**University of Stavanger**

**Assignment 2**

**Security and Vulnerability of the Networks**

Abolfazl Taleb zadeh

**October 2021**

**Abstract**

The second mandatory assignment is mainly about practicing key exchange capability using Diffie Hellman algorithm. Utilizing key exchange algorithms play a vital role in the world of cryptography particularly under the circumstances in which using a symmetric algorithm is urged. Therefore, a unique single key is needed to be exchanges among concerned parties. As far as transferring the keys without following the confidentiality principles will put a big question mark on the primary purpose of keeping the secrecy of the message, a well-trusted method is needed for sharing the credentials among all the parties within a communication. For implementing the Diffie Hellman algorithm, a class structure is preferred. In the class the arguments have been defined as the properties and different stages of the instruction is defined as methods which can be called in the proper order by the user. In addition, a pseudo-random number generator algorithm (CSPRNG) is used to fortify the enumerated key, which in this specific case the choice is B.B.S (Blum Blum Shub) algorithm. It can be observed that using B.B.S the calculated key gets significantly more secure and also the intended size of the key for being usable by the user-defined encryption algorithm which would be SDES, is delivered thoroughly. The beneficiary and efficiency of application of both Diffie Hellman and B.B.S is well verified and experimented in the second part of the assignment which aims to establish a safe mutual communication tool using the assumptions implemented in the class.

**I. Introduction**

Using symmetric encryption algorithms has always been known for having the burden of sharing the keys between the two sides of the communication. Diffie-Hellman, also known as DH, was the first algorithm ever used for sharing the keys of a symmetric encryption method which was published in 1976 by Whitfield Diffie and Martin Hellman. DH makes it possible for two parties of a mutual communication which use a symmetric encryption method to share a single key without any need of special equipment or predefined secret information. The most important downfall of this method is that none of the sides of the communication cannot be sure about the authenticity of the public key that is obtained from the other side. In the other word there always is a scale of man in the middle attack risk [1]. A solution which is given for this problem is Public Key Infrastructure (PKI) which as stated in www.wikipedia.com, “have been proposed as a workaround for the problem of identity authentication. In their most usual implementation, each user applies to a “[certificate authority](https://en.wikipedia.org/wiki/Certificate_authority)” (CA), trusted by all parties, for a [digital certificate](https://en.wikipedia.org/wiki/Digital_certificate) which serves for other users as a non-tamperable authentication of identity”. (“Key Exchange” November 2014 section 2.3 public key infrastructure). There are also other methods like Password-Authentication Key Exchange (PAKE), Quantum Key Distribution (QKD) and etc. which we are not going to go any further into details. In this assignment since the generated password using DH is pretty simple, a pseudo-random number generator algorithm (PRNG) is being used to fortify the enumerate key using DH algorithm, which in this case B.B.S (Blum Blum Shub) algorithms is chosen. Pseudo-random number generators implement a value called seed to generate a random number which will be repeated in a specific range. There are multiple types of PRNGs which use different methods (e.g.RC4, NIST CTR\_DRBG, ANSI X9.17). in the first part of the assignment a secure key exchange instruction is practiced using DH with having the benefit of key extension capability using B.B.S algorithm. And in the second part this method has been implemented in real a mutual communication using two servers through different ports.

**II. Design and Implementation**

In this part the implementation method and the technique which is used in different part of each section will be discussed and analyzed in detail. there are eight steps in the first part of the program which will be argued in turns. The first part of the program is constructed in the form of a Python class because it will be used in other parts of the program. The class there are several public variables which would be known as the properties of the class after the instances are made from. These variables store the necessary values concerning different stages of the class. These variables include, a list which is called “primes” which holds a list of prime numbers that will be used for multiple purposes, a list called “z” which stored the values of a cyclic group of “p”, an integer variable called “f\_k\_dec” which keeps the decimal final key which is calculated by B.B.S algorithm, an integer variable named “p” which holds the prime value which is used for DH algorithm, integer variables “ya” and “yb” which store the public keys for the first and the second communication side also known as “alice” and “bob”, an integer variable called “k” which holds the value of the key enumerated by DH algorithm, integer variables “a” and “b” which store the private keys for both parties, an integer variable named “q\_times\_p” which holds the value of multiplication of two prime numbers which is used by B.B.S algorithm. The class also has an initiation method which receives a number “n” in the defining stage. This number assigns how big the chosen prime number and as the result the calculated key can be. In the initiative method of the class several methods also get executed which will be discussed individually. The very first method that is called is “prime\_gen()” which has the task of generating a set of prime number smaller that the number stored in “n”. within this method, a flag which is named “is\_prime” is used for specifying whether the generated number is a real prime number. There are to nested loops in this method, the outer one iterated from two, which is to be the first prime number, to the number “n”. and the inner loop runs with value of two which is the smallest value for the outer loop, to the current value of the outer loop which is stored in variable “i”. This is to check every single number smaller than the current value of the outer loop as a factor, and test if its correspondent mod is zero and as the result the number is a prime number or not. So, it checks if the mod of the outer loop current value (i) has the remainder of zero on the division by the value of the inner loop (j). If so, it changes the value of the flag “is\_prime” and break the inner loop and so on. But if no factor was found for regarding a value for variable “i”, the “is\_prime” flag remains “True” and going through the second conditional phrase the value “i” is appended to the primes list which we mentioned before. This way, all the prime numbers smaller that “n” value is calculated and stored. The next method which is called in initiative of the class is “q\_assigner” which has the task of choosing one of the prime numbers in the “prime” list. In this method, a random number is chosen in the range of the length of the list and is used as the index of an element of the prime list. This element is chosen and assigned in “p” variable of the class. The next and of course one the most important methods of the program will be “g\_calculator()”. This method is responsible for producing the list of generators of the cyclic group and selecting a single generator from the list randomly. At the beginning of the method a local list named “generators” is defined (the reason the list of generators is not stored as a property of the class is that, as far as a single generator is randomly chosen from the generated list and stored in the public variable “g” and also the list of generators does not include in the program mandatory requested output, it is not necessary to keep the list in the class). After defining the generator list, the cyclic group is produced and kept in the public list “z” through a simple loop which runs for the carriable “i” from one to p-1, in which p is the prime number selected previously. As far as a cyclic group of is a set of numbers from one to p-1, the numbers just simply get appended to the “z” list. Then considering the definition of a generator of a cyclic group which is stated in [www.wikipedia.com](http://www.wikipedia.com), “When  is cyclic, its generators are called primitive roots modulo n” (cyclic groups, section 2.2, p.1 l.4). A loop tests every member of “primes” list if they are a generator of . It is done by calculating the result of X to the power of Y mod p, which X would be a member of “primes” list (every prime number smaller than p) and Y is every member of the cyclic group . The result then is stored in a list, gets converted to a set to remove repeated results and gets compared to the , if the set was identical to the cyclic group, then “g” is known as a generator appends to “generator” list and process continues for the next number in “primes” list. Eventually after all the generators found, in a process similar to the method “q\_assigner()” one of the generators get selected and assigned to public “g” variable and generator is returned for the purpose of printing to the output. The example below shows how the generators of a cyclic group is calculated:

and for every

which is not equal to

which is equal to so 3 is a generator for cyclic group

which is equal to so 5 is a generator for cyclic group

The next method in the initiative of the class is “bbs\_random\_selector()” which chooses a prime number suitable for B.B.S algorithm. Before going through the method in detail, let’s take a closer look on how B.B.S method exactly works. As was told earlier, B.B.S is a PRNG (pseudo-random number generator) algorithm. The reason behind using this algorithm is to generate a bigger and more reliable pseudo-random number based on a smaller random number (k) which is calculated using DH algorithm. The instruction of generating the more complex and secure random number using B.B.S is to use choose two prime numbers in a defined range also having the remainder of three when divided by four. For this purpose, a similar method to “primes\_gen()” is used which also checks the required condition for B.B.S prime numbers. Two numbers “p” and “q” are selected and their multiplication result is stored in the public variable “q\_times\_p”. The class also includes some other methods which is discussed in details now. The first method to talk about is “secret\_number\_generator()” which assigns two random number to variables “a” and “b” as the private key of two individuals (let’s say Bob and Alice). And then calculates their corresponding public keys using the formula . And assigns it to the public class variables “ya” and “yb”. The next method “is k\_generator()” which does the task of generating a key for the DH algorithm. The formulation for calculating the key in DH algorithm is . The value for the key is calculated and assigned to the class public variable “k” as the key. There is still one more method which does the task of generating the final extended key using B.B.S. the name of that method is “bbs()”. Before discussing the stages in the method let’s see how B.B.S algorithm creates an extended key using two prime numbers smaller than “p” and a seed value which is equal to the key calculated by the DH algorithm. The process is done by updating the value of seed by raising it to the power of tow and calculating its remainder of the division to “p” times “q”. and then finding the remainder of the calculated seed from dividing it to the number two which either will be 1 or 0. Then that will be assigned to the index “i” of a binary number which “i” stands for the number of the rounds which this process is repeated. So, this process goes on until the number of bits gets equal to the interested amount (which in this case is ten, as far as the required key in SDES algorithm that is going to be used in our program is 10 bits). The example below clarifies the procedure in a better way:

For the values and we have and also, we have then:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| X | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| i | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |

Binary key = 0000010010

Decimal key = 18

**III. The Results**

**IV. Discussion**

**V. Conclusion**

**VI. Works Cited**

[1] https://en.wikipedia.org/wiki/Key\_exchange

[2] https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman\_key\_exchange

[3] https://www.youtube.com/watch?v=Yjrfm\_oRO0w

[4] https://www.youtube.com/watch?v=hp7bpkNL790

[5] https://dev.to/techparida/how-to-deploy-a-flask-app-on-heroku-heb

[6] https://www.youtube.com/watch?v=M-0qt6tdHzk&t=7s

[7] https://www.youtube.com/watch?v=9MHYHgh4jYc

[8] Cryptography and Network Security: Principles and Practice, Stallings, W.

[9] http://cs.wellesley.edu/~webdb/lectures/flask/flask.html

[10] https://www.youtube.com/watch?v=Yjrfm\_oRO0w&t=317s

[11] CS255 Stanford course textbook “Cryptography and Computer Security. Very basic number theory fact sheet”

[12]