Foundations of Mathematics — Math 300 Section 905

Stephen Shell

Term Paper Final Draft — 30 November 2020

Public Key Cryptography

1. Intro

Imagine you are a New York undercover investigator, and you walk into the large, open, and public area that is Grand Central Station. You have been assigned to exchange a confidential file with another undercover agent. You identify the agent, but you are hesitant to simply walk up and hand over the file. The enemy, a highly dangerous hitman, has henchmen stationed everywhere, ready at any moment to swoop in and snatch the file away. You wish there was a way to encrypt the file, so that if it was stolen, they would not be able to read it, but you have never met this agent before, so you have no predetermined secret language. Even if you were to encode your message, you would still have to hand over a decoder or cipher, which could be stolen along with the file.

At this point, the best you can do is just hope that no one will steal the file, or you could somehow get into a more private place, unlike Grand Central Station, to give the other agent your file or a cipher to decode all of your future messages. This is actually the idea known as *private* key encryption. It is a method of sending encrypted messages assuming the recipient already has the means of decoding the encrypted message. But it is not good enough for a top secret undercover investigator such as yourself. No, what you need is *public* key encryption.

Public key encryption or asymmetric encryption is similar but different from private key encryption or symmetric encryption. While private key encryption involves only one shared key, one for both coding *and* decoding (or both locking *and* unlocking) a secret message, public key encryption involves four keys, or two keys for each person; one for locking and a different one for unlocking. Hence the name, "asymmetric," because private key encryption uses the same key for both parties, but public key encryption uses different keys on each side, causing a lack of symmetry. To sum up, private key or symmetric encryption first requires the sender and recipient to both share the same key, but public key or asymmetric encryption allows anyone to send a private message to anyone without it being stolen.

Public key encryption is essentially hacker-proof; even computers that can decode the most complex ciphers and iterate through thousands of keys/combinations per second will fail to crack public key encryption. Yes, this is significant! Many of us use public-key encryption daily! For example, anytime you sign a document digitally, send an email, transfer Bitcoin, or access secure websites via https, you are using public key encryption!

The field of public key cryptography is incredibly useful and growing quickly. Cryptographers must pioneer strategies against the newest hacking methods to keep our information safe. It is the battle of good versus evil that no one sees but everyone benefits from. The advancement of cryptography by means of public key encryption is vital to the security of our internet data.

In this paper, I will explain public key encryption using examples such as the classic Alice and Bob illustration, I will describe specific methods of public key encryption, including the mathematicians and mathematics behind RSA, and I will conclude with information regarding the future of cryptography.

2. Public vs. Private Key Encryption

Let us go back to differentiating private and public key encryption using the classic Alice and Bob illustration. It goes like this: Alice and Bob are friends who share a private key. This key can code and decode information, and no one else has this key. So, when Alice wants to send a private letter to Bob, all she has to do is lock or encode the message with her key, so Bob can unlock or decode her message with his identical key. A similar illustration involves a bank and its customers: A bank wants to send money to its customer, so it makes two identical keys; one for the bank to keep, and one to give to the customer. Let us say the customer wants to withdraw some cash digitally. The bank then puts money into a digital box, locks the box, and sends the box to the customer. The customer uses their key to unlock the box and retrieve their cash.

At first, this may seem viable, but there are three potential issues with private key encryption: First, it is built on the assumption that Alice and Bob have already met up in person privately to make sure they have the same key. This is not always feasible in the real world of sending information. Senders and recipients may live hundreds of miles away or not be able to meet up for other reasons. Second, a bank could feasibly have specialized keys and boxes for a few customers, but for hundreds, thousands, or hundreds of thousands of customers, a bank would have to keep track of just as many keys and boxes without mixing them up. Third, in either case, if somehow a hacker could steal the private key between two parties, not only could they read messages sent both ways, so from party A to party B and from party B to party A, but they could also encode their own fake messages and pass them along as coming from someone else! Essentially, a hacker could be a middle man in a private conversation, intercepting and decoding every message, and either re-coding them or encoding their own fake messages, possibly without anyone having a single clue!

These issues can make private key encryption inconvenient, over-complicated, and dangerous! However, all of these issues are resolved with public key or asymmetric encryption. This is how public key encryption works, using the same two examples from above: Alice and Bob want to exchange information, but have no shared key. They each have their own private key for decryption and their own public key for encryption. So Alice encrypts a message with Bob's public key. Everyone has access to Bob's public key and can send him encrypted messages with said key. However, these messages cannot be decoded by Bob's public key. Such messages can only be

decoded by Bob's private key, which he alone owns access to. In this manner, Bob is safely able to receive and decoded Alice's coded message. Similarly, Bob can send Alice a message, encrypted with Alice's public key, which she can decode with her own private key. Now, going back to the bank example, A bank has their own public key, so all customers can use this single key to lock their boxes for depositing money. Now, the bank has no need of keeping track of thousands of keys for unlocking these boxes. They have one single, private key.

As you can see, many of the issues with private key encryption are resolved with public key encryption. First, it is no longer necessary to meet up and safely exchange keys, since private keys sent over the internet could be intercepted by hackers. Second, there is no massive stockpile of keys needed for the bank to unlock customer's boxes. And third, if a hacker somehow steals Alice's private key, they can only decode messages sent to Alice. They cannot decode messages sent to Bob, because they do not have his private key, thus making a middle man unable to both receive messages and send fake messages both ways.

3. Diffie-Hellman Cryptography

Ralph Merkle, one of the inventors of public key cryptography, said it best:

> "You encrypt the message using the public key which I gave you over the open communications channel. The eavesdropper knows the encryption method, which is publicly revealed, they see the secret encrypted message coming back, and now they want to work it backward... but they can't."[1]

Two other mathematicians, Whitfield Diffie and Martin Hellman, co-invented public key cryptography with Merkle in 1976. Since their patented idea, mathematicians have developed mathematical methods to bring this idea to life, one of the most important being the Diffie-Hellman key exchange.

Diffie and Hellman published the Diffie-Hellman key exchange in 1976. Before I get into the mathematics behind what these mathematicians created, it is important to understand it conceptually using an illustration involving colors.

The color illustration proceeds as follows: First let us imagine an area divided into three columns. Now let us take Alice and Bob as an example once more. We will put Alice in the first column, which represents information private to her. We will put Bob in his own private domain in the third column. This leaves the middle column to represent the public domain where all information must travel through the internet to get from Alice to Bob and vice versa. Hackers, eavesdroppers, or whatever you prefer to call them reside in this public domain or this open communications channel.

Next, we will give Alice and Bob their own unique private keys represented by specific colors, red and blue. The public domain will also have a key, yellow, but this key of course will be public

and available for everyone to use. Alice wants to send Bob a message, but first, she needs to create a brand new private key which she will share with Bob. So she takes her private color, red, and mixes it with the public color, yellow, to make orange. Similarly, Bob takes his private color, blue, and mixes it with the public yellow to make green.

At this point you may interject, "But the eavesdropper in the public domain should know that Alice's color is red if it mixes with yellow and makes orange? Well yes, but the eavesdropper does not know how red. Alice's red color is a value, when mixed with another value (the yellow color), creates a new color value that makes it impossible to cipher. Go ahead, mix red and yellow and then try to unmix them. It is not possible!

So now, there is an orange-ish color that Alice shares with the public domain, and there is a greenish color that Bob shares with the public domain. Alice gives her orange to Bob, and Bob gives his green to Alice. Then Bob mixes Alice's orange with his private blue, and Alice mixes Bob's green with her private red. Theoretically, if you mix the exact same amounts of the exact same three colors, you should end up with a specific color no matter when or what order you mix them. So after Alice and Bob are done exchanging and mixing, they should share a dark, orange-ish greenish color of some sort. Whatever this shared color is, it is the same for Alice and Bob, and it has not crossed into the public domain. All the eavesdropper has to work with is the public yellow, Alice's orange, and Bob's green. He does not know Alice's red, Bob's blue, or Alice and Bob's shared concoction of red, yellow, and blue. In fact, not even Alice and Bob know each other's own private colors!

Now that Alice and Bob have this shared private key, they can now exchange information by private key encryption, encoding and decoding with the shared key. Essentially, the Diffie-Hellman key exchange is technically a public key cryptosystem that provides a means for a private key exchange, thus making private key encryption possible.

4. Diffie-Hellman Mathematics

So now that we understand the concept of the Diffie-Hellman exchange, we can dive into the mathematics behind it. How can we use numbers to mix like colors so that unmixing is impossible? Diffie and Hellman proposed this: Represent the public color by two numbers g and n. Let g be a small prime number, and n a very large number between 2,000 to 4,000 bits (...as of now. This number will continue to grow in the future, as computers are able to process larger numbers more efficiently). Next, represent Alice and Bob's private colors with their own private numbers a and b between 1 and n. These three colors are our red, yellow, and blue by which we create the other colors.

Now, we will begin mixing. Take Alice's red a and "mix" it with the public's yellow g and n with the formula $g^a \bmod n$. This is Alice's orange. Bob's green is similar. Take Bob's blue b and "mix" it with the public's yellow g and n with the formula $g^b \bmod n$. These colors are now

public, and the only way to retrieve a or b from these formulas is by a brute force algorithm which could take a computer years and years to find (I will explain more later). How could it be so complicated? It has to do with the nature of modulo.

With an equation like $g^a \bmod n$, you are raising a small prime g to a large a power to get an extremely massive number. Then, you are dividing that by the large number n, to get a remainder between 0 and n. Modulo is very difficult to reverse engineer. Let us think about modulo with small numbers: Let g be 2, a be 4, and n be 7. This gives us $2^4 \bmod 7 = 16 \bmod 7 = 2$. If we were simply dividing by 7, then reverse engineering would be simple. We would just multiply by 7. However, modulo muddies the water a bit. While $14/7 = 2$, and only 14 could produce a 2 when divided by 7, many numbers could produce a 2 when you mod it by 7: $2 \bmod 7 = 2$, $9 \bmod 7 = 2$, $16 \bmod 7 = 2$, $23 \bmod 7 = 2$, and so on.

So in this scenario, the only way to algebraically find a is to systematically raise g to every number between 1 and n and divide by 7 until you find one that gives you 2. With small numbers, $g = 2$, $a = 4$, and $n = 7$, this is not difficult:

1. Try $a = 0$: $(2^0 \bmod 7) = (1 \bmod 7) = 1$
2. Try $a = 1$: $(2^1 \bmod 7) = (2 \bmod 7) = 2$
3. Try $a = 2$: $(2^2 \bmod 7) = (4 \bmod 7) = 4$
4. Try $a = 3$: $(2^3 \bmod 7) = (8 \bmod 7) = 1$
5. Try $a = 4$: $(2^4 \bmod 7) = (16 \bmod 7) = 2$
6. Try $a = 5$: $(2^5 \bmod 7) = (32 \bmod 7) = 4$
7. Try $a = 6$: $(2^6 \bmod 7) = (64 \bmod 7) = 1$
8. Try $a = 7$: $(2^7 \bmod 7) = (128 \bmod 7) = 2$

That only took 8 steps with only a few calculations necessary. We have narrowed Alice's a down to either 1, 4, or 7, since those value for a yielded 2 at the end. So yes, it is *possible* to narrow down Alice's a. The Diffie-Hellman method is not impossible to decode, but for 4,000 bit numbers n, a becomes *nearly impossible* to find.

In this way, modulo is incredibly easy to calculate when you know a. You divide and take the remainder. But to reverse engineer, you need a computer to run through every possibility between 1 and n, which becomes increasingly more difficult to do as n increases. Just as it is easy to mix colors, it is essentially impossible to unmix them.

Now let us go back to Alice and Bob. We have created Alice's orange, $g^a \bmod n$, from her red a and the public's yellow g and n. In a similar fashion, we have mixed Bob's blue, $g^b \bmod n$. Now,

Alice sends Bob her orange, and Bob sends Alice his green. Alice takes Bob's g^b mod n and raises g^b to her a to get $(g^b)^a$ mod n or g^{ba} mod n. Similarly, Bob mixes Alice's g^a mod n with his b to create $(g^a)^b$ mod n or g^{ab} mod n. Because of the Power Rule of exponents, we know that g^{ba} mod $n = g^{ab}$ mod n. Therefore Alice's current color, which was Bob's green mixed with her red, is the same as Bob's current color, which was Alice's orange mixed with his blue.

So now, Alice and Bob both share a private key which is the formula g^{ab} mod n. Because of the cyclical property of modulo, just as g^a mod n and g^b mod n equal some number between 1 and n, g^{ab} mod n also yields some number between 1 and n. Interestingly, the number that g^{ab} mod n yields has no relation to the numbers that g^a mod n and g^b mod n yield. So, even if someone figured out a, they still would have incredible difficulty finding b. This is the safety and security of the Diffie-Hellman key exchange.

5. Conclusion

Before I conclude, I would like to clarify what I mean by saying that it is nearly impossible for a computer, even with the best hardware and software, to decode a Diffie-Hellman encrypted message. We know that coding a message using a small prime g, a large number n, and modulo can be done quickly, while decoding takes a long time. The only way to decode is by a brute force algorithm, which, in this case, means plugging in numbers from 0 to n by trial and error. As n increases, the time it takes for the computer to run through the algorithm increases as well. This is called computer runtime. When we used the Diffie-Hellman method with small numbers, it did not take long to decode the private key. A computer could have done it in a second, which is a very short runtime. But using the Diffie-Hellman method with large numbers could take years!

According to a 2015 article, it would take 100 million dollars of computing to crack a Diffie-Hellman key that took 1024 bits to create. It also says that increasing the bit size to 2048 would make it 10^9 harder to crack. As technology advances, computing power will quicken, and we will need longer bit numbers or better encryption strategies. There are many other public key encryption strategies in use such as RSA, the Paillier cryptosystem, Elliptic-Curve Cryptography, and the Cramer-Shoup cryptosystem. Of these, RSA has been considered insufficient, which is why Microsoft, a very large company, uses Elliptic-Curve Cryptography.

Cryptography is fighting the battle for the security of our online data. It is defending against local and foreign hackers. It is an ever-changing and ever-growing field creating new jobs every day. I find it so amazing how concepts such as exponential and modular arithmetic can blaze the trail for encryption methods which keep our digital data safe. We owe cryptographers and mathematicians our gratitude for guarding the future of the internet.

Sources

1. @book Galbraith, author = Galbraith, S. D., title = Mathematics of Public Key Cryptography, publisher = University Press, Cambridge, year = 2012

2. @article Hellman, author = Hellman, Martin E., title = An Overview of Public Key Cryptography, publisher = ICEE Communications Magazine, year = 1978

3. @article Rivest, author = Rivest, Ronald L., title = RSA Chips (Past/Present/Future), publisher = Springer-Verlag Berlin Heidelberg, year = 1985

4. @article WeakDH, author = various authors: D. Adrian, K. Bhargavan, Z. Durumeric, P. Gaudry, M. Green, J.A. Halderman, N. Heninger, D. Springall, E. Thomé, L. Valenta, B. VanderSloot, E. Wustrow, S. Zanella-Béguelink, P. Zimmermann, title = Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice, publisher = various publishers: INRIA Paris-Rocquencourt, INRIA Nancy-Grand Est, CNRS, Université de Lorraine, Microsoft Research, University of Pennsylvania, Johns Hopkins, University of Michigan, year = 2015

5. @video Numberphile1, title = Public Key Cryptography - Computerphile, year = 2014 url = $https://www.youtube.com/watch?v = GSIDS_lvRv4$

6. @video Numberphile2, title = RSA-129 - Numberphile, year = 2017 url = $https://www.youtube.com/w$
$YQw124CtvO0$