A Study of Identity Based Encryption Systems

Samuel Petit
3rd Year Integrated Computer Science student at Trinity College Dublin, the University of Dublin
Email: petits@tcd.ie

***Abstract*—The abstract goes here.**

I. Introduction

Will be studying IBE. A type of Asymmetric paired keys encryption system. Charcterised with the fact that one of its keys identifies the recipient (email...phone no).

Makes a lot of sense to compare with classical systems currently in use such as RSA.

So let's compare both systems and try to explain why IBE is not as popular

A. Introduction to Public Key Cryptography

Public Key Cryptography, also called Asymmetric Cryptography is an encryption scheme. In other words, it is a method for encrypting messages. Unlike symmetric cryptography which uses the same key for both encrypting and decrypting messages, asymmetric cryptography uses two different keys such that one is used for encrypting and the other for decrypting information. In this context, a key is a string of characters that is used in some mathematical formula to turn a information (a message for example) such that it is impossible to understand in its encrypted form. Similarly, we would then also use a key to map the encrypted information back to its original, usable form. In asymmetric encryption, we typically call the encryption key the Public Key, and the decryption key the Private Key. The first asymmetric key cryptosystem was published in 1976 by Whitfield Diffie and Martin Hellman, previously, all useful modern encryption system used symmetric key encryption systems. While both systems still have their usages in todays world, asymmetric key encryption systems are now used on a daily basis throughout the world. With systems such as both HTTP over TLS and HTTP over SSL protocols, digitally signed files, bitcoin, encrypted messaging services and many other all using some for of asymmetric encryption.

1) How it works: Put simply, let's say we had a public key K_{public} and $K_{private}$. We would then obtain a ciphertext $cipher$ with the following formula:

$$cipher = K_{public}(message)$$

Similarly, we obtain the original message from the ciphertext with the following formula:

$$message = K_{private}(cipher)$$

2) Requirements for public key algorithms: In order for a Public key encryption scheme to be safe, we have a few requirements. The first one being ease of setup: it should be computationally easy to generate a pair of Public and Private keys.

Encrypting a message should also be computationally easy, this means that a sender X, with a message to send M and knowing the recipient's public key should be able to compute the ciphertext fairly easily.

Similarly, decrypting a message should be computationally easy, thus meaning that a receiver Y, with a ciphertext that was encrypted using his own public key should be able to obtain the original message using an easy computation.

In terms of keys, it should be impossible to obtain the private key from a public key, since these are, as its name suggests, public this would be a massive security issue.

Finally, it should also be impossible to find a message from the encrypted text and the public key used to obtain the ciphertext.

3) Different implementations of Asymmetric Encryption: Many popular protocols and systems can be used as examples of working asymmetric encryption. To start with, Identity Based Encryption (IBE) uses a set of asymmetric keys. Other popular protocols or systems include the Diffie-Hellman key exchange protocol which is used to exchange cryptographic keys over a non secure channel. RSA is a very popular cryptosystem which includes algorithms and functions to compute a set of keys as well as handling encryption and decryption.

B. An Overview of IBE

Explain what is exactly IBE now that we know more about public key encryption

1) IBE typical implementation: how is an IBE system implemented

2) What are its advantages and flaw?: pros and cons of IBE

C. An Overview of RSA

1) Key generation:

```
$p \leftarrow prime()$ {prime() returns a prime number}
$q \leftarrow prime(N)$
$n \leftarrow pq$
$\phi(n) \leftarrow (p-1)(q-1)$
$e \leftarrow coprime(\phi(n))$ {coprime returns a value coprime to $\phi$ where $1 < e < \phi$}
$d \equiv e^{-1} mod \phi(n)$
```

To explain a bit more about this algorithm, we start by generating randomly two prime numbers p and q. We obtain n from their product, n is used as part of the encryption and decryption function as we will see soon. Then we use Euler's totient function such as to count the positive integers up to $(p-1)(q-1)$, we could alternatively use Carmichael's totient function here with a slight change in algorithm, the same keys would be generated regardless. We then pick a value e such that it is positive, smaller than our value obtained from Euler's totient $\phi(n)$ function and coprime to it. We have then obtained the public key $\{e, n\}$. To compute our private key with simply compute the modular multiplicative inverse of e modulo $\phi(n)$ ($d \equiv e^{-1} mod(\phi(n))$) to obtain our private key: $\{d, n\}$.

2) Encryption: Given a public key $\{e, n\}$, we can then very simply encrypt a message M. This is done simply by computing:

$$C = M^e \mod n$$

This part is fairly simple as it only contains a single operation, though keep in mind that in practice e could be a very large number so M^e could be demanding on the device.

3) Decryption: Finally, for decryption, given that we have a private key $\{d, n\}$, and a ciphertext which was encrypted using the private key's corresponding public key, we can obtain the original message M using a formula very similar to that of the encryption:

$$M = C^d \mod n$$

4) Computing Signatures: RSA supports the use for signature. Signatures are used in order to make sure a message was sent by a specific person.

The first step for verifying signatures is for the sender to generate one. Using the senders private key and the message to send we can obtain the signature S:

$$S = M^d \mod n$$

We would then send the signature S with the ciphertext to the recipient. Note that we must respect $M < n$ here too.

5) Verifying signatures: The person receiving the message can then verify the identity of the sender. If we use the senders public key in the following formula to obtain the original message, then we can be certain of the senders identity.

$$M = S^e \mod n$$

Once again, $M < n$ is required.

6) Security: RSA's security relies on the fact that factoring numbers is complicated and expensive. For instance, let's assume that we have a public key $\{e, n\}$, we could in theory find the values for p and q from the keygeneration algorithm by factoring n (recall that $n = pq$). Then all we would have to do is use Euler's theorem to obtain $\phi(n)$ and from there we could find e and finally obtain the private key d by solving the equation $ed = 1 \mod \phi(n)$.

In practise though, we pick very large values for p and n, factoring very large prime numbers is very hard and trying to obtain p and q using a brute force method would take a very long amount of time. (HOW MUCH ???)

D. Math Systems at the core of Public Key Encryption

Goal: Explain that at the core of public key encryption, underlies many important mathematical concepts.

1) Discrete logarithms: The difficulty of breaking a Public Key encryption system is based on the Discrete Logarithm problem (DLP). Given a multiplicative group G_n^* of order n, Let's then define the element α such that it is a subgroup of G_n^* (i.e. $\alpha \subseteq G_n^*$). We then have α, a subgroup of G_n^*, note that α is also cyclic of order n (same as G_n^*).

The Discrete Logarithm Problem is then defined as, given $\beta \in \alpha$, find x such that $0 \leq x \leq n-1$.

$$\alpha^x \equiv B \mod p \tag{1}$$

It is important to note that not the discrete logarithm problem is not always hard to solve. In the context of encryption we choose a group Z_p^* with p a prime number which makes this problem very hard and expensive to solve. It is also worth noting that not all primes are safe to use and some primes make this problem much easier to solve, using methods such as the Pohlig–Hellman algorithm. In order to be safe, one can follow the rule that a prime must be of the form $2p+1$, where p is a large prime number.

2) Prime numbers: As we have started to introduce prime numbers earlier, we can't help but notice that prime numbers are at the core of many concepts being used in the types of asymmetric encryptions we are covering here. There is a reason for that, it is the fact that primes are very easy to multiply together, however factorising a number into two prime numbers is extremely computer intensive. Much more so than it would be if the numbers weren't primes.

So then, how do we check that a number is prime ? Do we try all possible factors until we know we have a prime number? This approach would work, however, it would be terribly expensive when we are generating very large primes. Thankfully, there are mathematical theorems we can use to make this check easier.

Two famous mathematical methods to check if a number is prime are Miller–Rabin and Fermat's primality test. Fermat's little theorem is the basis for Fermat's primality test. It states that given, a, an integer and p a prime number where a is not divisible by p, we have:

$$a^p \equiv a(\mod p)$$

Thanks to this equation, we can use it to find Fermat's Primality test:

$$a^{p-1} \equiv 1(\mod p)$$

We use it by picking two integers a and p such that a is not divisible by p. If the equality holds, then p is a prime number.

The way we generate large prime numbers is by generating integers and testing if the generated integer is prime. We can compute the probability of picking a prime number given a size. For instance if we wanted to find a prime number a with a bit size of 2^{1024}. We would then have a probability of picking a prime number of

P(a is prime) $= \frac{2}{\ln(2^{1024})} = \frac{2}{1024\ln(2)} = 0.00281776375$

This probability is based on the more general Prime number theorem which statistically describes the distribution of prime numbers. Its distribution is described as such:

$$\pi(N) \sim \frac{1}{log(N)}$$

Where $\pi(N)$ is the prime counting function: it computes the amount of primes that are less than or equal to N.

3) Square and multiply: Something you may have noticed is that some of these operations require computing many exponents, given the fact that we generaly pick very large numbers for encryptions, finding a way to compute massive exponents in a way that is more manageable for computers would help efficient computations quite drastically. This is exactly what the square and multiply algorithm enables, it is essentially an algorithm for computing exponents iteratively. This algorithm relies on the fact that exponent value can be broken down into the multiplication of multiple terms:

$$x^n = \begin{Bmatrix} x(x^2)^{\frac{n-1}{2}}\text{, if n is odd} \\ x(x^2)^{\frac{n}{2}}\text{, if n is even} \\ 1\text{, if n = 0} \end{Bmatrix}$$

This algorithm can be implemented such that is it based on binary values, by using the bit values to determine which powers are computed. Here is a quick example using x^5. 5 in binary is 101. Iterating bit by bit:
Initialisation: result $=$ 1
Step 1: result $=$ result^2; $(= x^0)$; bit 1 is 1; result $=$ result$x$; $(= x^1)$
Step 2: result $=$ result^2; $(= x^2)$; bit 2 is 0, so there is no computation this step.
Step 3: result $=$ result^2; $(= x^4)$; bit 3 is 1, result $=$ result$x$; $(= x^5)$

The pattern from this example is fairly straightforward, for every bit iteration, square our current result, then multiply by x if the bit at the current position is 1.

Note that side attacks are possible on this particular algorithm, due to the fact that based on the iteration being a 0 or a 1 a single or two operations are executed. For instance if a hacker could access the power drawn by the device overtime then we could notice that more power would be drawn to compute an iteration of this algorithm with a bit value of 1 in comparison to a 0. In some cases we can monitor these from the frequencies emitted by a devie and thus we may be able to obtain a private key using such methods.

There are ways to counter this, known as padding (explained later and add a bit here).

4) Chinese remainder theorem: The chinese remainder theorem plays a big role in optimising the amount of work required for decrypting a message.

Remember from the RSA setup stage that we compute $n = pq$, where n is part of the public key. Also remember that decryption requires a ciphertext C and a private key $\{d, n\}$ to obtain the plain message M. $M = C^d \mod n$.

We can use the Chinese remainder theorem to split the computation of modular exponentiation, in the case of decryption using p and q.

$$M_P = M \mod P$$

$$M_Q = M \mod Q$$

We can even push the optimisation furter:

$$M_P = M \mod P = (C^d \mod n) \mod p$$

$$= C^d \mod p = C^{D \mod (P-1)} \mod p$$

We can do these simplifications thanks to the fact that $n = pq$ thus

$$(C^d \mod n) \mod p = C^d \mod p$$

And thanks to Fermat's little theorem giving us the final optimisation. Note that $D \mod (p-1)$ can be computed on setup instead of during decryption thus making this approach much faster for message decryption.

Once we have computed both M_q and M_p, we then need to find such that M stasfies both:

$$M \equiv (C^d \mod n) \mod q$$

$$M \equiv (C^d \mod n) \mod q$$

Using the fact that p and q are relatively prime and because of the chinese remainder theorem, we can directly assume that:

$$M \equiv (C^d \mod n) \mod pq = M \equiv C^d \mod pq$$

Thus obtaining the decrypted message M which is what we were looking for.

5) Extended euclidean algo to find gcd, coefficients: euclidean algo

E. IBE vs RSA

Compare the two

1) Attacks on IBE Public key systems: Go through possible attacks on Public key systems

2) Are these attacks all possible on IBE? is it better or worse with IBE?: go through how IBE protects from attacks differently than rsa systems

3) which is most safe?: expalin which system is safest

II. Conclusion

The conclusion goes here.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.