_1 + 2^2 x_2$ -- *[2] [three terms as 2^3]*

   We solve for x_0 by solving:

$\beta_{p-1 \div q} = a_{(p-1 \div q)} \times x0,$

$12_{40 \div 2} = 7_{(40 \div 2)} \times x0$ -- *[3]*

We get this equation by raising both sides of *[1]* to the power:

$p-1 \div q$

With modulo,*[3]* can be rewritten:

$-1 \pmod{41} = -1^{x0} \pmod{41}$ -- *[4]*

5. To solve *[4]*, we plug values 0, 1, 2... [can plug other values too] for x_0 and check the lowest value for which LHS = RHS. Therefore, $x_0 = 1$.

6. We now solve for x_1. To solve for x_1, we first compute $\beta 1$ and $q1$:

$\beta 1 = \beta \times a^{-(x0)}$, $\beta 1 = 12 \times 7^{-(1)}$ with

modulo this is rewritten:

$31 \pmod{41}$

For q1:

$2^2 = 4 = q1$ *[moving from* 2^1 *in q]*

7. Solving for x_1:

$\beta 1_{p-1 \div q1} = a_{(p-1 \div q)} \times x1$

$31_{40 \div 4} = 7_{(40 \div 2)} \times x1$

with modulo, LHS of this equation is rewritten:

$1 \pmod{41}$

LHS is equal to *1*. Only way RHS of this equation is equal to *1* is when $x_1 = 0$, therefore $x_1 = 0$.

8. We now solve for x_2. To solve for x_2, we first compute $\beta 2$ and $q2$:

$\beta 2 = \beta 1 \times a^{-(x1)}$, $\beta 2 = 31 \times 7^{-(0)}$

with modulo this is rewritten:

$31 \pmod{41}$

For q2:

$2^3 = 8 = q2$ *[moving from* 2^2 *in q1]*

9. Solving for x_2:

$$\beta 2_{p-1 \div q2} = a_{(p-1 \div q) \times x2}, 31_{40 \div 8} = 7_{(40 \div 2) \times x2}$$

with modulo this is rewritten:

$-1 \pmod{41} = -1^{x2} \pmod{41}$

This equation is true when $x_2 = 1$, therefore $x_2 = 1$.

10. Using values of x_0, x_1, and x_2 we can solve *[2]*:

$x = 2^0 (1) + 2^1 (0) + 2^2 (1), 1 + 0 + 4 = 5 = x$

From $q = 2$, we get:

$5 \pmod{2^3}$ *[factors of* $40 = 2^3 \times 5$*]*

11. Now we find x for the other q.

12. For $q = 5$:

$x = 5^0 x_0$ *-- [5] [one term as* 5^1*]*

We solve for x_0 by solving:

$$\beta_{p-1 \div q} = a_{(p-1 \div q) \times x0}, 12_{40 \div 5} = 7_{(40 \div 5) \times x0} \text{ -- [6]}$$

We get this equation by raising both sides of *[1]* to the power:

$p-1 \div q$

With modulo, *[6]* can be rewritten:

$18 \equiv 37^{x0} \pmod{41}$ *-- [7]* equivalent sign (≡) denotes that $18 \div 41$ and $37^{x0} \div 41$ give the same remainder

13. Solving *[7]* it is clear that $x_0 \neq 0$ and $x_0 \neq 1$ as these values make the equation false. Therefore, we plug values 2, 3, 4… for x_0 and accept the lowest value for which the equation is true. *[7]* is true when $x_0 = 3$, therefore, $x_0 = 3$.

14. Using value of x_0 we solve *[7]*:

    $x = 5^0 (3),\ 1 (3) = x = 3$

    From $q = 5$ we get:

    $3 \pmod 5$ *[factors of* $40 = 2^3 \times 5$*]*

15. We have a value of x for each q. For $q = 2$:

    $x \equiv 5 \pmod{2^3}$

    For $q = 5$:

    $x \equiv 3 \pmod 5$

    Using these x values, we solve for x using the Chinese Remainder Theorem (a theorem to solve simultaneous linear congruences equations with coprime moduli):

    a. Chinese Remainder Theorem [CRT] can only be used when moduli have a GCD of *1* [they are coprime]. In:

       $x \equiv 5 \pmod 8$ *-- [8]* , $x \equiv 3 \pmod 5$ *-- [9]*

       8 and 5 are coprime. Hence, CRT can be applied.

    b. We now write *[9]* as:

       $x = 5j + 3$ *-- [10]* plugging

       [10] in *[8]* gives us:

       $5j + 3 = 5 \pmod 8$ Solving

       for j we get:

       $j = 2 \pmod 8$ *-- [11]*

We now write *[11]* as an equation and plug it in *[10]*:

$j = 8k + 2$,

$x = 5\ (8k + 2) + 3$ -- *[12]*

Solving *[12]* we get:

$x = 40k + 13$

giving us:

$x = 13 \pmod{40}$ -- *[13]* and is our solution for

the system of equations. We get:

mod 40 as x is an exponent and

exponents are always:

mod p - 1

16. From *[13]*, *[1]* now becomes:

$12 = 7^{13} \bmod 41$

The discrete logarithm problem has been solved and the private key:

13 has been found.

Although this algorithm provides a way to solve the discrete logarithm problem, this is only viable when:

$p - 1$ has small factors. As the factors become larger, solving using this method becomes tedious and defeats the purpose of using the algorithm in the first place.

Comparison of RSA and Diffie-Hellman Key Exchange On The Stated Criteria

As seen in the previous sections, RSA and Diffie-Hellman differ in their objective as Diffie-Hellman cannot be used to send messages like RSA. However, key exchange [task of Diffie-Hellman] can be performed by RSA [through a method different than Diffie-Hellman]. In this section, we will be comparing the two cryptosystems on the basis of the criteria stated in the introduction: Mathematical Function Complexity, Vulnerability, and Performance.

From close examination of both cryptosystems in the previous sections, we can conclude that Diffie-Hellman has the more complex algorithm. This boils down to two attributes of the functioning of Diffie-Hellman. First, Diffie-Hellman involves the usage of more variables in its algorithm compared to RSA. In RSA, we randomly choose two prime numbers and calculate the other elements on the basis of these two primes. On the other hand, in Diffie-Hellman, we choose a prime, a generator, and the private keys for both the users. These variables are then used to calculate the other elements. Second, is the simplicity in the functioning of RSA. In RSA, once communication between two users has been established usually the variables remain unchanged and the function is repeated with the same variables. On the other hand, in Diffie-Hellman, variables are changed every time communication is initiated between two users and the function is repeated with different variables.

During the discussion of both cryptosystems, we delved into their vulnerabilities and briefly discussed various mathematical methods to break them. Mathematically, the weakness of both these cryptosystems lie in mathematical methods which attempt to solve the functions.

However, in the case of both cryptosystems such mathematical approaches are only valid when the numbers are small. When the numbers get larger, calculations become larger and it becomes impractical to solve for the algorithms even while approaching it through a computer. Hence, as far as the weakness of both the cryptosystems are concerned both are secure from mathematical attacks if certain standards in the choice of primes and key length are maintained [choosing large numbers and keys].

While looking at the performance of both cryptosystems, we look at the speed of algorithms and time they take to complete an operation. Out of the two cryptosystems, RSA is the faster one. This stems from the fact that RSA, compared to Diffie-Hellman, is less complex. RSA requires less processing power compared to Diffie-Hellman for its functioning. Easier calculations and less variable exchanges are two factors which reduce the processing power required by RSA and even make it faster than Diffie-Hellman.

Another difference between the two is that due to larger support for RSA, its standard is freely available for use by an individual. On the other hand, for Diffie-Hellman, its standard is not freely available to an individual and this makes RSA the more widely used cryptosystem.

Conclusion

From the exercise in the previous section we observed that according to our criteria, neither cryptosystem has a clear advantage over the other. This leads us to conclude that depending upon the situation RSA is better than Diffie-Hellman or Diffie-Hellman is better than RSA. If we are looking for a cryptosystem with high performance, easily available standard, and ability to work with data and cryptographic keys RSA is the way to go. However, if we are looking for a cryptosystem specifically for key exchange or a mode to establish secure communication between two users Diffie-Hellman should be your choice.

Bibliography

- Dr. Stinson, *The Pohlig-Hellman Algorithm*. Available at: http://anh.cs.luc.edu/331/notes/PohligHellmanp_k2p.pdf
- Agrawal, Manindra et al. "Primes Is In P". IIT Kanpur. Available at: https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf
- Lambers, Jim. "MAT 610". Available at: https://www.math.usm.edu/lambers/mat610/sum10/lecture4.pdf
- "The Chinese Remainder Theorem.". Available at: http://gauss.math.luc.edu/greicius/Math201/Fall2012/Lectures/ChineseRemainderThm.article.pdf
- "Diffie-Hellman Key Exchange.". Available at: http://www.cse.unt.edu/~tarau/teaching/PP/NumberTheoretical/Diffie%E2%80%93Hellman%20key%20exchange.pdf
- Milanov, Evgeny. "The RSA Algorithm.". Available at: https://sites.math.washington.edu/~morrow/336_09/papers/Yevgeny.pdf