Chapter 12

Security Principles

Description

Computer and network security is based on the principles of secrecy, data integrity and access authentication. Each one of these principles is implemented by methods and protocols. Secrecy is implemented using two methods of ciphering: symmetric and asymmetric. Data integrity is provided by fingerprints of the data created by hash algorithms. Secure authentication is critical to gain access to resources deployed in the clouds. This section focuses on the topic of security principals and its methods and protocols and how all of theses apply to the security of the cloud.

Learning Outcomes

- Explain the fundamental principles of computer and network security.
- Analyze the methods and protocols of secure access to cloud virtual machines.
- Comprehend the system of authentication to access cloud resources.
- Describe the use of security certificates to validate webservers legitimacy.

Main concepts

- Computer and network security fundamentals.
- Security principals.
- Asymmetric key ciphering.
- Symmetric key ciphering.
- Authentication.
- Secure Shell Protocol.
- Security Certificates.
- Secure HTTP protocol HTTPS.

Learning Activities

- Authentication Access to EC2 instances.
- The role of the private key file in the authentication process.
- The functioning of the Secure Shell (SSH) protocol.
- Configuration of Public Key Infrastructure with Certificate Authority.

Security Principles

Computer and network security are based on a set of concepts, rules and methods with the overall goal of protecting data whether it is stationed on a server or travelling across networks. In general terms, robust security must provide the following:

- **Confidentiality** which is implemented with **ciphering** methods.
- **Integrity** which is implemented with **checksums** or **hashes**.
- **Authenticity** which is implemented with **digital signatures**.
- **Authorization** which is implemented with identity **authentication** methods.

Confidentiality

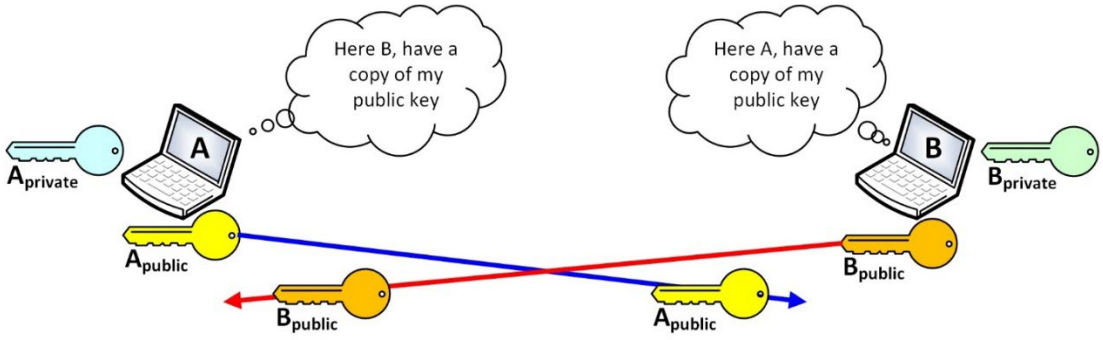
Data confidentiality means secrecy of the data. This is obtained by ciphering the clear text data using methods and algorithms. There are two major ways to cipher data: **asymmetric** key algorithms and **symmetric** key algorithms.

Asymmetric key ciphering

Ciphering with asymmetric keys is the method by which each end-node participating in a communication process owns a pair of keys for the session. One of the keys of the pair is used to cipher clear text data while the other key is used to decipher the ciphered data.

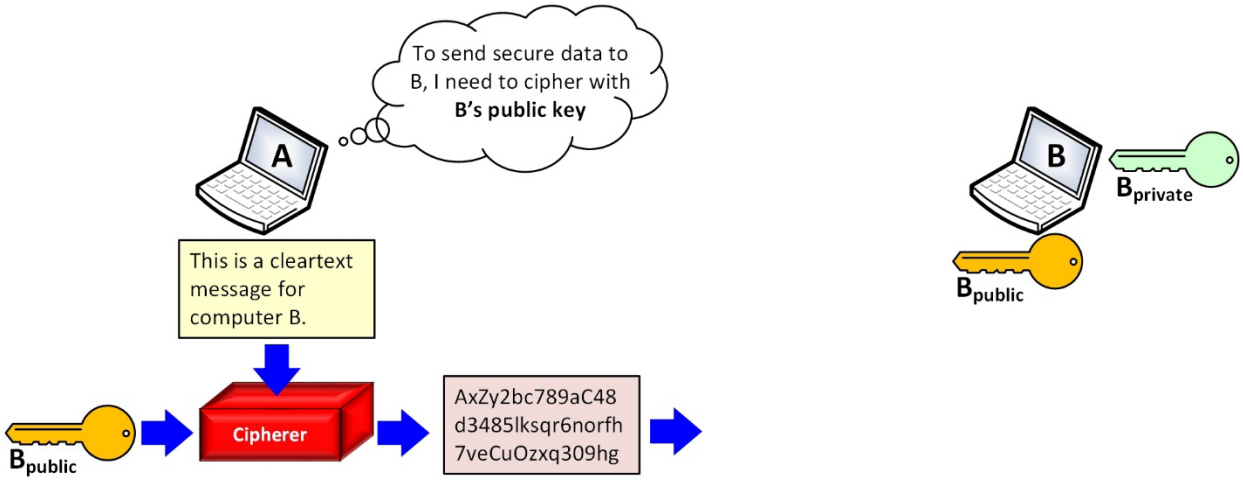
Each node creates its own pair of keys, one is public, the other remains private

One of the keys of the pair is provided to the other node hence such key is **public**. The other key of the pair never leave its owner; thus, it is a **private** key. The computers exchange their public keys only.



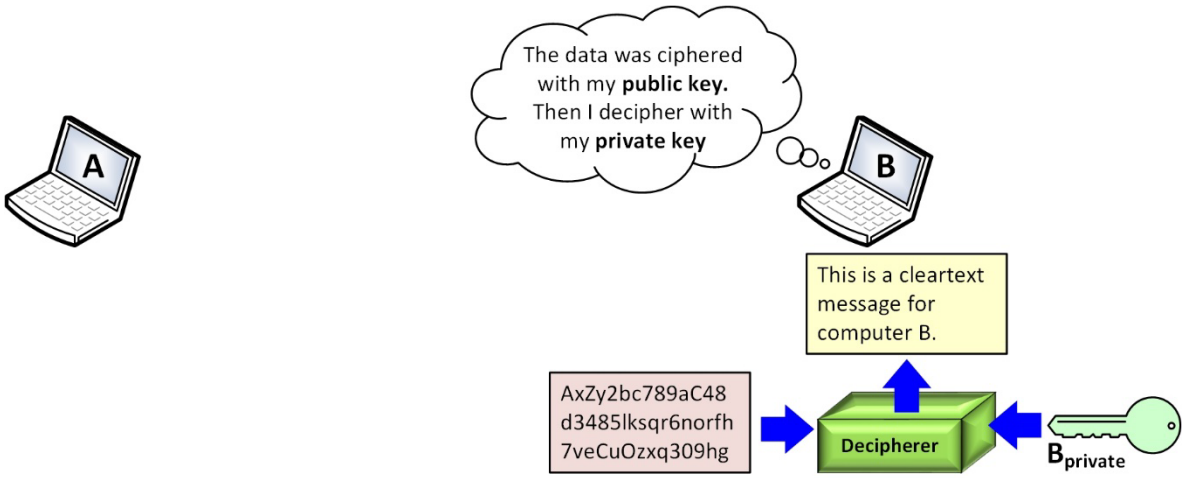
The nodes exchange their public keys

The data ciphered with one key of the pair, can only be deciphered with the other key of the pair. In the following example, computer A sends a ciphered message to computer B. To do so, A ciphers the message with the public key that B already advertised. The only computer that could decipher such cipher-message is computer B.



Computer A sending secure data to computer B ciphered with the public key of B

The clear text data and the public key are supplied to the inputs of a cipherer system (for example Advanced Encryption Standard AES). The output of the cipher block is the cipher-data that is delivered to computer B. When this node receives the data from A, it uses a reversing mechanism.



Computer B deciphers the secure data with its private key

Computers do not have only one key pair, but rather they may have many key pairs for different processes and protocols. The keys are generated according to a complex mathematic procedure, the most common of which is Rivest, Shamir and Adleman (RSA) after the three computer scientists [1] who invented the algorithm in 1977.

The RSA system is based on the principle that for any public key (e) there is a corresponding private key (d) in a modulo number space n. However, knowing one the keys do not offer any clue of the value of the other. In other words, one key can not be guessed by having the other key. That is why, one key is made public while the other is kept private.

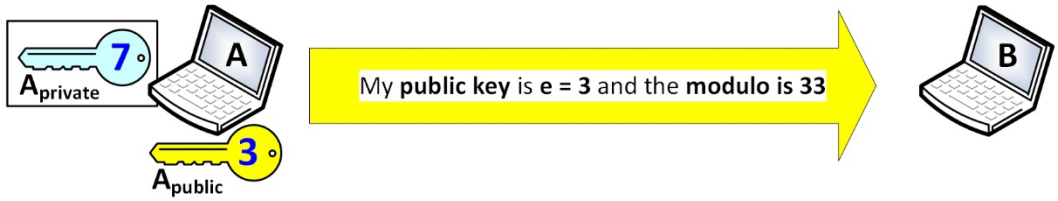
Another fundamental principle of public-private key security is that information ciphered with either one of the keys can only be deciphered with the other key of the pair. The RSA system has two (2) operations to implement **data secrecy**:

1) Cipher with $c = m^e \ mod(n)$
2) Decipher with $m = c^d \ mod \ (n)$

Where:

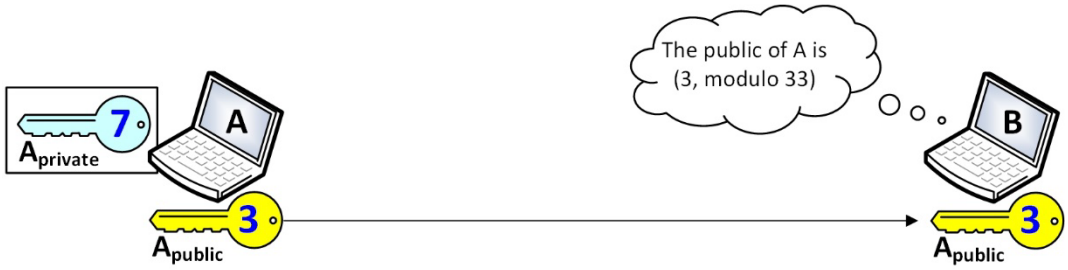
- The letter **c** stands for **cipher-text**.
- The letter **m** is **cleartext message**.
- The letter **e** is for the **public key**.
- The letter **d** is for the **private key**.
- **mod** is math **modulo operation**. (as in addition, subtraction, multiplication, division, modulo).
- The letter **n** is the **value** where the modulo will be calculated.

This is better explained with an example with two small value keys. Computer A has calculated its key pair and found a private key d = 7 and a public key e = 3. The modulo value is 33.



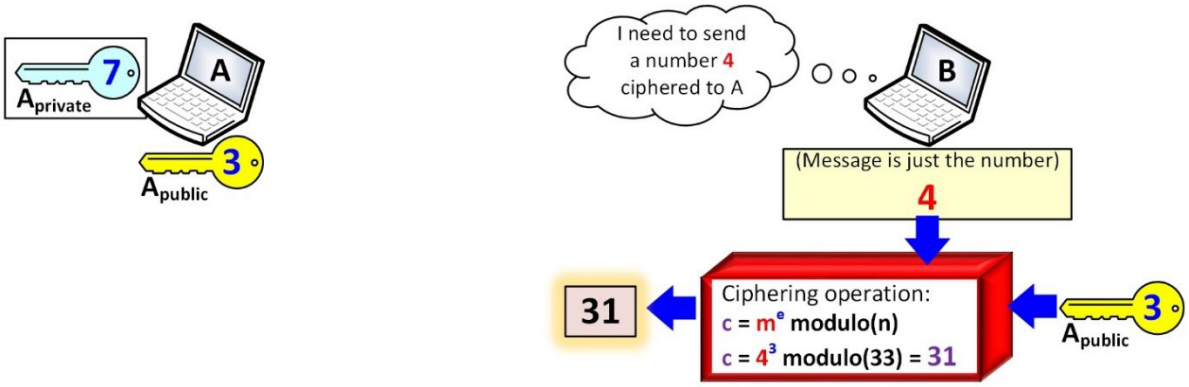
Computer A shares its public key (3,33) to computer B

Computer A sends, literally, its public key to B. This can be done for a one-time session or for a long living session. Computer B stores the public key of PC-A. This key will be used for all the secret conversations in the direction from B to A. Conversely, computer B must share its public key with A, so the conversation from the other direction is also ciphered. However, for the sake of simplicity, this explanation covers only the messages going from B to A.



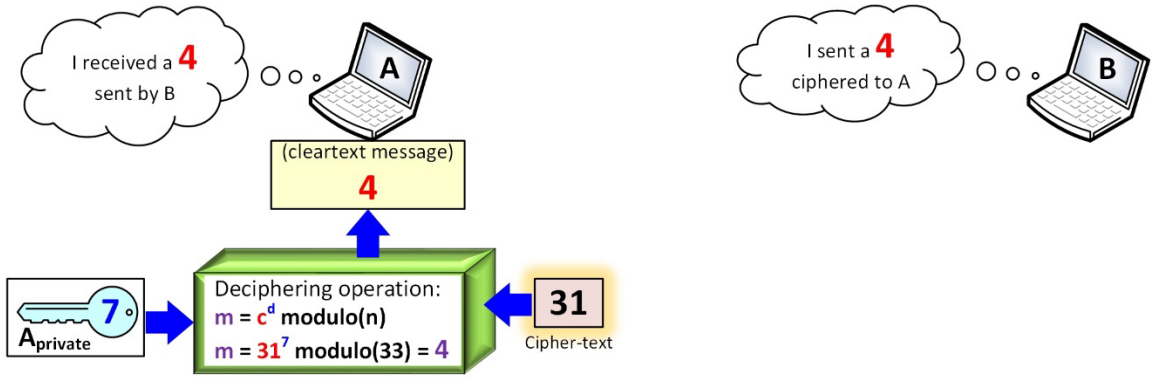
Computer B knows A's public key (3,33)

Let's assume that PC-B needs to send a message to A and the message is just the number 4. Computer B delivers the 4 to its cipher system. The public key (3) of A is also supplied to the cipher which performs the math operation $c = m^e \ mod(n)$ yielding $c = 4^3 \ mod(33) = 64 \ mod(33) = 31$.



Computer B ciphers the message value 4 and it obtains a cipher-text valued 31

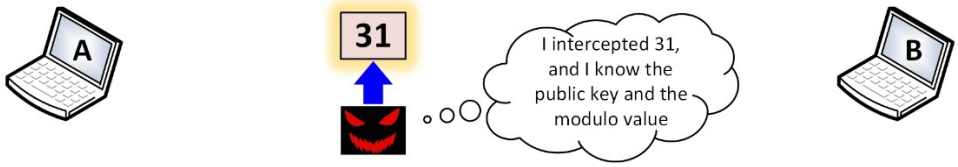
Computer B sends the ciphered message 31 to computer A which performs the reverse operation as illustrated in the figure below.



Computer A deciphers 31 and obtains the message 4

Since B ciphered with the public key of A, only A can decipher that because A is the only node in possession of the private key. Computer A performs the math operation $m = c^d \ mod(n)$ yielding $m = 31^7 \ mod(33) = 27{,}512{,}614{,}111 \ mod(33) = 4$ which is the original clear text message from B.

A valid question at this point is how solid this secrecy system of public and private keys might be. Would it not be easy to find the private key by having the public key and the modulo value? Let's try an exercise to demonstrate this point. A malicious third party has somehow got a hold of the ciphered message, the public key and the modulo.



A malicious party attempting to break the public-private key system

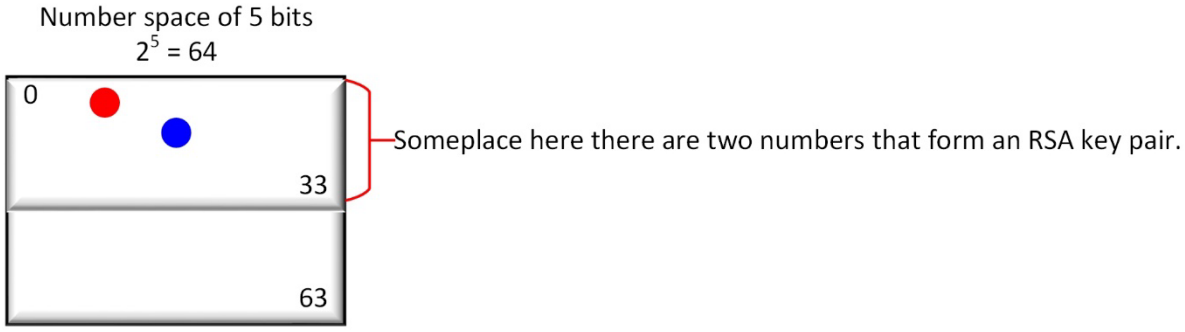
The RSA algorithm is public and well known. The malicious party has all the components of the system except the private key. One of the principles of the system is that the two keys reside in the number space of the modulo. Thus, somewhere between 1 and 33, the other key of the pair exists. Knowing this, the malicious party can start guessing a private key, number by number. For example:

Attempted (private key) value	Replace values in $c = m^e \ mod(n)$	Result
1	$c = 31^1 \ mod(33)$	31
2	$c = 31^2 \ mod(33)$	4
3	$c = 31^3 \ mod(33)$	25
4	$c = 31^4 \ mod(33)$	16
5	$c = 31^5 \ mod(33)$	1
6	$c = 31^6 \ mod(33)$	31
7	$c = 31^7 \ mod(33)$	4
8	$c = 31^8 \ mod(33)$	25
9	$c = 31^9 \ mod(33)$	16
10	$c = 31^{10} \ mod(33)$	1
11	$c = 31^{11} \ mod(33)$	31
12	$c = 31^{12} \ mod(33)$	4
13	$c = 31^{13} \ mod(33)$	25
14	$c = 31^{14} \ mod(33)$	16
Keep going until 33	$c = 31^{33} \ mod(33)$	25

Table of possible values of a deciphering operation with different private keys

The result seems inconclusive, even confusing and that is precisely a security concept, **confusion**. Cipher data should appear confusing to uninvited observers. The number 4 appears several times, but only one is correct, the one obtained with the private key valued 7. For the hacker to draw a conclusion, one sniffed message is not enough, but rather, the hacker would have to capture a large number of ciphered messages to be able to sort out the correct private key. This brings up another security concept, **opportunity**. If there is no chance of obtaining a larger volume of cipher data, breaking the system is less likely.

This example gives an idea of the power of public-private key system even if this was done with two small numbers inside a small number space. Notice, that the number 33 fits in a binary number space of at least 5 bits. In such small number range, 3 and 7 are the only numbers that work as a public-private key pair.



The number space of 5 binary bits contains the number 33 used as a modulo

Conceptually, that is a weakness of the system because it is known that the two numbers exist in such number space. In principle, it is a matter of trying all the possibilities, assuming that the data is available,

and testing until finding the right combinations. But this is an example on a small number space for educational purposes. What if the number space is made bigger, let's say 512 bits. How many numbers fit in a number space of 512 binary bits?

$$total\ numbers\ in\ 512\ bit\ space = 2^{512} = 1.340780792994259709957402499820 6 \times 10^{154}$$

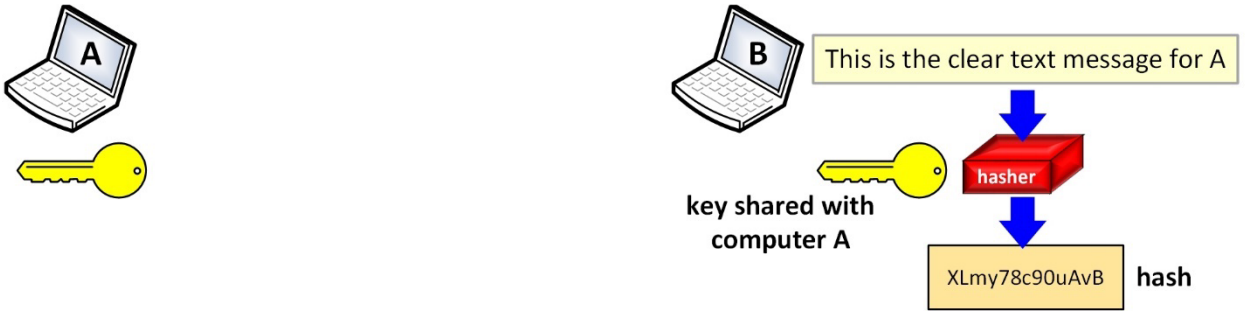
That is a number with 154 decimal positions which is an unfathomable quantity. In such vast number space, there are billions of possible key-pair combinations, so if one public key is known, where in the space of 1.34 x 10^{154} is the private key of the pair? Even for a powerful computer, it would take a long time to test all the combinations. The difficulty of guessing the other key of the pair is what makes the RSA system so strong. However, key spaces of 512 bits are considered not strong enough because supercomputers can run over all the cases in a feasible time. For that reason, the key length has been increased to 1024 bits, then 2048 bits and even 4096 yielding these gigantic number spaces:

- $total\ numbers\ in\ 1024\ bits\ space = 2^{1024} = 1.797693134862315907729305190789 \times 10^{308}$
- $total\ numbers\ in\ 2048\ bits\ space = 2^{2048} = 3.231700607131100730071487668867 \times 10^{616}$
- $total\ numbers\ in\ 4096\ bits\ space = 2^{4096} = 1.044388881413152506691752710716 6 \times 10^{1233}$

The larger the key space, the strongest the security. It is extremely difficult to find the other key of the pair even though the public key, the modulo and the algorithm are known. This is why the RSA public-private key system works.

Integrity

Integrity is the security concept that, when implemented, provides confidence that the data is accurate. In principle, data could be tampered by a malicious party or the data could contain errors. To counter these risks, methods that provide data integrity are needed. The fundamental method of data integrity is the **hash** which is a **checksum** or **digest** derived from the original clear text.



Computer B calculates a hash out of the clear text message

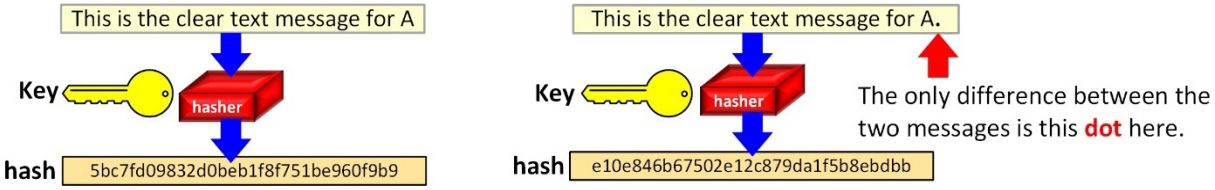
The clear data is supplied to a hashing method (for example **S**ecure **H**ashing **A**lgorithm SHA) together with an input key. The same algorithm and key must be available at the receiving end. The hasher "digests" the message and the key to produce a string of data with these properties.

- Hashes are not reversible. That means there is nothing to decipher.
- Hashes have fixed length regardless of the size of the data being hashed.
- Hashes are just digest fingerprints of the data.
- Hashes prove integrity of the data.

It is worth to note that hashing does not always require the use of a key. Data can be hashed without it. However, when a key is used in the hashing process, it introduces a secret element that reinforces the process of integrity checking. In this case, the key receives the name of “salt”. The only way that the same hash can be obtained is that the key is available.

Hash methods always yield the same hash length regardless of the size of the data being a few bytes or many bytes. For example, SHA-512 always produces hashes of 512 bits long while the older, and weaker, MD-5 always produces hashes of 128 bits long.

The following diagram shows the result of two messages done with an online hash calculator [1]. The hash algorithm chosen is MD-5 and there is an embedded fixed key (not shown in the calculator by default. Notice that the only difference between the two messages is a dot at the end of the right-side message.



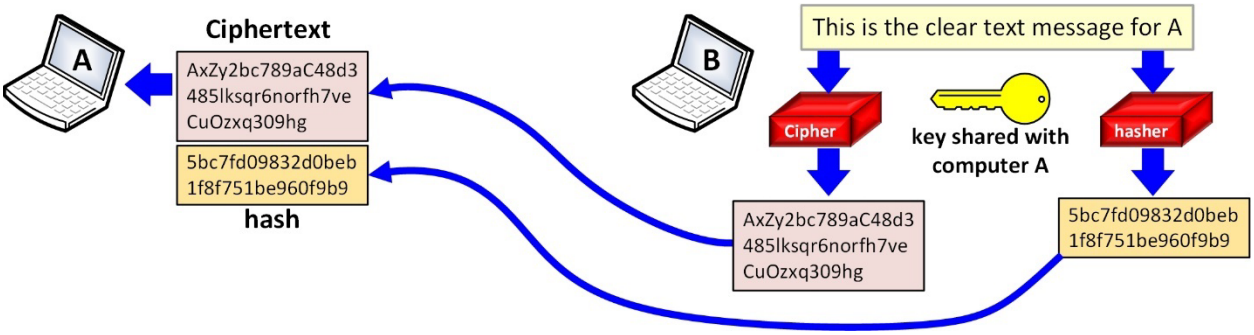
Hash results from similar messages

A desired property of a good hash algorithm is that it spreads and mixes the data so there are no discernible patterns. Even though the difference between the two messages above is a mere dot, the resulting hashes are so completely different as it can be seen by comparing them below.

5	b	c	7	f	d	0	9	8	3	2	d	0	b	e	b	1	f	8	f	7	5	1	b	e	9	6	0	f	9	b	9
e	1	0	e	8	4	6	b	6	7	5	0	2	e	1	2	c	8	7	9	d	a	1	f	5	b	8	e	b	d	b	b

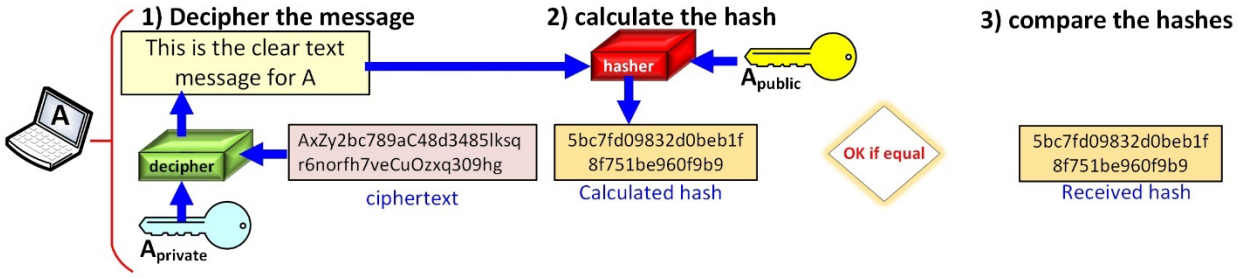
These two hashes have the same fixed length of 128 bits (32 hex characters times 4). Longer hash algorithms are stronger than shorter hash algorithms. For instance, SHA-512 is stronger than SHA-384.

If computer B sends a ciphertext to computer A plus the hash of the clear-text to validate its integrity, it is implicit that computer A is using the same algorithm, the same key and the same data to obtain the same hash result. Let’s assume that A’s public key is used for both ciphering and hashing.



Computer B sending a ciphertext and the hash of the cleartext

In the receiving side, computer A obtains the cleartext by deciphering. Afterwards, it runs the same process than computer B did to calculate the hash. Then it compares the two hashes and if they have the same value, then it concludes that the data has not been modified.



Computer A validates the hashes

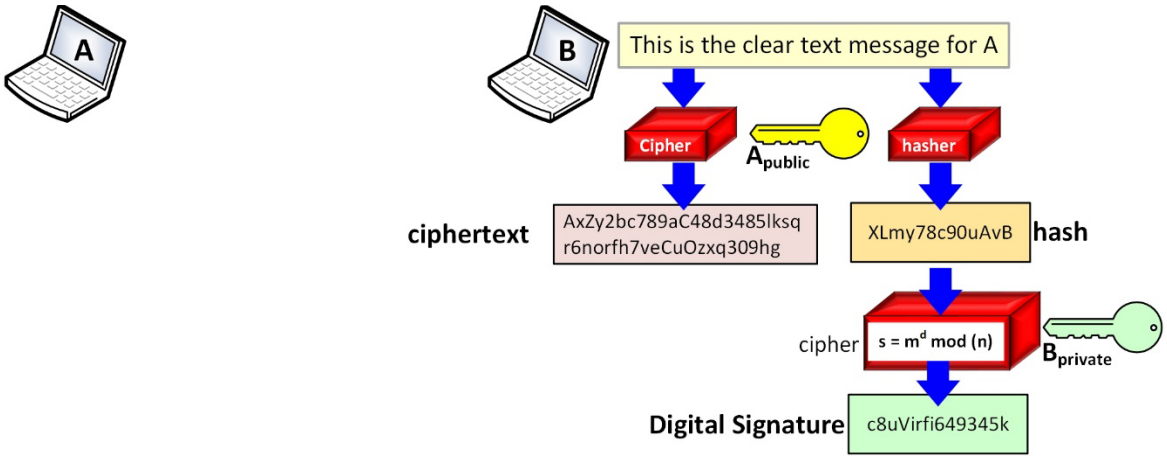
The procedure portrayed above provides secrecy and integrity however it has a weakness. If a malicious party obtains the hash in transit, it could try an attack. In principle, the hacker might have possession of the public key of A and since the hash algorithm is well-known, then there are too many pieces available to attempt a brute force attack. Even though for data hashed with a strong algorithm, it will take an unrealistic number of tries of combinations of data to arrive to the same hash. The next section expands on the security procedure to add authenticity.

Authenticity

Authenticity is the act of proving that the **data origin** is **legitimate**. The method used to provide authenticity is the **digital signature**. This is another application of the public-private key system. The sending party of the communication session accompanies the ciphered data with a string of data that is also ciphered but with the private key of the sender. The signature is created with the operation:

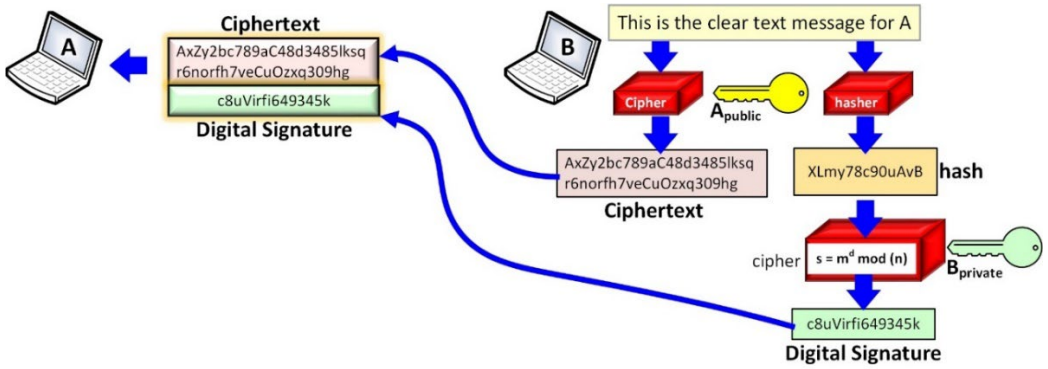
$$s = m^d \ mod \ n$$

Where s is the signature, m is a string of data, d is the private key of the sender (B in this case) and n is the modulo value. Let's start the explanation with an illustration.



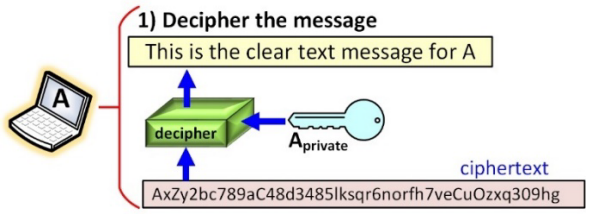
The process of signing the data digitally

In this example, computer B is sending a message to computer A. Therefore, it processes the message with the public key of A to obtain a ciphered message. Computer A will be able to decipher such ciphertext with its own private key. The difference now is that the same message is also sent to a **hash** processor. The result of the hash is a **checksum value** of the message that validates its **integrity**. If even one bit of the message were to change for any reason, the hash will not match ay the destination, thus declaring the message invalid. The checksum is passed thru another module that ciphers it with the private key of the sender (B). That might sound odd because anyone with the public key of B will be able to open it, but what this process is meant to prove is **authenticity** or **legitimacy** of origin. If anyone can decipher the hash with the public key of B, then it only proves that B sent the message for only B could have **signed** it with its private key. Once that the process is complete, B sends the ciphertext and the **digital signature** to A.



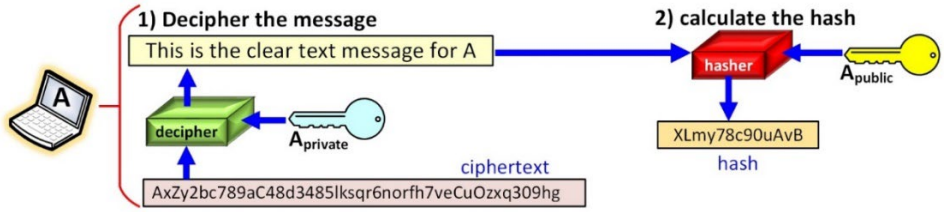
Computer B sends the data ciphered and the digital signature to computer A

Computer A receives all these data and it proceeds first to decipher the cipher-text using its private key since the data was created by B using A's public key. The result is the clear text message.



Computer A deciphers the cipher-message

Once that the cleartext is obtained, computer A needs to **validate** that the message is **legit** and **correct**. Hence, it proceeds to calculate a hash of the message. It feeds the hasher protocol with the clear-text message and its own public key (A) to create a digest value of the message. This hash should be identical to the hash previously calculated by computer B because it is the same message and the same key.

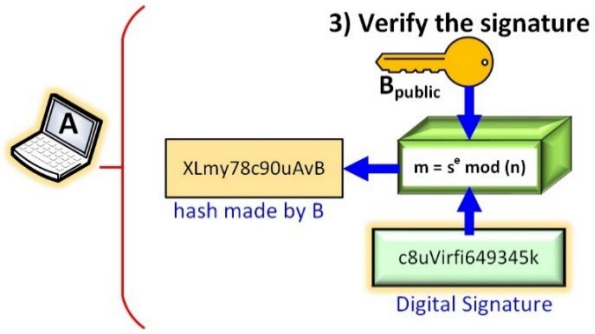


Computer A calculates a hash of the clear text message

Now, computer A needs to compare the hashes so it deciphers the digital signature of the hash created by computer B. Since this computer signed the hash with its private key, computer A can open it with the public key of B. This proves that the signature was made by computer B because only the public key of B can be used to decipher something ciphered with the private key of B. This proves **legitimacy** that B signed the hash. The operation to verify the signature is:

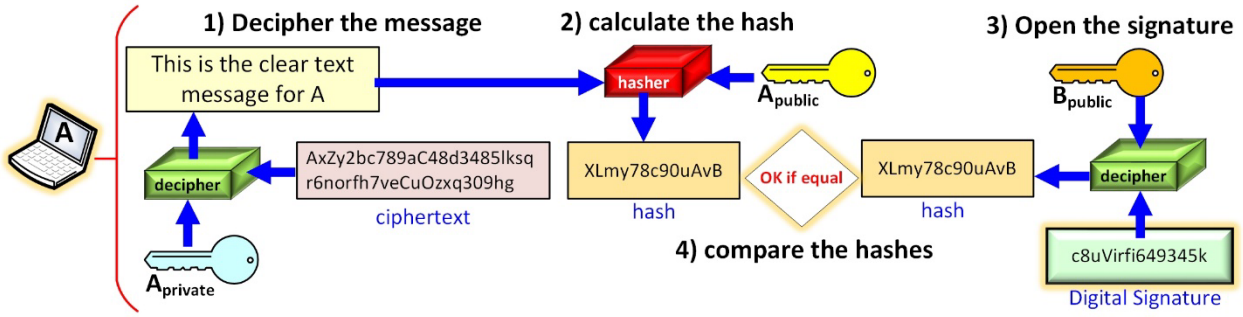
$$m = s^e \ mod \ n$$

Where m is the deciphered original data, s is the digital signature, e is the public key of the sender (B in this case) and n is the value of the modulo. The process is shown in the figure below:



Computer A verifies the digital signature of the hash

Now, it remains one more step of the process. Computer A needs to verify that the message was not altered in transit so it compares the hash obtained from the deciphered message with the hash that came ciphered inside the digital signature. If the hashes are equal that means that the message is correct, this proves **integrity**. But if the message differs, by even one bit, the message must be discarded as it was probably tampered or maybe it has an error.



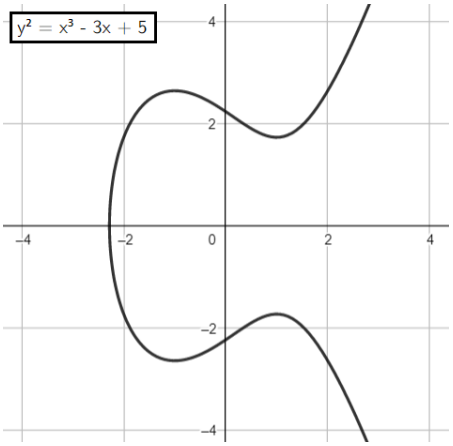
Computer A deciphers the digital signature of the hash

This is a robust security system that ciphers the messages, validates the authorship of the message and that the message is correct. In short, the system implements the **security principles** of **secrecy**, **authenticity** and **integrity**.

Elliptic Curve Cryptography (EC)

The confidence on the RSA security system rests chiefly on the key size. As computer power has continued to increase, the potential for breaking the RSA system has also increased. The response to this challenge has been to keep increasing the key size. But this is a problem for low power equipment such as mobile devices since the processing of the keys consume resources.

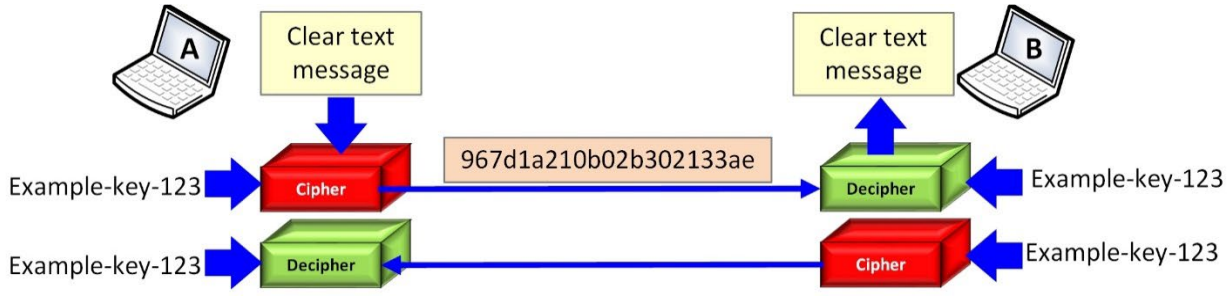
Elliptic curve cryptography is an alternative to RSA that can offer the same or greater level of protection but with smaller size keys than RSA. This system is based on the calculation of the public and private key using graph mathematics over elliptic curves. The computer choose one out of the many elliptic curve equations and then it starts a series of iterations easy to calculate (for a computer, not for a human) but extremely hard to reverse. At the moment of this writing, elliptic curve algorithm are being adopted at a fast pace by most security protocols and it is also the algorithm being used in blockchain technologies.



Sample elliptic curve

Symmetric key ciphering

Symmetric key systems are based on the principle that the participating nodes **share the same key** to cipher the communications. In contrast, asymmetric systems require four different keys for every pair of communicating nodes. In fact, public-private systems are computing intensive as they consume resources to do all the calculations, keep track of the keys, and the ciphering and deciphering of the data. Symmetric key system require less resources than asymmetric systems, so that makes them attractive to use. The same key is supplied to the cipher and decipher mechanisms.



The symmetric key systems use the same key to cipher and to decipher

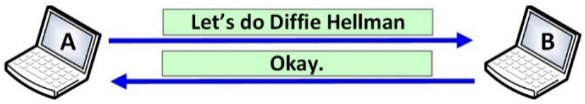
An inherent problem with symmetric key systems is the distribution of the key. For two nodes A and B to cipher their communications, the same key must be installed in both machines but doing that manually is out of the question. However, the key can be distributed in a dynamic way. For example, the key could be exchanged ciphered using a public-private system. Once that the key is in place, the communications session switches to the symmetric key system which is less demanding for the computers involved.

There is another way to use symmetric keys which does not distribute the key at all; instead, both communication partners calculate the same **session key**. Such a system is explained next.

Diffie-Hellman (DH)

Diffie Hellman (DH) is a process to generate a **secret session key** via the exchange of messages over an unsecured network between two computers. This brilliant system allows the two communicating parties to calculate the same session key that will be used for both ciphering and deciphering. Diffie Hellman is not a ciphering algorithm, but a process to calculate the symmetric key in a dynamic exchange of data. Furthermore, the key resulting from the DH process is passed onto a cipher system (such as Advanced Encryption System AES, for example).

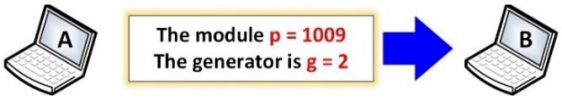
The Diffie-Hellman process is found at the start of a secure protocol transaction (for example, the SSH protocol uses DH). One of the nodes tells the other to start a Diffie Hellman process.



Computer A asks computer B to initiate a Diffie Hellman process

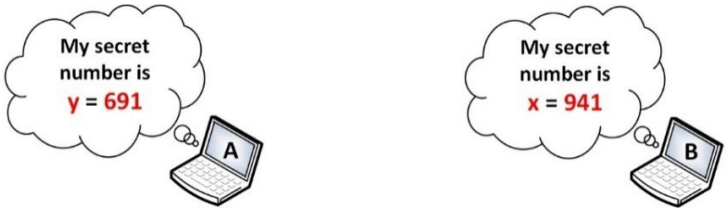
Note: In the following example, small numbers are used to make the explanation possible, but in real life, the numbers used as extremely big.

Computer A proposes to B, two components to be used in the operation. The **modulo number p** where the calculation will be done and the **generator g base number**. These values are exchanged in clear text format so an observer of the traffic can see the numbers in the message.



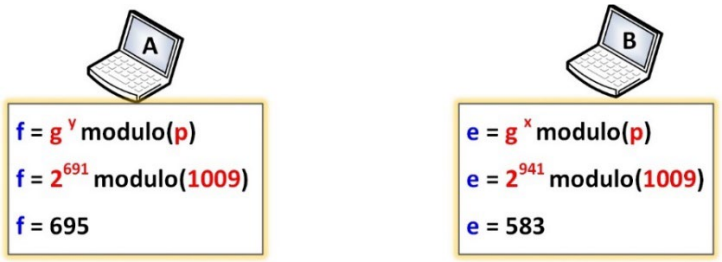
Computer A sends the p and g numbers to B

Both computers find now **large prime numbers** within the large modulo number space. These numbers never leave the respective computers because they are the **secret component** of the operation. (Again, in reality these numbers should be extremely big prime numbers).



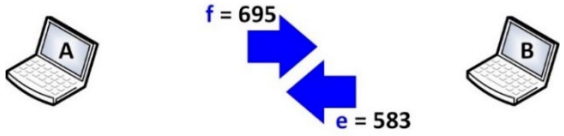
Each computer finds its own secret prime number

The process continues with each computer doing a calculation with the equation $g^{secret} modulo(n)$. Notice that the equation, the generator and the modulo value are public, the only hidden components are the secret values on each side of the operation.



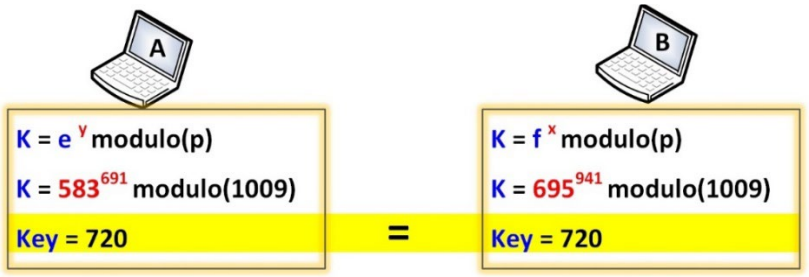
Each computer performs the same operation

Now, the two computers exchange the values found in the previous step and again all of this happens in a clear-text exchange of data between A and B.



The two computers exchange their $g^{secret} modulo(n)$ values

Finally, the two computers do a calculation involving the exchanged values and the secret components. Amazingly, the result is the same key value (K = 720) on each side. The key itself is never exchanged and it only last for one communication session. Every new session involves a new key calculation. Diffie Hellman only calculates keys, it is not a cipherer. Once that the key is calculated, it is supplied to a cipher system block, AES for example, to start the ciphering of data.

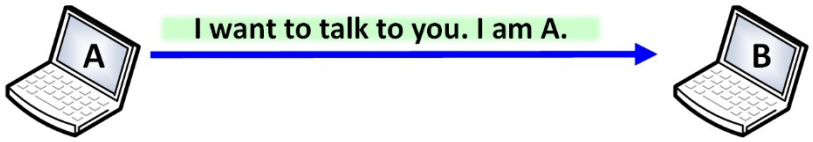


The computers calculate the secret session key

Diffie-Hellman is a remarkable invention dating from 1976 [2] and it is used in secure protocols such as SSH, IPSec, and SSL/TLS. Even though the message exchange is done in the open, it is virtually impossible to guess the secret component and the session key if the number space is very large (2048, 4096 bits). Since the ciphering and deciphering is done with the same key, the processing cost is lower than using the public-private RSA system. Another advantage is that the key is only used for one session, so there is nothing to be stored that can be potentially stolen. Moreover, elliptic curve cryptography has been incorporated into the DH process to create the components of the operation providing an even higher level of security.

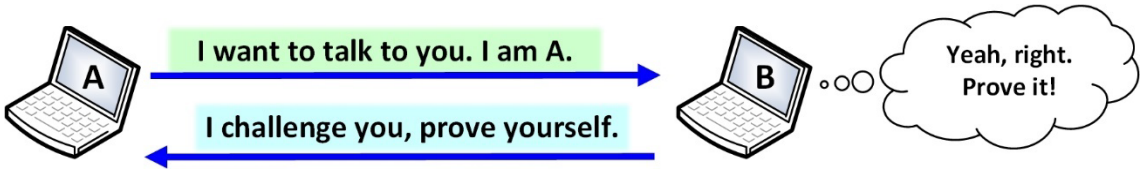
Authentication

The process of authentication consists of proving the legitimacy of the client trying to access a resource in another computer. Only an authenticated client can gain access to a remote computer. The following illustrated transactions represent the general philosophy behind the authentication process when the user of computer A tries to access the remote computer B.



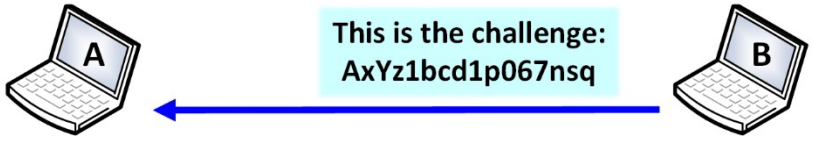
Beginning of the Access Process

Generically, A requests to access B; however, B can not just allow that. First, it will make A to prove its identity. It does so, by supplying a challenge. The access will be allowed only if A passes the challenge.



Computer B challenges A to prove its identity

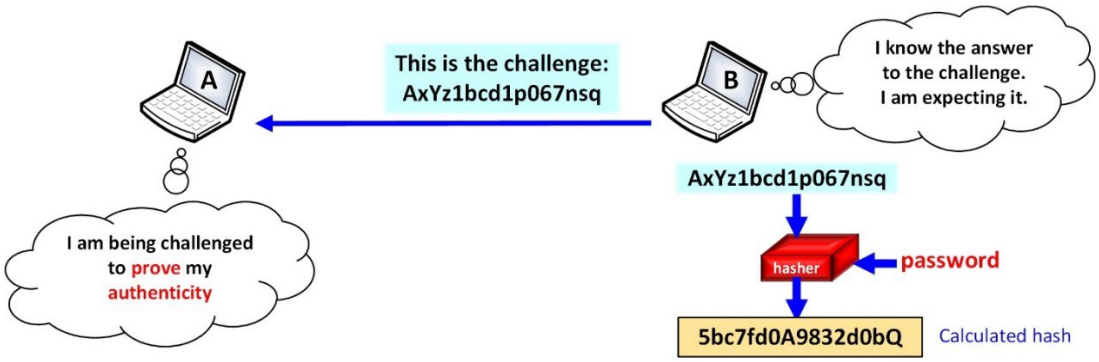
In this example, a string of data is delivered to A. In turn, A has to reply with an answer that only A could possibly create. That means, that A and B must have some secret in common.



Challenge data is delivered to A

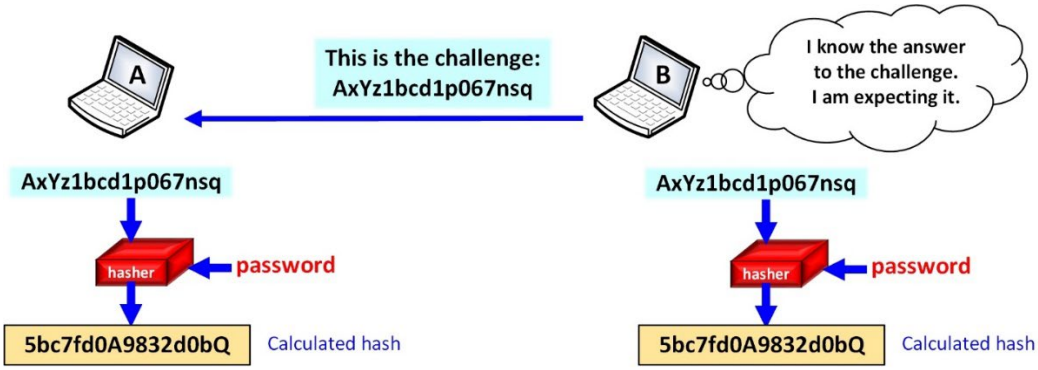
The secret component in this example is a password hence this is a password authentication method. Both A and B share the password. This is a common method present in many communications system. The idea is that B has a list or a database with users and corresponding passwords. When A says “I am A” is not enough, it has to proves its authenticity. That is why B challenges A with a string of data. The answer coming back from A must be something that only A can create. So, indirectly, this proves authenticity or identity.

Computer B sends the challenge data while locally it also creates a hash of it with the password of A. This is the answer that is expected from A. If the answer does not match, then B can not be allowed in. However, if the answer is exactly the same that B has calculated then that indirectly proves that A is A because the only way that the answer can be the same is that it was created with the same password.



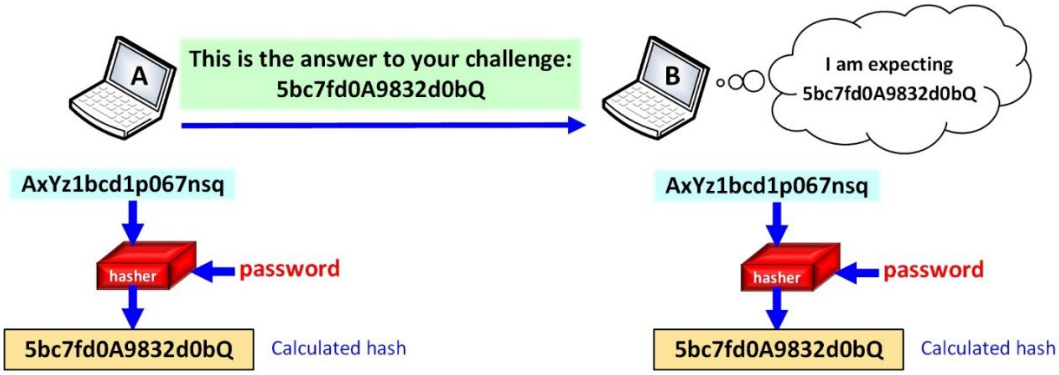
Computer B expects certain answer from A

When A receives the challenge, it does the same process than B has done. It creates a hash of the challenge data with its password.



Computer A creates a hash with the challenge data and its own password

Computer A sends the hash of the challenge data to B. The answer must match to the one being expected.



Computer A replies back to computer B

Now, this procedure has a weakness. Let's assume than an evil hacker has been sniffing the conversation. Consequently, it has obtained the challenge in the way from B to A and the answer to the challenge in the way back from A to B.

The hacker saw the challenge and the answer

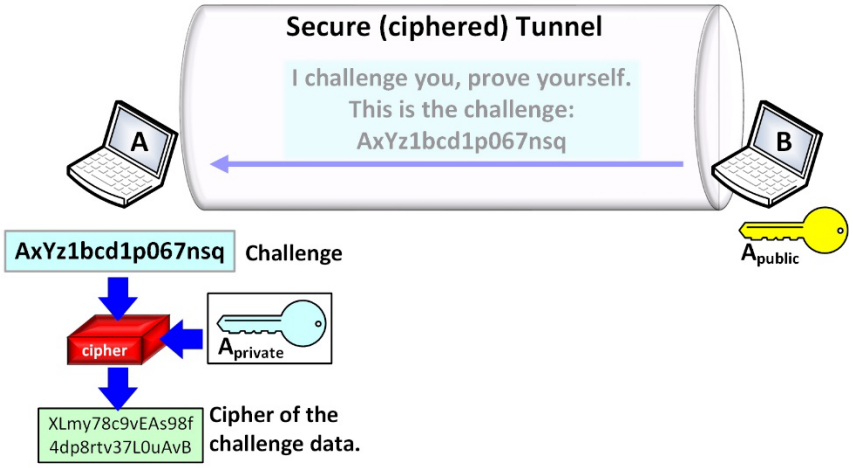
Thus, the hacker has two pieces of information out of three. The only piece missing piece is the password. The hacker can now attempt a brute force dictionary attack. This consists of testing the challenge through the same hash algorithm with the passwords supplied by a huge dictionary with the most commonly used passwords. If that does not give a matching answer, then it tries with all kinds of combinations until a match is found. If the password is weak, it can be obtained and that is obviously bad.

The hacker performs a dictionary attack

Consequently, the transactions can not be allowed to happen in the open. But what if the authentication conversation were conducted within a ciphered tunnel? In such case, the hacker could not get the challenge and the reply. The problem is how to create the tunnel in the first place. There are different ways to do this, but Diffie Hellman is the most common. Before the challenge messages are exchanged, a Diffie Hellman negotiation creates a session symmetric key that is used to cipher the conversation.

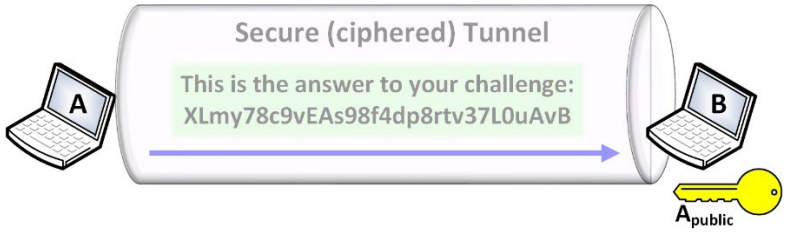
A ciphered tunnel hides the authentication exchange

If the client A does not pass the authentication challenge, the tunnel is broken down and the process is terminated by computer B. Furthermore, the authentication process can be made even better or stronger. Instead of using a password which is considered weak authentication, public-private keys are used.



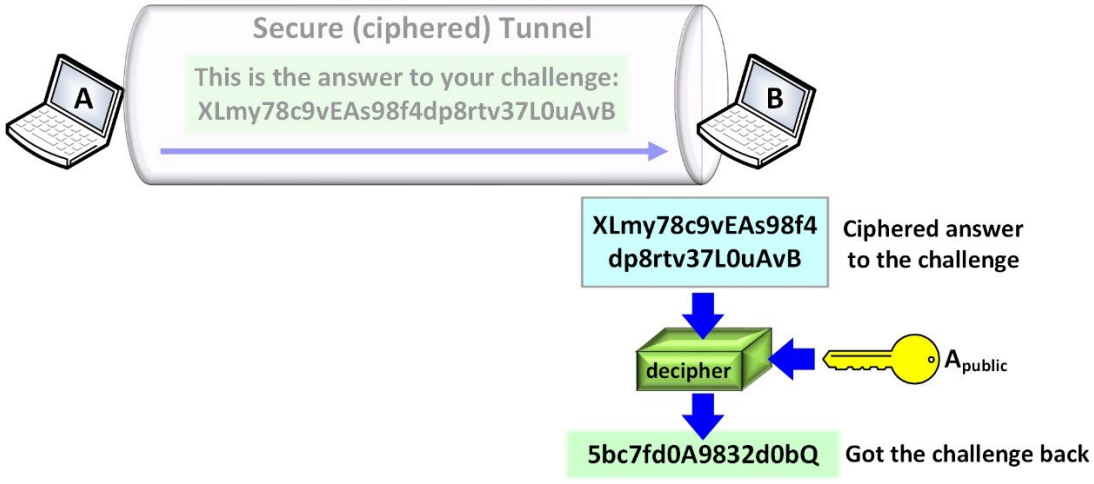
Computer A ciphers the challenge data with its private key

For this system to work, computer B must have A's public key. Computer A ciphers the challenge string of data with its own private key. Only the public key of A can decipher that.



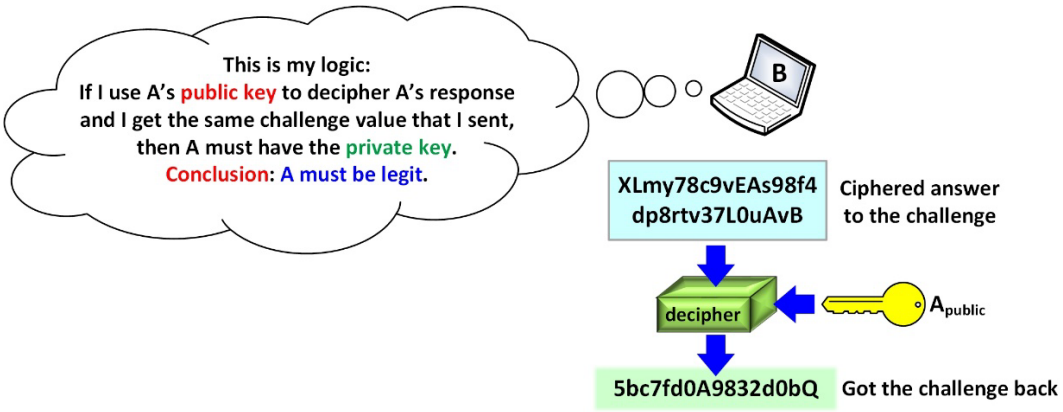
Computer A returns the challenge data ciphered with its private key

Computer B receives the ciphered challenge and it proceeds to decipher it with the public key of A.



Computer B obtains the challenge data back

If the resulting clear text is the same as the challenge, then that proves that it was ciphered with the private key of A and since the only computer that could have such key is A that proves that it is authentic.



Computer B authenticates computer A

Learning Activity

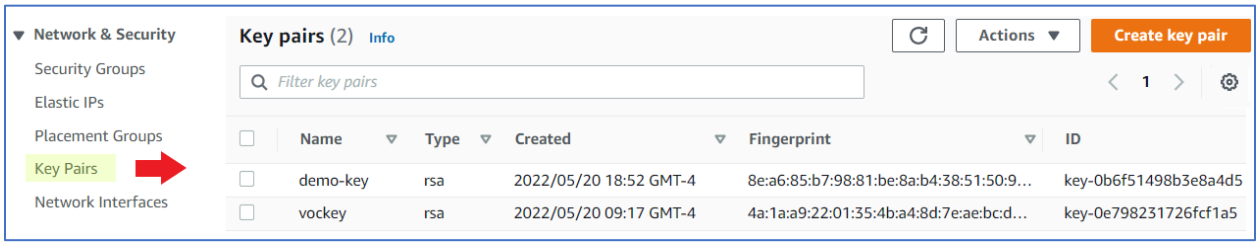
Observing the Security Principles when Accessing Cloud Resources

Secure Shell (SSH) Protocol

SSH is a protocol for securing communications between two ends over an unsecured network. It mode of operation is client-server where the end-node that initiates the SSH exchange is called the SSH client while the end-node that is the target of the session is the SSH server. SSH most common application is to secure remote login procedures. Particularly, it is the protocol used to access AWS EC2 instances in a secure manner. In this learning activity, this specific case is analyzed step by step to observe the application of network security concepts.

Understanding the Security Key Pair

The AWS service Elastic Computer Cloud has a section dedicated to network security where the key pairs are managed.



- What are these key pairs?
- What is their function?
- How are they used?

To answer these questions, let's start by creating a new key pair named test-key. There are two key **algorithms** available Rivest, Shamir, and Adleman (RSA) and Edwards-Curve Digital Signature Algorithm with the Elliptic Curve 25519 (ED25519) to create the key pair. RSA is an algorithm that creates public