Design of Algorithms

CSC 304

Group Project

**Algorithms Used in Cryptography (Encryption & Hashing)**

The importance of cyber security has been growing every year since information began to be stored on computers. With today's more prevalent access to computers and the internet, being able to store and use information safely has become a necessity. The methods to protecting this information have grown extremely advanced but continue to fail to the world's most talented hackers. Going back to the basics of cyber security is cryptography. To hide these messages, you must encrypt them, meaning information is only given to authorized people. For those people to read those messages they must decrypt them. There are many ways in which to encrypt and decrypt messages between parties and these strategies can include many algorithms discussed in our Design of Algorithms class.

During our class, there have been numerous algorithmic techniques with a plethora of applications. A main focus in the class has been basic data structures and sorting and searching methods. Algorithms are a crucial part of the encryption process. A sorting algorithm comes into play when you look at the fundamentals of encryption. The theory behind encrypted data is that the algorithm does not have to be kept a secret as long as the key is. Decryption without the key involves a search

algorithm which looks for repeating patterns in the data. When that pattern is found, the program can then use the most repeated sequence of words to create a cipher.

There are two main types of ciphers used in cryptography, public-key and secret-key. A public-key cipher is asymmetric meaning that there are two keys to encrypt and decrypt data. There is one public-key that is shared with everyone and then a single private key that is known by only the single intended user. A secret-key cipher is also known as a symmetric cipher because both sides of the message have the same cipher key. The first secret-key cipher algorithm to be approved by the U.S. government for public disclosure was the Data Encryption Standard (DES), which was created in 1975.

There are two major categories for secret-key algorithms, stream and block ciphering. A stream cipher will encrypt the data as a sequence of bits and encrypt each bit one by one. This stream cipher can be manipulated because if a hacker knows the format of the code then they do not have to know the cipher, they can just alter the message by switching individual bits on and off within the cipher-text.

Block ciphering is more commonly used than stream ciphering in the modern era. Block ciphers became popular when DES was introduced. Like the name might suggest, instead of processing data bit by bit it will encrypt a block of data all at one time. This can be seen when looking further into DES. DES will accept 64-bit blocks of data and will use a 56-bit secret key to produce 64-bit blocks of cipher-text. The block cipher will use a single bit of plaintext across a whole encrypted block. This type of cipher means that if even a single bit is changed the entire

message after that bit will be rearranged. So, this block cipher eliminates the issue of a hacker being able to change specific parts of a message without the cipher key. The downfall to a block cipher is that because it is coding large blocks of a specific size, if the message has two repeated blocks then the cipher will create two of the same coded blocks. This repetition in coding can be used by a hacker to identify some patterns and start making attempts to guess what the message is. Obviously, this ciphering needs another defense system to help prevent against this problem.

Modes are used in many encryption processes to combine the block cipher with the plaintext. There are four main modes created for a standardized use in DES--electronic codebook, cipher block chaining, cipher feedback and output feedback. These modes were created so that they could combine data between encryption steps which would mean that each one achieves slightly different results. Cipher block chaining is the most common mode and it uses the previous block of cipher-text with the next block of plaintext before encrypting the data. An initialization vector is applied to the first block before it is encrypted which means that it will encrypt each block given the encryption of the previous blocks. This mode helps to detect changes that were made to the cipher-text because a change in the order of the cipher-text block will create gibberish when decrypted.

Going back to DES and block ciphers, the most basic method of attack is called brute force. This is what is known as trial and error method, each key is used until the right one is found. “The length of the key determines the number of

possible keys -- and hence the feasibility -- of this type of attack" (Search Security). . DES uses a key that is 64-bits long and therefore it would take 2^56 tries to test for the right key. When DES was first introduced this seemed like a secure encryption algorithm but with today's technology, knowledge and experience DES is thought to be insufficient.

In 2001, a better successor to DES launched, called the Advanced Encryption Standard (AES). This algorithm was then replaced by the Triple DES (3DES) in 2005, and has been approved to stay in use (at least by government standards) until 2030. As the name would suggest, 3DES works by applying the DES algorithm 3 times to each block of code. 3DES uses 3 different keying options, with the first of those having a key length of 168 bits (three 56-bit keys). The key is made to be only 112-bits to combat attacks on ciphers using the 56-bit and 168-bit lengths. Keying option two will reduce the effective key size down to 112-bits. However, it is more susceptible to attacks since the hacker can obtain cipher-texts from arbitrary plaintext, giving 3DES only 80 bits of security. One drawback of using the 3DES algorithm is that the key length for 3DES is extensive, thus causing the encryption to be slow.

One of the biggest public-key ciphers used today is the Deiffie-Hellman (DH) algorithm, which essentially launched public-key cryptography. The DH algorithms is "Like all public-key algorithms… based on the difficult mathematical problem it presents to would-be crackers: discrete logarithms in a finite field" (Security Search). DH is based on the concept that it is easy to perform modular

exponentiation but it is hard to undo and find the discrete logarithm of the result. DH is less of a cipher and more of a way to turn a public-key into information that only authorized people have access to. Elliptic curve cryptography (ECC) is similar to DH because it is based on discrete logarithms. ECC uses computations mapped to a two-dimensional elliptic curve which results in the algorithm using much smaller keys. However, because of restricted processing environments, this smaller key size is negligible—So ECC isn't usually chosen over DH.

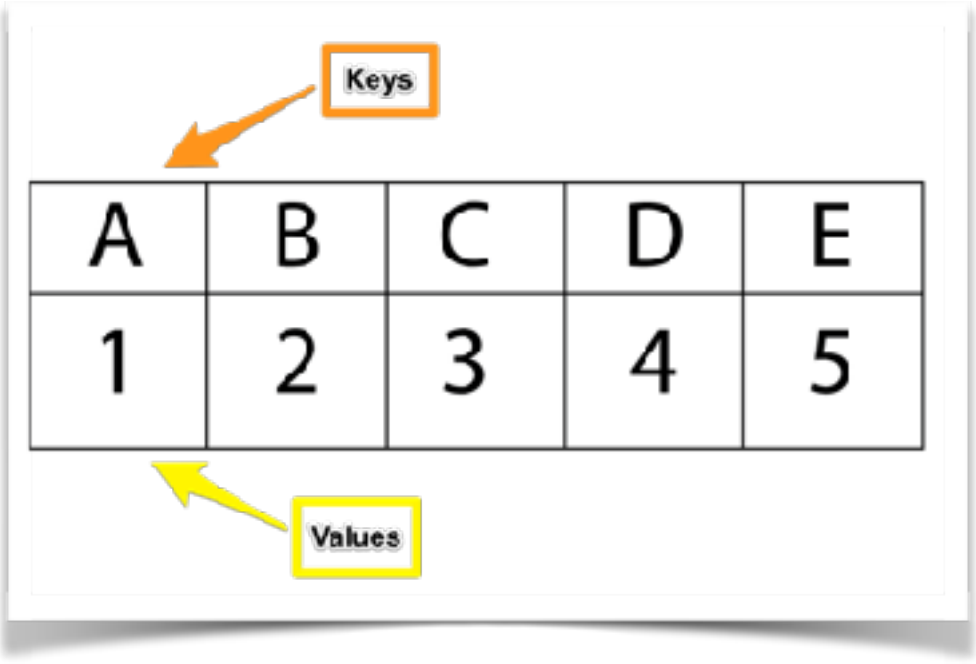
Another widely used cipher algorithm is RSA, which "performs key distribution by using a public key to encrypt a randomly generated secret key" (Security Search). The recipient of this key uses the corresponding private key to decrypt it. RSA performs digital signatures by encrypting a hash using the recipient's private key. The mathematical operation used in RSA is factoring because it is simple to multiply two prime numbers, but far more difficult to extract these two prime numbers if both were unknown in the first place. Most e-commerce sites use RSA to secure the communication between the server and the browser.

A public-key cipher is not usually chosen over a secret-key cipher for encrypting large amounts of data due to secret-key ciphers encrypting data at a much faster rate than public-key ciphers, which use many mathematical operations to encrypt their data. Therefore, hackers can use this mathematical approach to uncover messages. The key sizes for public-keys must be larger than secret-keys to accommodate this weakness, while the strength of the key would remain the

same. The public-key cipher would be much slower than the secret-key cipher because it has a large key size.

Encryption can be an effective way to store information while keeping it hidden from the public eye. Block ciphers can quickly store data by encrypting blocks of data at a time, and create complex keys that are nearly impossible to decode without the desired secret-key cipher. Lastly, the DH and RSA algorithms are both relatively effective means of encrypting data such that only the desired users can access it via a secret-key. Besides encrypting data, however, there are other ways to store data by using key values. One such method is called hashing.

Hashing, according to the “Problem Solving with Data Structures and Algorithms” textbook is “creating a collection of items which are stored in such a way as to make it easy to find them later…and producing hash values for accessing data or for security”. This collection of items is usually called a “hash table”.



Hashing is very similar to encryption in that it uses a some kind of arbitrary mathematical function(s) to convert a string of text into numerical values, so that they can be stored securely and also be accessed very quickly. This not only makes it much faster to search for and add things than say a binary tree—which has an average complexity of **Θ(log(n))** for search and **Θ(log(n))** for insertion, or some kind of overflow table, whereas the hash table has a constant time access of **Θ(1)**—but it also makes the data much more secure because the hash function is completely deterministic of the name(key) it is given and the value assigned to it.

But, before it is possible to input the keys and values into a hash table, we must first create the hash values from the data we want to hash out. The most common way of doing this is taking the data you would like to insert, for this case it would be integers, into the hash table and then dividing it by the size of the table “n”, the equation would look something like this, (h(item)=item%n). This function takes the incoming data and then divides each item by the size of the table you want to create and then returns them to the program.

There are many different methods of hashing that can be used, but two of the most popular are the MD5 method and the SHA method. MD5 is a hashing method that first divides the input in blocks of 512 bits each. 64 Bits are inserted at the end of the last block. These 64 bits are used to record the length of the original input. If the last block is less than 512 bits, some extra bits are 'padded' to the end. These extra bits of “padding” are added on to the end of each block so that they are equal

in length, and so that shorter blocks are harder to decode back to the original data before it was hashed out.

The other method of hashing is the SHA method. The SHA-1 method "produces a 160-bit (20 byte) message digest. Although SHA-1 is slower than MD5, this larger digest size makes it stronger against brute force attacks"(SHA Site). This makes it much harder to crack the mathematical equation that was used to encode the data. The SHA256 hashing method is hands down the most secure method of hashing information for many reasons but the sole reason that makes it so secure is that when you try to brute force the data to decode it, you can't just brute force the SHA compression function, you must brute force the SHA message schedule as well.

The SHA compression function is used to update the registers that hold the data in the table. These are used to operate on "32 bit words and produces a 32 bit word as the output"(SHA Site). This makes the 512 bit blocks much harder to crack and start to decode back to the original text. The SHA message schedule is used to determine what order the code blocks are placed in. So, the message schedule takes the encoded jumbled information that are put into blocks, and then jumbles the blocks out of order even more than they already were. This is what makes SHA-256 such a secure method of hashing, it is often used to store passwords on consumer machines.


*The above image is showing how you can import python's native hashlib package to hash your own information*

As mentioned above, hashing is often used to store and authenticate passwords, but why is it done this way? Well, hashing is such a valuable way to store passwords on a local machine because, if for some reason someone gains access to the database where your passwords are stored, hashing them before storing them makes it as difficult as it possibly can be to retrieve the passwords in plain text format so that they can be read and used.

One way that the hashed data can be made more secure is by creating complex mathematical algorithms to determine the hash values. This makes it more secure because when the algorithm to compute the hash values takes a long time to compute the values, it makes it exponentially more difficult for someone who is trying to brute force the data to plain text. First, they have to try to figure out the algorithm and then once they do, they have to run the algorithm which has been designed to run extremely slow, which can help with damage control when a database is compromised. As secure as these hashing methods are, they are not impervious to brute force attacks. The SHA-256 hashing method has been hacked

for testing and its **time complexity for the successful hack was 2^28.5.** So, they are not impervious to the brute force attacks, but they are in no way shape or form fast in any situation and this is what they are designed to do. Make it harder for people to crack the hashing method, not necessarily to make them uncrackable, because after all, no method of hiding data will prevent people from uncovering it.

In conclusion, the encryption option is a very convenient way to send and receive data securely, because the only thing that is required to decrypt the data that is received is the key, be it public or secret. On the other hand, hashing is a very good way to store sensitive data locally, that way potential hackers will not only have to break into your system, they must brute force all the hashed data, which hopefully takes long enough that you would see what is happening and have enough time to react and prevent any further damage. Both types of data storage largely depend on the type of data being stored, as well as the situation or conditions that are being addressed.