

Understanding Asymmetric Encryption Algorithms

Colin MacCreery
Western Michigan University
College of Engineering
Computer Science Department
Kalamazoo, Michigan
Email: colin.c.macreery@wmich.edu

Jason Pearson
Western Michigan University
College of Engineering
Computer Science Department
Kalamazoo, Michigan
Email: jason.j.pearson@wmich.edu

Abstract—This paper contains information on how various cryptology algorithms work including RSA public key encryption, ElGamal asymmetric cryptology and Elliptic Curve public key encryption.

I. INTRODUCTION

A. Types of Encryption

1) *Asymmetric Encryption*: Asymmetric encryption is also known as public key cryptography. These public key systems work by having a public key and a private key. The duty of the public key is to encrypt a message. The encrypted message is only able to be decrypted by an appropriate private key. One major down side of public key systems is that creating the key tends to be computationally expensive.

2) *Symmetric Encryption*: Symmetric encryption uses a single key for encryption and decryption. The keys on both sides are not always identical but both can encrypt and decrypt a message. Typically a symmetric encryption uses one of two methods stream ciphers or block ciphers. Stream ciphers encrypt data as it comes typically a few bytes at a time. For block ciphers a number of bits is taken at a time and is encrypted. If the data is too small to fit into a block it is padded so that it can fit into a block. Some common algorithms that use symmetric encryption are AES and blowfish.

3) *Hybrid Cryptosystems*: Hybrid cryptosystems use both asymmetric and symmetric encryption schemes. A hybrid system works by encrypting the message with a symmetric encryption key. This key is then encrypted using an asymmetric encryption technique. The symmetric key is then appended to the message. The benefit of using two encryption methods is that less time is needed for encrypting and decrypting the message due to using a symmetric technique plus since the symmetric key is encrypted using an asymmetric method we can still get the security of a public private key pair. A new symmetric key is generated each time to improve security further.

B. Dependent Algorithms

1) *AKS Primality Test*: The AKS primality test is a deterministic algorithm for determining if a number is prime. This is the fastest way to deterministically determine if a number is prime because it runs in polynomial time. The speed of this algorithm is $O(\ln^{6+E} p)$ or can be sped up even more to

$O(\ln^{4+E} p)$. This extra speed up will incur the problem of an infinitesimally small chance for the result to be ambiguous.

2) *Rabin-Miller Primality Test*: The Miller-Rabin primality test provides a probabilistic efficient algorithm for determining if a number is prime. The algorithm takes in a random odd number and determines if the number is prime by testing to see if the inputted number has any factors. This algorithm is very quick and runs in no more time than $(1 + o(1)) \lg(n)$ where \lg is logarithm base 2. This method is probabilistic so it can't be guaranteed that the number is prime but more likely than not it is prime.

3) *Diffie-Hellman-Merkle key exchange*: Diffie-Hellman-Merkle (or sometimes just Diffie-Hellman) is a key exchanging protocol that ensures a party's private key security is maintained. This is done by essentially "mixing" a private key with a publicly agreed-upon key. These mixed keys are then exchanged and both parties' private keys are now hidden within the mix. To obtain the same shared key, the mixed keys are then mixed again with their own private keys. Since multiplication is commutative, the order here doesn't matter and the result is that both parties have a shared key that is the same and the only public information that was conveyed were private keys hidden in a mix.

4) *Padding Schemes*: something about padding schemes would be good too

5) *Euler's Totient Function*: The $\phi(n)$ value counts the number of positive numbers relatively prime to n . For example $\phi(7)$ would be 6 because 1,2,3,4,5,6 are all relatively prime to 7. Prime numbers have a special property where the phi value of a prime number will always be the value of the prime number minus one.

6) *Extended Euclidean Algorithm*: This algorithm is an extension to the Euclidean Algorithm. The Extended Euclidean Algorithm is used typically for finding the coefficients of Bezouts identity. This algorithm can be adapted to find multiplicative modular inverses in a much faster way than the brute force method.

II. RSA ASYMMETRIC CRYPTOGRAPHY

The RSA asymmetric cryptography algorithm is named after its creators Ron Rivest, Adi Shamir and Leonard Adleman and was first thought of in 1973. At the time it was highly classified and was not declassified until 1997. The idea behind

this algorithm is to use large prime numbers to create a public key and a private key. Security for this algorithm rises from the difficulty of factoring large integers.

A. Private and Public Key Generation

The key generation process is a five step process.

1) *Creating Large Prime Numbers:* First two large prime numbers are calculated using a prime number testing algorithm such as AKS. These will be denoted p and q . We want both of these values to be relatively the same length digit wise but it can differ a bit.

2) *Compute n :* Computing n is just the product of p and q . The value of n will be used as the modulus for our keys and it's length is known as the key length.

$$n = p * q$$

3) *Compute Euler's Totient Function:* Now we must use Euler's totient function to give us a max value for our public key e . Luckily we are able to use algebra to make computing this value much easier.

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

4) *Create Public Key:* Now we need to create a value for e which will be our public key exponent for encrypting a message. For this value we want to use a reasonably small value without it being too small.

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

5) *Create Private Key:* Lastly since we have generated a value for e we will create d which will be used for decryption. To generate d we compute the modular multiplicative inverse of e (modulo $\phi(n)$). This value is commonly computed using the Extended Euclidean Algorithm. This is represented by the equation below.

$$d \equiv e^{-1} \pmod{\phi(n)}$$

B. Encrypting and Decrypting a Message

With the values of e and d determined the next step is to use this key pair to encrypt and decrypt a cipher text. To start a message we need to convert it's text into a number. This is done by the use of ASCII values and by using an agreed upon padding scheme. To compute a ciphertext c and restore it to a readable message we use the following equations respectively.

$$c \equiv m^e \pmod{n} \text{ and } m \equiv c^d \pmod{n}$$

C. Example of Usage

1) *Creating Large Prime Numbers:* To start with an example we will need to prime numbers for p and q . This example will be a simple one so smaller prime numbers are used. Let $p = 991$ and $q = 821$.

2) *Compute N :* We then use p and q to determine our value for n . Since $n = p * q$ this makes the value of $n = 813611$.

3) *Compute Euler's Totient:* Compute the Euler's totient value next.

$$\phi(813611) = \phi(991) \phi(821) = (990)(820) = 813611 - (1811) = 811800$$

4) *Create Public Key:* With these values created a public and private key can be created. Let $e = 7423$ which was arbitrarily chosen and now d is computed.

5) *Create Private Key:* Using the extended Euclidean algorithm to compute the modular multiplicative inverse we get $d = 788287$. The values of d and e are our private and public keys respectively and can now be used for message encryption/decryption.

Let our plain text message $m = \text{"Hi"}$. First we use an ASCII table to determine the value of m as a large integer. A simple way to do this is to concatenate each ASCII value together to form the integer. So $m = 72105$. Now that we have all the values we need we can encrypt and decrypt a message

$$\begin{aligned} c &\equiv 72105^{7423} \pmod{813611} \\ c &= 707473 \\ m &\equiv 707473^{788287} \pmod{813611} \\ m &= 72105 \end{aligned}$$

The ASCII values for our message "Hi" were 72 and 105 so that means it was successfully decrypted.

D. Discussion

The RSA method only works because of the presumed difficulty of factoring large numbers. When quantum computers begin to become legitimate Shor's algorithm will be able to factor large numbers in polynomial time. This advancement will break the RSA method of cryptology.

Ignoring the realm of quantum computers several attempts have been made to break RSA. A specialized machine for finding prime factorization of large numbers was designed and able to break a RSA key. With enough hardware simpler keys can be cracked but to combat this larger RSA keys can be used.

Another potential issue is that the random number generator may not always be 100 percent random. If someone was able to guess the correct seed then the work for prime factorization would be much easier.

III. ELGAMAL ASYMMETRIC CRYPTOGRAPHY

ElGamal public key cryptography is an alternative to RSA. This encryption algorithm was first described by Taher ElGamal in 1985. The ElGamal algorithm is dependent on the difficulty of computing discrete logs in a large prime modulus.

A. Private and Public Key Generation

The first step is to create a large prime number p . This p will be the exclusive max in our cyclic group. After p is generated an arbitrary number g that is between 1 and $p - 1$ is created. The variable g is the generator for our cyclic group. Then one last number x is selected where x is between 1 and $p - 1$, x will be used as the private key. We now compute a y value which will be the last part of our public key which is (p, g, y) . To compute y see the next figure.

$$y = g^x \bmod p$$

B. Encrypting and Decrypting a Message

When using ElGamal for encryption the cipher text will be twice the length of the plain text message. To encrypt first a random value k is chosen where k is between 1 and $q - 1$. Each time a message is sent the value of k should be changed. The message is then broken up into chunks and a padding scheme is followed so that the string is less than our value of p . Now for each chunk of the message m to be encrypted we create an ordered pair (a, b) by using the following equations.

$$\begin{aligned} a &= g^k \bmod (p) \\ b &= y^k (m) \bmod (p) \end{aligned}$$

A series of these order pairs are sent then to the receiver who can then decrypt them and assemble the message. To decrypt each ordered pair we can use the following equations.

$$\begin{aligned} s &= a^x * b \bmod (p) \\ m &= b * s^{-1} \bmod (p) \end{aligned}$$

C. Example of Usage

For this example we are going to have $p = 17$. Our generator $g = 6$. Using these values we can then finish our public key and private key. For this example our private key $x = 5$. This value was chosen at random. With these values we can compute the last portion of our public key which comes out to $y = 7$. Our message will be simple this time with $m = 13$. First before the message can be encrypted by a public sender the sender must create a random k value. In this example $k = 10$. Now we can encrypt the message by using the following equation.

$$\begin{aligned} a &= 15 = 6^{10} \bmod (17) \\ b &= 8 = 7^{10} (13) \bmod (17) \end{aligned}$$

Now we can decrypt a and b using the private key x .

$$\begin{aligned} s &= 16 = (15^5 * 8) \bmod (17) \\ m &= 13 = (8 * 16^{-1}) \bmod (17) \end{aligned}$$

The original message was 13 and after encrypting and decrypting 13 has been returned.

D. Discussion

One big concern with ElGamal is that a message m is going to be twice the size when encrypted. It is also slower than a symmetric algorithm for the same level of encryption so these features of algorithms makes ElGamal perfect for hybrid systems.

One advantage that ElGamal has is that it works with any cyclic group such that the computational Diffie-Hellman assumption holds for the group. This assumption makes it so the encrypting function is one way. If the decisional Diffie-Hellman assumption the ElGamal will achieve semantic security. If these assumptions hold then that cyclic group can be used with ElGamal encryption. So in the future if the discrete logarithm problem is made trivial a new cyclic group could be developed.

IV. ELLIPTIC CURVE ASYMMETRIC CRYPTOGRAPHY

Elliptic Curve Cryptography is based on the mathematics of elliptic curves. These are curves of the form

$$E = \{(x, y) | y^2 = x^3 + ax + b\}$$

The curve must also satisfy some other requirements: that $a, b \in K$. K is a field which can be defined as $\mathbb{R}, \mathbb{Q}, \mathbb{C}, \mathbb{Z}/p\mathbb{Z}$ but for cryptographic key-exchange usually $\mathbb{Z}/p\mathbb{Z}$ is used. These points are coupled with a point at infinity (O) and satisfy the inequality

$$4a^3 + 27b^2 \neq 0$$

Because of the usual basis on finite fields such as $\mathbb{Z}/p\mathbb{Z}$, there is also a requirement that the *field characteristic* not be 2 nor 3.

A. Group Operations: Addition

Elliptic curves are used in a unique way in regards to addition. To add two points on the curve, $P + Q$, the points are plotted and a line intersecting those points is drawn. This line will intersect a third point $-R$ and then that result point is reflected across the x -axis to obtain the result point R . Algebraically it is defined

$$\begin{aligned} P + Q &= R \\ s &= \frac{y_P - y_Q}{x_P - x_Q} \\ x_R &= s^2 - (x_P + x_Q) \\ y_R &= s(x_P - x_R) - y_P \end{aligned}$$

The slope is first calculated for the line that lies across P, Q and using that slope, the point R 's (x, y) coordinates are obtained. This is how addition is defined when using an elliptic curve.

B. Group Operation: Point Doubling

It is also viable to simply double the same point using this idea. The point is first plotted and then a tangent line drawn intersecting that point which will intersect the curve at a second point. This second point is $-R$ and is again reflected across the x-axis to obtain the result point R . Algebraically it is defined

$$\begin{aligned} 2P &= R \\ s &= \frac{3x_P^2 + a}{2y_P} \\ x_R &= s^2 - 2x_P \\ y_R &= s(x_P - x_R) - y_P \end{aligned}$$

Very similar as before, but for the slope the curve's a variable must be used in the calculation.

C. Special Considerations

For points that are directly vertical from each other, or a single point that has $y = 0$, the result is the point at infinity. Obviously such lines have no slope and thus do not intersect the elliptic curve at all.

D. Group Operation: Scalar Multiplication

There is also a unique way to obtain multiples of some point Q where $Q = kP$. This is done by repeated addition up to the multiple k

$$Q = P + P + \dots + P \text{ } k \text{ times}$$

Scalar multiplication is the basis for Elliptic Curve Diffie-Hellman key exchange. This gives way to what is known as the "Elliptic Curve Logarithm Problem" which basically states that given point Q finding the point kP is infeasible. The curve is paired with a generator point G and that point has an ordinal for its cyclic group, which would be the number of scalar multiplications until the result is the point at infinity.

E. Private and Public Key Generation

First, a client communicates with a server and they both agree on a type of key exchange to use. If the server has selected ECDH (usually Ephemeral Elliptic Curve Diffie-Hellman), a curve is selected and then both parties must select a private key. This key must be between 1 and n where $n = \text{ord}(G)$, the ordinal of the generator point's cyclic group. The client and server calculate the scalar multiple of the generator point and then exchange this. After the exchange, they again multiply the received key by their own private keys and this completes the exchange. Both parties now have a "mixed key" that are equivalent to each other and a third party

would be stuck trying to calculate the private key of either the client or server from the scalar multiple point, which is infeasible.

F. Discussion

Elliptic Curve Cryptography has clear benefits over other algorithms that use large prime numbers. The key sizes are much smaller for comparable security and the group operations are not easily reversible. A potential hacker would have to iteratively check every scalar multiple of some point to get the private key another party had selected and the massive curves created by the mathematics community in use in these functions are much too large to be able to check each one. By the time it was obtained the communication would likely be finished.

It should be noted there was some controversy over an elliptic curve implementation of a random bit stream for the RSA BSAFE suite. It was alleged that the NSA paid \$10 million to RSA for them to select a particular random bit generator for this suite, which was created by the NSA itself. The New York Times had reported that they had obtained documents that "appeared to confirm" the existence of a backdoor in this algorithm, potentially allowing NSA's BULLRUN decryption group to easily reverse some particular communication that used this algorithm. NIST has since recommended anyone selecting this algorithm use a different bit generator.

V. REFERENCES

A. General References

- mathworld.wolfram.com/CyclicGroup.html
- eprint.iacr.org/2012/195.pdf
- www.cs.princeton.edu/~dsri/modular-inversion-answer.php
- mathworld.wolfram.com/Rabin-MillerStrongPseudoprimeTest.html

B. RSA References

- doctrina.org/How-RSA-Works-With-Examples.html
- doctrina.org/Why-RSA-Works-Three-Fundamental-Questions-Answered.html

C. ElGamal References

- asecuritysite.com/encryption/elgamal
- homepages.math.uic.edu/~leon/mcs425-s08/handouts/el-gamal.pdf
- nku.edu/~christensen/1002mat584ElGamal%20example%20appendix

- math.la.asu.edu/~nc/elgamal.pdf
- cmssc414.wordpress.com/2009/09/23/el-gamal-examples/

D. Elliptic Curve References

- imperialviolet.org/2010/12/04/ecc.html
- arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/
- [youtube.com/watch?v=F3zzNa42-tQ](https://www.youtube.com/watch?v=F3zzNa42-tQ) (Robert Pierce)