Research Paper Mathematics

Use of Various Algorithms For Cryptosystem

* C. Sajeev ** C. Suyambulingom

* Research Scholar , Sathyabama University

** Professor (Rtd.), Tamilnadu Agricultural University , Coimbatore

ABSTRACT

The idea of information security lead to the evolution of Cryptography. In other words, Cryptography is the science of keeping information secure. It involves encryption and decryption of messages. Encryption is the process of converting a plain text into cipher text and decryption is the process of getting back the original message from the encrypted text. Cryptography, in addition to providing confidentiality, also provides Authentication, Integrity and Non-repudiation. The crux of cryptography lies in the key involved and the secrecy of the keys used to encrypt or decrypt. Another important factor is the key strength, i.e. the size of the key so that it is difficult to perform a brute force on the plain and cipher text and retrieve the key. There have been various cryptographic algorithms suggested

Keywords : Elliptic curve cryptography, Public Key Systems

Diffie-Hellman (DH) public-key algorithm

Diffie-Hellman is not an encryption mechanism as we normally think of them in that we do not typically use it to encrypt data. Instead, it is a protocol to securely exchange the keys that encrypt data. Diffie-Hellman accomplishes this secure exchange by creating a “shared secret” (sometimes called a “Key Encryption Key” or KEK) between two devices. The shared secret then encrypts the symmetric key for secure transmittal. The symmetric key is sometimes called a Traffic Encryption Key (TEK) or Data Encryption Key (DEK). Therefore, the KEK provides for secure delivery of the TEK, while the TEK provides for secure delivery of the data itself.

The protocol has two system parameters *p* and *g*. They are both public and may be used by all the users in a system. Parameter *p* is a prime number and parameter *g* (usually called a generator) is an integer less than *p*, with the following property: for every number *n* between 1 and *p*-1 inclusive, there is a power *k* of *g* such that $n = g^k \bmod p$. To make a more simple description we shall imagine two people – Alice and Bob who want to securely exchange data.

Suppose Alice and Bob want to agree on a shared secret key using the Diffie-Hellman key agreement protocol. They proceed as follows: Alice and Bob agree on a finite cyclic group *G* and a generating element *g* in *G*. (This is usually done long before the rest of the protocol; *g* is assumed to be known by all attackers). First, Alice generates a random private value *a* and Bob generates a random private value *b*. Both *a* and *b* are drawn from the set of integers . Then they derive their public values using parameters *p* and *g* and their private values. Alice’s public value is $g^a \bmod p$ and Bob’s public value is $g^b \bmod p$. They then exchange their public values. Finally, Alice computes $g^{ab} = (g^b)^a \bmod p$, and Bob computes $g^{ba} = (g^a)^b \bmod p$. Since $g^{ab} = g^{ba} = k$, Alice and Bob now have a shared secret key *k*. The important point is that the two values generated are identical. They are the “Shared Secret” that can encrypt information between systems .

Here is an example of the protocol, with non-secret and secret values*:*

Alice and Bob agree to use a prime number p=23 and base g=5.

Alice chooses a secret integer a=6, then sends Bob $A = g^a \bmod p$

$A = 5^6 \bmod 23 = 8.$

Bob chooses a secret integer b=15, then sends Alice $B = g^b \bmod p$

$B = 5^{15} \bmod 23 = 19.$

Alice computes $s = B^a \bmod p$

$19^6 \bmod 23 = 2.$

Bob computes $s = A^b \bmod p$

$8^{15} \bmod 23 = 2.$ [8]

At this point, the Diffie-Hellman operation could be considered complete. The shared secret is a cryptographic key that could encrypt traffic. That is very rare however because the shared secret is an asymmetric key. As with all asymmetric key systems, it is inherently slow. If the two sides are passing very little traffic, the shared secret may encrypt actual data. Any attempt at bulk traffic encryption requires a symmetric key system such as DES, Triple DES, or Advanced Encryption Standard (AES), etc. In most real applications of the Diffie-Hellman protocol (SSL, TLS, SSH, and IPSec in particular), the shared secret encrypts a symmetric key for one of the symmetric algorithms, transmits it securely, and the distant end decrypts it with the shared secret. Because the symmetric key is a relatively short value (256 bits for example) as compared to bulk data, the shared secret can encrypt and decrypt it very quickly.

Which side of the communication actually generates and transmits the symmetric key varies. However, it is most common for the initiator of the communication to be the one that transmits the key.

Once secure exchange of the symmetric key is complete, data encryption and secure communication can occur. Changing the symmetric key for increased security is simple

at this point. The longer a symmetric key is in use, the easier it is to perform a successful cryptanalytic attack against it. Therefore, changing keys frequently is important. Both sides of the communication still have the shared secret and it can be used to encrypt future keys at any time and any frequency desired. In some IPSec implementations for example, it is not uncommon for a new symmetric Data Encryption Key to be generated and shared every 60 seconds. The protocol depends on the discrete logarithm problem for its security. It assumes that it is computationally infeasible to calculate the shared secret key $k = g^{ab}$ mod p given the two public values g^a mod p and g^b mod p when the prime p is sufficiently large. It is stated that breaking the Diffie-Hellman protocol is equivalent to computing discrete logarithms under certain assumptions.

II. RSA Algorithm

RSA algorithm has to be described with an example.

Generating Public and Private Keys

Before any transmission happens, the Server had calculated its public and secret keys. Here is the way.

1) pick two prime numbers, we'll pick p = 3 and q = 11
2) calculate n = p * q = 3 * 11 = 33
3) calculate z = (p - 1) * (q - 1) = (3 - 1) * (11 - 1) = 20
4) choose a prime number k, such that k is co-prime to z, i.e, z is not divisible by k. We have several choices for k: 7, 11, 13, 17, 19 (we cannot use 5, because 20 is divisible by
5). Let's pick k=7 (smaller k, "less math").
5) So, the numbers n = 33 and k = 7 become the Server's public key.
6) Now, still done in advance of any transmission, the Server has to calculate it's secret key. Here is how.
7) k * j = 1 (mod z)
8) 7 * j = 1 (mod 20)
9) (7 * j) / 20 = ? with the remainder of 1.

Since we selected (on purpose) to work with small numbers, we can easily conclude that 21 / 20 gives "something" with the remainder of 1. So, 7 * j = 21, and j = 3. This is our secret key. We MUST NOT give this key away.

Now, after the Server has done the above preparatory calculations in advance, we can begin our message transmission from our Browser to the Server. First, the Browser requests from the Server, the Server's public key, which the Server obliges, i.e., it sends n=33 and k=7 back to the Browser. Now, we said that the Browser has a Plain message P=14, and it wants to encrypt it, before sending it to the Server. Here is how the encryption happens on the Browser.

Encrypting the message

Here is the encryption math that Browser executes.

1) P ^ k = E (mod n) "^" means "to the power of" P is the Plain message we want to encrypt n and k are Server's public key (see Section 1) E is our Encrypted message we want to generate After plugging in the values, this equation is solved as follows:
2) 14 ^ 7 = E (mod 33) This equation in English says: raise 14 to the power of 7, divide this by 33, giving the remainder of E.
3) 105413504 / 33 = 3194348.606 (well, I lied when I said that this is "Pencil and Paper" method only. You might want to use a calculator here).
4) 3194348 * 33 = 10541348
5) E = 105413504 - 10541348 = 20

So, our Encrypted message is E=20. This is now the value that the Browser is going to send to the Server. When the Server receives this message, it then proceeds to Decrypt it, as follows.

Decrypting the Message

Here is the decryption math the Server executes to recover the original Plain text message which the Browser started with.

1) E ^ j = P (mod n) E is the Encrypted message just received j is the Server's secret key P is the Plain message we are trying to recover n is Server's public key (well part of; remember that Server's public key was calculated in Section 1 as consisting of two numbers: n=33 and k=7). After plugging in the values:
2) 20 ^ 3 = P (mod 33)
3) 8000 / 33 = ? with the remainder of P. So to calculate this remainder, we do:
4) 8000 / 33 = 242.424242..
5) 242 * 33 = 7986

6) P = 8000 - 7986 = 14, which is exactly the Plain text message that the Browser started with.

III. DES Algorithm

The DES algorithm uses the following steps:

Step 1: Create 16 subkeys, each of which is 48-bits long. The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K+**. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Step 2: Encode each 64-bit block of data.

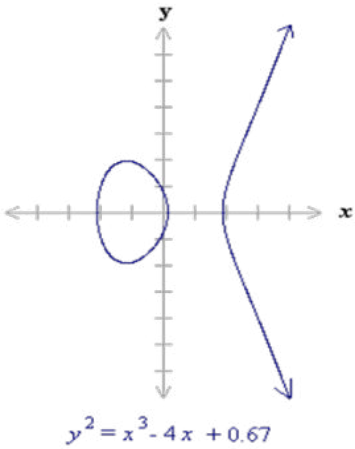
There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

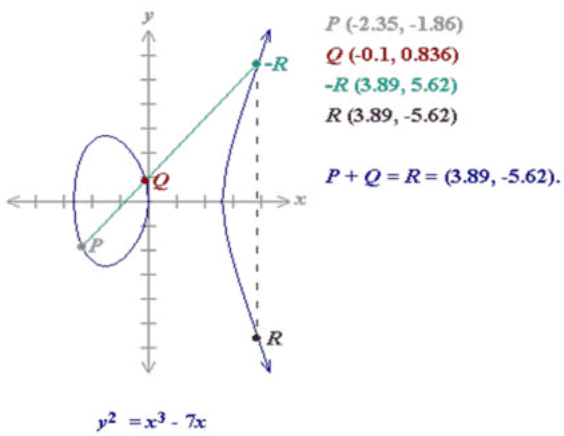
Decryption is simply the inverse of encryption, following the same steps as above, but reversing the order in which the sub keys are applied.

IV ECC Algorithm

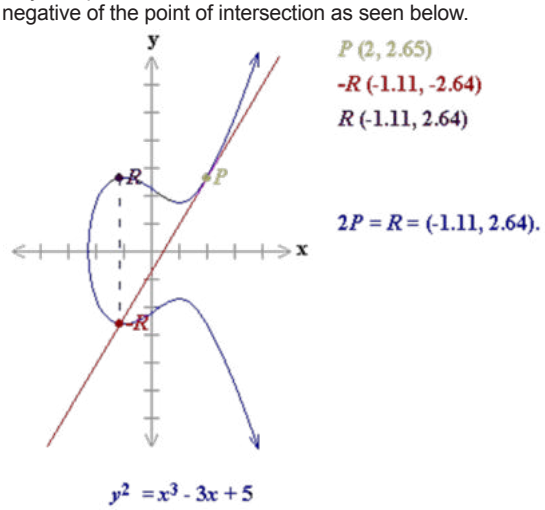


An *Elliptic Curve* is a set of point on a curve $y^2 = x^3 + ax + b$ given certain real numbers a and b. For example

Elliptic Curve Groups: The set of points on an elliptic curve, plus a special point ∞ form and additive group. The addition of two points on an elliptic curve is defined geometrically, as shown in the following example.



Elliptic Curve Encryption Algorithms depend on the difficulty of calculating kP where k is a product of two large primes and P is an element in the Elliptic Curve Group. Geometrically to add a point P to it self you first construct the tangent line to the curve at the point. Then the line will intersect the curve at only one point, and the addition of $2P$ is then defined to be the negative of the point of intersection as seen below.

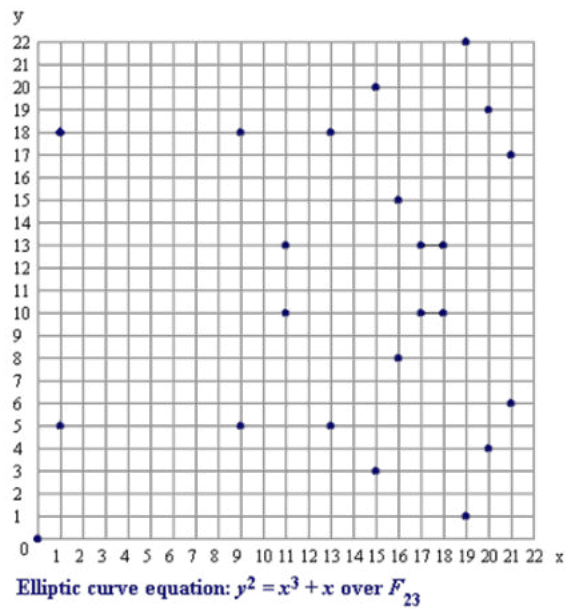


Elliptic curve groups over real numbers are not practical for cryptography due to slowness of calculations and round-off error. This Elliptic Curves over Finite Fields are used. An elliptic curve over a finite field F_p of characteristic greater than three can be formed by choosing the variables a and b within the field F_p.

the elliptic curve is then the set of points (x, y) which satisfy the elliptic curve equation $y^2 = x^3 + ax + b$ modulo p, where $x, y \in F_p$; together with a special point ∞. If $x^3 + ax + b$ contains no repeated factors, or equivalently if $4a^3 + 27b^2 \equiv 0 (\text{mod } p)$, then these points form a group.

It is well known that ECG (the Elliptic Curve Group) is an additive abelian group with ∞ serving as its identity element.

Example: In the ECG of $y^2 = x^3 + x^2$ over the field F_{23} the point $(9, 5)$ satisfies the equation $y^2 \equiv x^3 + x^2 (\text{mod } 23)$ as $25 \equiv 729 + 9 (\text{mod } 23)$. The elements of this ECG are given in the picture below.



$y_3 = \lambda(x_1 - x_3) - y_1$ $x_3 = \lambda^2 - x_1 - x_2$ Obviously we no longer have a curve to define our addition geometrically. Emulating the geometric construction for addition, the formulas for addition over F_p (characteristic 3) are given as follows: Let $P(x_1, y_1)$ and $Q(x_2, y_2)$ be elements of the ECG. Then $P + Q = (x_3, y_3)$, where

$$x_3 = \lambda^2 - x_1 - x_2$$

$$y_3 = \lambda(x_1 - x_3) - y_1$$

and

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & if \quad P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & if \quad P = Q \end{cases}$$

These formulas can be easily calculated with computers. For field of characteristic 2 the equations for addition are worse! At the heart of every cryptosystem is a hard mathematical problem that is computationally infeasible to solve. The Discrete Logarithm Problem is the basis for the security of many cryptosystem including the Elliptic Curve Cryptosystem.

Definition of the Discrete Logarithm Problem:
In the multiplication group $F_p^{\times}$, the discrete logarithm problem

that is: Given elements r and q in $F_p^{\times}$, find a number k such that $r = qk(\mathrm{mod}\ p)$.

Similarly the Elliptic Curve Discrete Logarithm Problem is: Given points P and Q in an ECG over a finite field find an integer k such that $Pk = Q$. Here k is called the discrete log of Q to the base P.

This doesn't seem like a difficult problem, but if you don't know what k is calculating $Pk = Q$ takes roughly $2^{k/2}$ operations. So if k is say, 160 bits long, then it would take about 2^{80} operations. To put this into perspective, if you could do a billion operations per second, this would take about 38 million years. This is a huge savings over the standard public key encryption system where 1024 and 3074 bit keys are recommended. The smaller size of the keys for Elliptic Curve Encryption makes it idea for applications such as encrypting cell-phone calls, credit card transactions, and other applications where memory and speed are an issue. There are pros and cons to both ECC and RSA encryption. ECC is faster tha RSA for signing and decryption, but slower than RSA for signature verification and encryption.

V. Conclusion

In this paper we perused the concept of Cryptography including the various schemes of system based on the kind of key and a few algorithms such as RSA and DES. We knew in detail the mathematical foundations for elliptical curve based systems, basically the concepts of rings, fields, groups, Galois finite fields and elliptic curves and their properties. The various algorithms for the computation of the scalar product of a point on the elliptic curve were known and their complexity were analyzed.

The advantage of elliptic curve over the other public key systems such as RSA, DES etc is the key strength. The following table summarizes the key strength of ECC based systems in comparison to other public key schemes.

RSA/DES Key length	ECC Key Length for Equivalent Security
1024	160
2048	224
3072	256
7680	384
15360	512

Comparison of the key strengths of RSA/DES and ECC

From the table it is very clear that elliptic curves offer a comparable amount of security offered by the other popular public key for a much smaller key strength. This property of ECC has made the scheme quite popular of late.

Over the years, there have been software implementations of ECDSA over finite fields such as $F_{2^{155}}$, $F_{2^{167}}$, $F_{2^{176}}$, $F_{2^{191}}$ and F_p (p: 160 and 192 bit prime numbers). Schroppel et mentions an implementation of an elliptic curve analogue of the Diffie-Hellman key exchange algorithm over $F_{2^{155}}$ with a trinomial basis representation. The elliptic curve based public key cryptography schemes has been standardized by the Institute of Electrical and Electronic Engineers (IEEE) and the standard is available as IEEE P1363.

REFERENCES

[1] Diffie, W., and Hellman, M. "Multiuser Cryptographic Techniques." IEEE Transactions on Information Theory, November 1976. | [2] A. J. Menezes, P. C. van Oorshot, and S. A Vanstone, Handbook of Applied Cryptography. CRC Press, New York, New York, 1997. | [3] P. C. van Oorschot and M. J. Wiener, On Diffie-Hellman Key Agreement with Short Exponents. EUROCRYPT'96, LNCS 1070, Springer-Verlag, 1996, pp. 332–343. | [4] Popek, G., and Kline, C. "Encryption and Secure Computer Networks." ACM Computing Surveys, December 1979. | [5] Kohnfelder, L. Towards a Practical Public-Key Cryptosystem. Bachelor's Thesis, M.I.T., May 1978. | [6] Denning, D. "Protecting Public Keys and Signature Keys." Computer, February 1983. |