Cryptography

How to Protect Your Data

Encryption is the act of changing information in such a way that only people who should be allowed to see the data are able to understand what the information is. The encrypted data can ideally only be read by trusted parties and look like a mess to everyone else. The simplest and oldest form of encryption is a Cipher!

Example Ciphers

1. **Caesar Cipher**: Probably one of the oldest, and easiest to break, ciphers.
    a. A key is chosen by the parties, which will be a number between 0 and 25. Then, each letter in the message is shifted forward or backward by that key to receive the encrypted message.
    b. For example, “This is an example” with a forward shift of 1 gives the encrypted message: “Uijt jt bo fybnqmf”.
    c. It is very easy to break as you can just apply all possible 26 forward and backward shifts and only one key will likely give an intelligible answer.

2. **Affine Cipher**: A substitution cipher where there is a formula for the substitution
    a. Each letter is assigned to a numerical value (usually starting the first letter at 0 and so on). Then a linear function is applied to the numerical value modulus the number of alphabets and then converted back to alphabet.
    b. So, let the numerical value of a certain alphabet be x. Then, the new value will be $(a*x + b) \bmod m$, where m is the number of alphabets, a and b are the keys picked by the two parties. This new value is then converted back to an alphabet.
    c. Also, notice that a Caesar cipher is an Affine cipher with a = 1.
3. **Vigenere Cipher**:
    a. This is a more complicated cipher than the ones we have seen already since we apply different shifts to each letter based on a pre-chosen key.

Here is an article with really good pictures that explains the method really well!

4. **Hill Cipher**: This technique requires a knowledge of Matrices. We cover the basics of Matrices here.
    a. The message, with n letters in it, is written as an n-element column vector. Each element of the vector is a number representation of the corresponding letter.
    b. The key is an n x n matrix, with random numbers in it.
    c. You then multiply the matrix with the vector and then modulo each element by 26. This basically means divide each element by 26 and replace the element with the remainder of that division.
    d. Then you convert the column vector back to letters and that is your encrypted message.
    e. Decryption is done in a similar way except you multiply the inverse of the key with the vector instead.

These are only a few important ciphers. However, we chose them because they cover most of the important topics in ciphers like shifting, modulus, matrices etc. but we recommend that you check out other ciphers too!

Hashing

Hashing is a very important concept in Computer Science and will function as a gateway from encrypting data to more foundational concepts of Cryptography in Information Security.

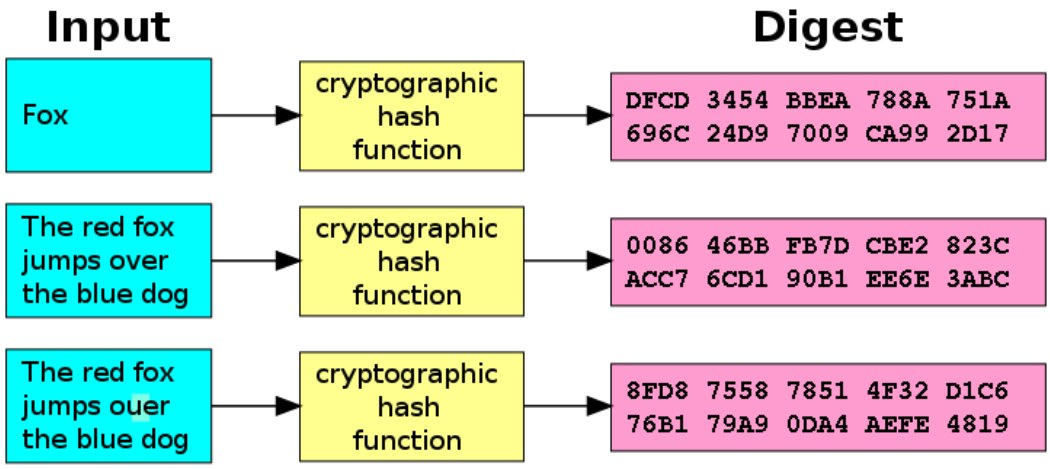
Hashing requires two things:

1. A Hash function: A mathematical function that takes an input and produces an output
2. A message: A string of symbols that is given as input to the hash function.

There are three main requirements of a good hash function:

1. It should be easy to calculate the hash value (output of hash function) given a message.
2. It should be very hard (ideally impossible) to calculate the original message given a hash value.

3. It should avoid collisions i.e. no two messages should have the same hash value.



Above, the input is the message and Digest is the hash value. In general, a hash function can be something as simple as f(x) = 1, where x is the input message. It is very easy to calculate and it's impossible to know what the message was given the hash value. However, every message has the same hash value and thus, this function is useless as it serves no purpose essentially.

Similarly, f(x) = x, is a horrible hash function! While it is easy to calculate, and we definitely avoid collisions, it is trivial to figure out the original message given the hash value, they are the same!

This is why these three conditions are chosen to make a good hash function. This is an in-depth exploration of hashing and how it is actually implemented in computers. However, it may be too complicated as it uses data structures, big-oh and other complicated Computer Science topics to explain a lot of important information on hashing. You have been warned!

Public Key Cryptography

Did you notice that there is a glaring issue with the ciphers that we discussed earlier? Everytime, we need to have a predetermined key and if someone gets access to the key, then they can easily decrypt the information and view all our secrets.

This wouldn't be as big an issue if both the parties live close-by and can meet to decide the key. But what if they live in different countries? What if one of them writes

the key down and then loses the paper? You don't want to have to meet everytime you want to change the key. With the advent of the internet, we need a new way encrypt information. You shouldn't have to go meet someone in person everytime you want to send them a message online. This gave birth to the idea of Public Key Cryptography.

Let's identify three individuals, Alice, Bob and Carl. Alice and Bob want to communicate over the internet and Carl wants to eavesdrop on their conversation! Alice and Bob need to create a key that -

1. Only they know
2. Can be decided over an unsecure network

At first, this seems impossible but there are many sophisticated techniques using some very simple Math, that can help Bob and Alice out. They will essentially need to decide their own personal keys that they keep private and a public key that everyone will have access to but do something clever so that only they can read the messages. One such technique is called the "*Diffie-Hellman Key Exchange*".

Diffie-Hellman Key Exchange

Here is how it works -

1. Alice and Bob publicly agree on two prime numbers, M and B.
2. Now, they both choose their own numbers called their private key, a and b respectively, and raise the base number B to the power of their respective private key modulo P, K_a and K_b respectively. They now publicly exchange these numbers. Note: So far, only *a* and *b* are unknown to Carl.

$$K_a = B^a \ (mod\ P)$$
$$K_b = B^b (mod\ P)$$

3. Now, they both raise the key they received to the power of their own private exponents.

Bob does : $K = (B^a mod\ P)^b = B^{ab}\ (mod\ P)$

Alice does: $K = (B^b mod\ P)^a = B^{ab}\ (mod\ P)$

4. Bob and Alice now have the same number *K* and Carl has no way of creating the key because he does not have access to *a* or *b*! Thus, *K* is a completely secret key that only Bob and Alice can know and Carl has no way of reproducing this key.
5. Alice and Bob can now begin encrypting their messages with this key and Carl can no longer eavesdrop!

Warning! The Diffie-Hellman approach is not perfect and has weaknesses that a hacker could take advantage of.

An important aspect of Public Key Cryptography is that it is an Asymmetric Key Encryption. This means that the key used to encrypt the message is different from the key used to decrypt the message.

In general, everyone has a pair of keys, called a key-pair, and these keys are mathematically linked in such a way that a message encrypted with one of them can only be decrypted with the other. You set one of them as your public key, which you publish to the world, and the other as your private key, which you tell no one. Now, this brings up two very important concepts in Cryptography - Authentication and Encryption!

Authentication: This refers to the authenticity of a message or basically is a way to confirm whether or not the message is from a particular user or someone else trying to copy that user. This is very important, because you cannot guarantee a secure connection if you don't know who the messages are coming from.

Encryption: We have covered this enough in our discussion but it essentially the act of encoding the message so only a particular user can read it. This is important because you cannot guarantee a secure connection if anyone can read your message.

How does Public Key Cryptography guarantee a secure connection?
As we discussed, to guarantee a secure connection, we must be able to successfully encrypt a message and ensure its authenticity. Our technique will be as follows -

1. When sharing a message, encrypt it with your partner's public key. This way, we are ensured that only the intended recipient can read the message since only they can decrypt the message using their private key (which no one else can figure out). Thus, we have ensured Encryption!
2. Also, encrypt your message with your own private key. Why? Well, since only you can encrypt a message with your private key, if a message sent out to the world can be decrypted by your public key, we are ensured that the message inside must be sent by you. This ensures the messages Authenticity!
3. Thus, when you use both techniques, you are guaranteed a secure connection without having to meet and physically exchange keys!

This video provides a really good explanation of this concept and it may be easier for you to follow along pictures than text. Both our discussion and the video, are abridged versions of these very complex concepts but will give you a basic idea of how this works. If you wish to know more, there are tons of great resources on the internet. Just go find them!

RSA Cryptosystem

Developed by Ron **Ri**vest, Adi **S**hamir and Leonard **A**dleman, the RSA encryption is one of the most commonly used type of public key cryptography. A *Cryptosystem* is a collection of similar algorithms needed to implement a particular service.

Here's a brief version of the algorithm:

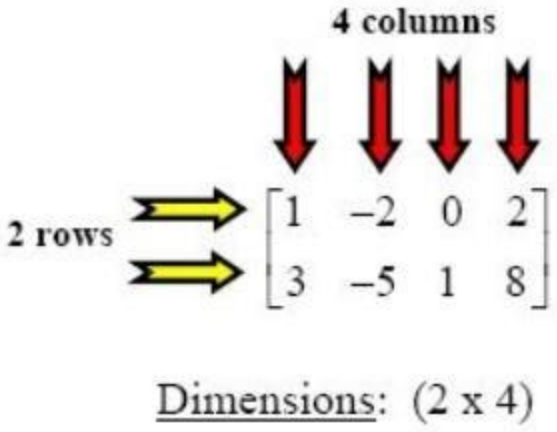
1. Privately select two large prime numbers, *P* and Q. If someone gains access to these, then you are vulnerable to attack.
2. Multiply the two numbers to create n = P x Q. This is your public key.
3. Calculate Φ(n) such that Φ(n) = (P – 1) x (Q – 1).
4. Choose a number, e, such that 1 < e < Φ(n).
5. Your total public key is (*n*, *e*).
6. Calculate d = (k*Φ(n) + 1)/e for some integer k. d is your private key!
7. Your total private key is (n,d).

To send a message m, the other person needs to calculate $x = m^e \pmod n$ and send x to you. This is the decrypted message. Now you encrypt it by calculating $x^d \pmod n$. This will give you back the original message m. The best way to use this algorithm is for the other person to sign the message with your public key and his own public key to ensure Authenticity and Encryption.

Mathematical Basis of Cyber Security

Matrices

A matrix is a mathematical tool to help store and manipulate numbers in an easy way!



Above is the example of a 2-by-4 matrix, which means it has 2 rows and 4 columns. The first number in dimensions is number of rows and second is number of columns. A Matrix can also have a name. Let's call this one 'A'. Each number is called an element of the matrix and can be identified by a unique pair of numbers or its position. For example, -5 is in the second row and second column, so we can call it A_{22}. Similarly, $A_{11} = A_{23} = 1$, where the first subscript is the row position and second number is the column position.

Matrix Algebra!

Matrix Addition and Subtraction are very similar to normal Addition and Subtraction, with the added requirement that the dimensions of the matrices must be the same. Then, you just have to add/subtract the corresponding elements to create a new third matrix.

Multiplication on the other hand, is a lot more involved. There are two kinds: Scalar and Matrix Multiplication. Scalar Multiplication is multiplying a Matrix by a number and for that, you just multiply each element by that number. Matrix Multiplication is multiplying a Matrix by another Matrix and here is a great tutorial on how to do Matrix Multiplication. NOTE: Order matters for Matrix Multiplication so $A \times B \neq B \times A$.

Matrix Division isn't really a thing but there is a way to sort of implement it, which we will talk about in a bit.

Important Matrices!

There are many important matrices that can be useful but the three most important for you to know are - Identity Matrix, Zero Matrix and Inverse Matrix.

Represented by the letter 'I', the *Identity Matrix* is the Matrix equivalent of '1' in that A x I = A. It is a square matrix (# of rows = # of columns) and all elements are 0 except for the elements on the diagonal, which are all 1.

The *Zero Matrix*, represented by the symbol 'ø', is the number equivalent of 0. It is a matrix of all 0's and functions in a similar way.

The *Inverse Matrix* doesn't refer to a specific Matrix but is a more general term for a matrix that when multiplied by another specific matrix, gives the identity matrix. Hence, it is a Multiplicative inverse of a particular matrix. For example, numbers have multiplicative inverses too. 2 has an inverse of ½ since, 2 x ½ = 1. Similarly, some matrices can have an inverse matrix represented by a superscript of -1. Here is a great article that talks about inverse matrices in depth and explains how one calculates it. This is also how we kind of do division in matrices, since multiplying by the inverse of a matrix is sort of like dividing by the original matrix!

There is a lot more to learn about Matrices. They are very important in solving systems of linear equations, in Computer graphics and in many cyber security problems! We cover a cool use of Matrices in Hill Ciphers here.

Modular Arithmetic

This is another foundational topic in Mathematics and Computer Science. This is arithmetic bound to a particular Modular environment. Now, what does this mean?

The modulus operator, usually written as '%' sign is an operation that means the remainder of. For example, 5 % 2 = 1, since the remainder when you divide 5 by 2 is 1. A modular environment is basically, when the highest number that can be represented in that environment is a particular number — 1. For example, in an environment modulus 6, we can only use numbers 0,1,2,3,4,5, because these are the only numbers that be remainders in a division by 6. Therefore, when any operation is done in a

modular environment, you must end that operation by taking the modulus of that answer by the number. This is represented by putting a '*mod n*' at the end of every equation where n is the number we are considering. For example, 7 x 5 (mod 6) = 35 (mod 6) = 5 (mod 6). Therefore,

$$7 \, x \, 5 \; = \; 5 \; (mod \; 6)$$

This is obviously very important to computers since computers can only store a certain number of digits and will eventually not have enough space to store a large enough number and thus need to implement modular arithmetic to avoid the situation when the result of a computation is higher than the largest number it can store. Note: A clock is an environment that is modulo 12 and military time is essentially a modulo 24 environment. It takes some while to get used to modular arithmetic since you can get stuff like

$$0 \; = \; 8 \; = 64 \; = \; 800... \; (mod \; 8)$$

However, it will begin to make sense with practice and time. It helps to think of it as normal numbers but instead of them being on a line, they are on a circle, like with time!

Fermat's Little Theorem

This 'little' theorem, is very important to computer security and surprisingly serves as a basis for most modern cryptography! It can be expressed as

$$\forall a \in \mathbb{Z}, \; \forall p \in \{primes\} \; a^p \equiv a \; (mod \; p)$$

In non-weird mathy terms, this means that if p is a prime number, then for any integer a, a^p - p will be an integer multiple of p. It might be useful to take a minute to convince yourself that this is what the equation means. We further restrain that a and p must be coprime (a is not divisible by p). Now, the theorem shows that

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

Or that $a^{p-1} - 1$ is divisible by p. This concept is critical to RSA encryption!

Cyber Attacks

Crack Me If You Can

“A Cyber Attacks if any malicious code to alter a computer’s code, logic or data”. In general, we will be covering any attempts to circumvent security of a system to get access to protected data by deciphering encrypted messages, getting access to passwords or stealing someone else’s identity.

Password Cracking

So let’s begin our conversation with something you have likely been trying to do ever since you have been on the internet, cracking your friend’s facebook password (don’t try this at home please). Password cracking may be done by the user themselves, to recover a forgotten password, but is more widely used by hackers to gain access to someone’s account. There are plenty of online tools but let’s discuss some techniques you can try yourself.

Password cracking is mostly done by trying to guess encrypted passwords that have been leaked online. Most big companies have a database of passwords that are associated with their users to authenticate the user when they try to log in. However, these passwords are not stored in plaintext, they are encrypted by using various techniques. If these encrypted passwords are leaked online, there are ways to figure out the original passwords from the hashes.

Brute Force and Dictionary Attacks

These are some of the most common password cracking techniques given a list of hashed passwords. Let’s start with brute force. You are probably familiar with this technique. Essentially, we try all possible passwords of a given length, hash them and test them against all leaked passwords. If there is a match, then the cracker knows that the hash belongs to that password. So, for example, if we want to find all 6-length passwords in the database, we hash all passwords from *aaaaaa* to *zzzzzz*. To make it more inclusive, the hacker can try adding upper case letters, symbols and numbers. This technique works for small passwords but as soon as you get to passwords of length 9 and above, it becomes pretty hard to do those in a realistic amount of time even with a lot of computing power.

This is where dictionary attacks come in. Our issue with brute force attacks is that the search space too large for us to be able to check all of them. Therefore, we do an approach that may also be familiar with you. Have you ever tried to hack your friend? It's likely that you didn't go through all possible passwords of a particular length as that would take forever. However, you know your friend and therefore, you have an idea of what kind of password they would use and you keep trying things important to them like friends and family names.

The dictionary attack is a similar approach. Since the attacker doesn't usually know the victim, they have to go on more general knowledge about people's password choices. For example, the password *"password"* and variations on it are very common. Hackers also use passwords that were leaked online earlier and other such things. This is where the database comes in. The attackers have a bunch of real and commonly used passwords that they try the hashes against.

Now, this would be very useful, except some people try to change things up by making some small changes. For example, switching an E with a 3 or a ! with a 1 to name a few. Therefore, hackers put these "rules" in and apply every rule to the passwords in the database to increase their chances of getting the write hash by a lot. There are several different databases and rule sets available online put there by different hackers that work very efficiently and can retrieve up to 70% of passwords at times.

There are many other algorithms that can help increase the speed of finding the hashes of particular passwords and some that even allow one to remove the need to figure out the plaintext password altogether but these usually require some issues in the implementation. I have found the website searchsecurity to have well written and easy to understand articles on all of these algorithms. There are also many online tools like *hashcat* and *Jack the Ripper* with excellent online documentations and videos on how to use them. This is a great video that shows an example of using hashcat and even some stuff we talked about if you're more of an audio-visual learner.

Breaking Ciphers

This section will be a discussion about how to break some of the most popular ciphers throughout history. We will go through some basic techniques to do so. The talk on encryption has a description of the ciphers we are discussing here.

Breaking the Caesar Cipher

Breaking a caesar cipher is usually a very easy task and can be brute forced. It is easy to apply all shifts (size 1-25 in English) and see which sentence after the shift makes sense.

However, there is a similar approach to the caesar cipher where you apply a different shift to each letter and the brute force is not as easy an approach in this case, especially if the shifting key gets very long. However, *frequency analysis* works particularly well in this case.

Frequency Analysis

In cases where a particular letter become another particular letters in ciphers, frequency analysis is a very useful technique since it relies on the fundamental semantic relation of a character within a language. For example, the letter '*e*' and "*th*" are very common in the English language. Linguists have created a average frequency of characters in English by using the appearance of the characters in known english texts.

These frequency relations must remain through the encryption. For example, if *f* appears the most in our enciphered text, it is highly likely that *e* was enciphered as *f*. We make such deductions and then try to test it by switching the particular characters in and seeing if we get meaningful words.

You may realize that this is surprisingly useful as it is independent of place and time given that the cracker has access to enough texts from that time or place. We can create the frequency distributions on any text especially for Affine Ciphers. Here is a good example on how to break an Affine Cipher.

Similar techniques can also be used to break ciphers where a character may or may not become the same character, like the Vigenere Cipher.

Vigenère Cipher

This is generally not an easy task to accomplish but there are several clever techniques that can break this cipher. I won't go into an in depth discussion but we can discuss the basics. You might notice that if we know the length of the key used to encrypt the data, it would be a lot easier to guess the key and understand the message. Two

popular ways are *The Kasiski Test* and *The Friedman Test*. *The Kasiski Test* depends on the fact that a long enough test is likely to have two repeating phrases in the encrypted message. Assuming that this means that those phrases refer to the same unencrypted phrase, the gap between tells us something about the size of the key. For example, if the gap is of length 16, we know that the key must be of size 1,2,4,8,16 as these would lead the key to be repeated exactly at the start of that phrase. *The Friedman Test* is a little more complicated but we end up with a few guesses for the key size.

Now, we have no option but to test each key size. Given a key size, we break up the text into columns with each column representing the characters that were encrypted by the same letter in the key. Now, we just have multiple caesar shifts and we use frequency analysis and brute force as mentioned earlier to figure out the exact key size and the value of the key.

Obviously this is making some assumptions but given a long enough text, these techniques do work. There are a lot of details that have been left out but this is a basic understanding of the different techniques used to break a Vigenere Cipher.

Problems with Key Exchange

In this section, we will discuss ways to attack the algorithms widely used over the internet for exchanging keys and establishing a secure connection. We have explained all these algorithms in an earlier guide on Cryptography.

Diffie-Hellman Key Exchange

I will assume that you understand the basics of this type of key exchange and are comfortable with how the algorithm works. If not, I would recommend reading the earlier section on how this is done.

The Diffie-Hellman Key Exchange is a great way to set up a connection with keys that are guaranteed to be secret. This checks the Encryption aspect of our key exchange algorithm. However, the biggest weakness of this algorithm is that it is impossible to verify the authenticity of the other person using it on its own. This kind of an attack is called a *man in the middle attack.*

Man In The Middle Attack

This is one of the oldest weaknesses of the Diffie-Hellman Key Exchange. Let's have our three users, Alice, Bob and Carl again. Alice and Bob are trying to establish a secure connection and are using the Diffie-Hellman Key Exchange to create their keys. However, Carl is trying to read their conversations without either of them knowing anything weird is happening.

Carl is stuck because he knows that the Diffie-Hellman algorithm, allows two users to create keys without anyone else finding out what the keys are. However, he realizes the flaw that he can put himself in the middle of this conversation. Once Alice wants to talk to Bob, she will start the algorithm, however, Carl intercepts her first message and tells Alice that he is Bob. There is no way of her telling the difference over the internet as all that is exchanged is numbers. So he establishes a "secure" connection with Alice and she thinks she has a secure connection with Bob. Now, Carl tells Bob that he is Alice and creates a secure connection with him. Now, there are two secure connections: Alice and Carl, Carl and Bob that work as one connection. Alice sends a message to Carl, he decrypts it with the Alice-Carl key, he does whatever he wants with that message and then encrypts it with the Bob-Carl key and sends it to Bob and so on. Therefore, the Alice-Bob conversation is now happening through Carl and neither of them are the wiser. This is a classic Man in the middle attack and happened very recently in 2015 (search superfish man in the middle attack).

There are more issues with the Diffie-Hellman approach but they are based on mistakes that the programmer who writes the algorithm can make. For example, if the value P, which is the large prime number that is the modular environment, is too small, then the possible key values in public might be easy to brute-force. The attacker can easily try all values of K^a and K^b. Since the have the K^{ab} they can realistically figure out the private keys and then the connection is insecure. Therefore, P needs to be large enough so that brute-force attacks are unfeasible.

RSA Cryptosystems

In general, it is hard to break the RSA algorithms. It is more important to discuss what mistakes can be made by a programmer in their implementation of an RSA algorithm that makes it easier to hack into. *Note: I will be assuming that you have an understanding of the RSA cryptosystem and will be using the same nomenclature as used in the previous section on encryption.*

Using Small Prime Numbers P and Q:

The most basic mistake that can be made is to use small prime numbers for P and Q. These numbers cannot be too large or too small. First, we must understand that *P*Q* is the modular environment *n*. This means that *n* defines what our permitted range of numbers will be (*0 - n-1*). Therefore, if *n* is too large, our algorithm would be too inefficient, which means starting up our process would be too slow. If the value is too small, then our algorithm would be open to a brute force attack. Since, the strength of and public key cryptography algorithm lies in the fact that the private keys cannot be reverse-engineered from the public components, if our modular environment is too small, the private keys can be extracted from the public components and we lose the *Encryption* aspect of our algorithm. There is a good range of values for *n* and this lies around 2000-4000 bits long.

Wiener's Attack:

Michael J. Wiener proved that if the value chosen for *d* is too small, an attacker can efficiently find the value of *d* which is the essential part if the private key. With this information, and the available public key *e* and attacker can easily encrypt the message. However, if *e* is large enough, it doesn't matter how small *d* is and Wiener's attack cannot be applied. *Note I glossed over a bunch of details but here is an in-depth look into the math behind this attack.*

This is a fairly detailed explanation of some more attacks. *Warning*: this article involves a lot of math and I wouldn't recommend reading it for anything more than recreational purposes unless you understand the RSA Cryptosystems well.

Multi-Prime RSA:

This refers to the implementation of the RSA cryptosystem where n is the product of more than 2 primes. This algorithm is useful because it is faster and more space-efficient in the decryption phase (thanks to the Chinese Remainder Theorem). We can break up the huge modulus step into smaller steps using the Chinese Remainder Theorem and this provides a huge boost in speed. However, this method hasn't been studied enough to prove it is just as secure. It is recognized that if 3-4 primes are used, a significant boost can be noticed in speed.

However, if more factors are used, the size of individual factors would become too small and it would become a lot easier to use techniques to factor out the n and thus break the algorithm.

Chinese Remainder Theorem

Chinese Remainder Theorem

Given pairwise coprime positive integers $n_1, n_2, \ldots, n_k$ and arbitrary integers $a_1, a_2, \ldots, a_k$, the system of simultaneous congruences

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution, and the solution is unique modulo $N = n_1 n_2 \cdots n_k$.

Source: https://brilliant.org/wiki/chinese-remainder-theorem/

Above is a very important theorem relating to RSA cryptosystem and its weaknesses. So we will briefly discuss it to aid the section on breaking the RSA Cryptosystem. Each mathematical expression means that if you divide x by n_k, the remainder is a_k. The theorem means that in any linear system as above, there is at least a single value for x that fulfills the requirements of the system modulo N.

1. Compute $N = n_1 \times n_2 \times \cdots \times n_k$.
2. For each $i = 1, 2, \ldots, k$, compute
$$y_i = \frac{N}{n_i} = n_1 n_2 \cdots n_{i-1} n_{i+1} \cdots n_k.$$
3. For each $i = 1, 2, \ldots, k$, compute $z_i \equiv y_i^{-1} \bmod n_i$ using Euclid's extended algorithm (z_i exists since $n_1, n_2, \ldots, n_k$ are pairwise coprime).
4. The integer $x = \sum_{i=1}^{k} a_i y_i z_i$ is a solution to the system of congruences, and $x \bmod N$ is the unique solution modulo N.

Source: https://brilliant.org/wiki/chinese-remainder-theorem/

Above is the way to compute the value of x. A proof of the theorem can be found here.