# Authentication and Delegation in Internet Aplications

Isaac Agudo

***Abstract:* The paper presents an overview of classical authentication schemes actually used on the internet. It shows that they are not suitable for an environment where Delegation is needed and presents public key cryptography as a option for the use of Delegation. It ends with some comments on X.509 and further extensions.**

***Key words:* Authentication, Authorization, Delegation, Access Control, Information Security**

## INTRODUCTION

Authentication is and has been the base for authorization in all enviroments, from real life to the Internet. Authentication is carried out when you pay by credit card, when you open the door of your home and of course when your read your e-mail or access your bank account by Internet.

In all these cases the users have to demonstrate to a second entity that he/she is who they are supposed to be. Depending on the authentication mechanism method used, we can conclude a simple classification of Authentication methods. Although authentication is a very important component for access controls, it is not enough. There is also a need for defining authorization policies, thus

***Access Control = Authentication + Authorization***

But Authorization may not necessarily require a previous authentication phase. This is the case of Attribute Certificates, which can be anonymous, and in which authentication is implicit. In this paper we are going to study only the authentication phase.

Authentication is usually carried out by a Challenge and Response mechanism. Usually, the system asks you a question that only you know the answer to (i.e. a password), or asks you for something that only you have (i.e. your identity card). There is another way of authentication based on your physical appearance, that is called biometric authentication but falls outside the scope of this paper.

## CLASSIC AUTHENTICATION

In this section we present a short overview of some authentication methods that are widely used on the Internet and that are suported by the Simple Authentication and Security Layer (SASL) [4]. We have classified them according to the mechanisms used to authenticate users in the System. We describe the common aspect of each mechanism and then present an example.

Three main aspects are emphasized:

- ***Registration.*** What are the requirements for the users to be able to be authenticated by the system.
- ***Storage of Information.*** What information is needed in order to authenticate the User.
- ***Authentication Mechanism.*** How is the user authenticated, and what is the basis for this authentication to be safe.

### User/Password

This was the first authentication method used in computer systems and is still the most widely used authorization method. The strength of this scheme is based on the ability of the user to keep his/her password secret. As a consequence, when a User/Password scheme is used on the internet, the communication between the user and the system must be confidential. Once the password is revealed, the user has to change it. As an example we explain the scheme used in many UNIX [1] like systems.

o ***Registration Phase***

In the registration phase, the user has to interact directly with the system. The User chooses a user name, or user ID, which is unique in the system and then the user is prompted for a password. In some systems, the user is given a temporary password that has to be changed as soon as possible. This avoids the direct interaction of the user in the registration phase but can be a flawed point if the attacker knows the password generation algorithm.

o ***Storage of Information***

An initial approach could be for the system to store pairs of the form ***User/Password*** in a local file. This solution is not safe as a local user may access this file and discover all the passwords. Instead, the system stores a cryptographic hash function of the users' passwords. One problem of this solution is that the password hash can be used to derive the plain text using a dictionary attack. If the hash function depends only on the password, we could guess the password by using a big enough dictionary of hashed text to plain text. The more complex the passwords are, the bigger the dictionary should be. Then, it is necessary to complete the password with some ramdom information in order to make a dictionary attack more difficult for the hacker. This is done by adding some random information, usually called "salt", to the user password.

The cryptographic hash function used is a 25 round encryption of a 64-bit string of zeros with the DES algorithm using the password as the encryption key and the salt is a random 12-bit integer which is used to permutate the output of each encryption round. This random information multiplies the size of a typical dictionary by 4096.

o ***Authentication Phase***

When the user types the password at the prompt stage, the system computes the cryptographic hash function of the password using the salt and compares it to the value stored in the local file. If the two values are the same, the system grants access to the user.

o ***Comments and Variations***

As previously mentioned, the strength of the system is based on the fact that the user has a single password that must be kept safe. Even if communications are confidential, with time, the old password can be revealed by a brute force attack, then the user is allowed to change the password and each password could have a validity time interval, in order to prevent this kind of attack. This scheme is quite safe in local systems, but weak in distributed environments. A easy extension is the use of two passwords. The first one, we could call "Access Password" or "Session Password", and is used to access the system and do some non critical tasks. This initial password could also initiate a ciphered connection. Then when the user requests a critical action, he is prompted for the second password or "Operational Password". One advantage of this modification is that it increases the complexity of the system as the attacker has to guess double the information. Moreover, the "Operational Password" is typed under a double ciphered connection (i.e. SSL and DES with the access password), and before discovering the "Operational Password" the attacker has to discover the "Access Password". This latter scheme is used in most e-bank web portals, but the users are given the password and the username by the system to avoid the use of easily obtainable passwords.

We said that this scheme is almost safe when the communication between the client and the server is ciphered, but this is not the only requisite. In computers nowadays, when you type the password on the keyboard, it goes through several process before reaching the destination server, moreover, cordless keyboards can be easily intercepted. This was not a problem in older computers without multiprocessing capabilities, but with modern computers this problem of keeping the password completely secret is a real challenge.

Indeed, for this very reason bank portals are moving to the use of the mouse instead of the keyboard. They use a virtual keyboard in which you have to press the keys of the password, one by one. Another modification is to use the MD5 [8] cryptographic hash function instead of DES that highly increases the number of possible passwords by using more salt information.

**One Time Password (OTP)**

OTP [5,6] is an evolution of the user/pass scheme for distributed environments, in which there is a need for higher security. As each piece of information interchanged by the user and the server can be inspected by other users, a single password can be jeopardized. Once we use a password, it is no longer safe and we have to change it. The goal of OTP is to provide a large number of passwords that depend only on a single secret password. OTP allows us to avoid the use of SSL or another confidential channel for authentication as passwords are single use only and can be sent in clear text via the internet.

In OTP we have two kind of passwords:

- ***Master Password.*** It is the password used to generate the session passwords and it is never revealed to any entity.
- ***Session passwords.*** These passwords can be used only once and are commonly numbered.

Lamport [2] proposed a simple scheme which depends on the use of a cryptographic hash function, $f$.

- ***Registration Phase***

First, the system chooses a large integer $n$, which represents the number of passwords that will be given to the user and therefore the number of times that the user will be able to login to the server without resetting the registration information. Then the user, or the system, computes $f^n(P_U)$ (that is the application of $f$, $n$ times) where $P_U$ is the Master password of the user plus a ramdom data called ***seed***.

- ***Storage of Information***

The information $(UserName, n, seed, f^n(P_U))$ is stored in the system in order to Authenticate the user against the system. This information does not help to compute $P_U$ as $f$ is supposed to be a one-way function.

- ***Authentication Phase***

The challenge is to compute $f^{n-1}(P_U)$, then the system applies $f$ one more time and checks the result with the stored information $(UserName, n, seed, f^n(P_U))$. If the result matches the stored information, the system decreases $n$ and replaces the stored value with $(UserName, n, seed, f^{n-1}(P_U))$.

- ***Comments and Variations***

In some cases the user only knows the session passwords and is not able to generate any more session passwords. The session passwords can be written on a paper document or stored as a physical device. Some Bank Web portals allow users to authenticate using an "Access Password" as defined in the previous section and then using the list of One Time Passwords as "Operational Password". This scheme is used, for example, by the German Post Banks online portal (http://www.postbank.de). A fully functional implementation of the scheme is on the system ***S/KEY*** [7].

**Authentication Tokens**

There are also hardware solutions for OTP, In these solutions, the user owns a hardware token that provides him with the session passwords, and on the server side there is a hardware checker that verifies the validity of the session passwords. One example is the SecurID (https://www.rsa.com/node.asp?id=1156) solution by RSA. RSA provides different kinds of tokens and also provides software tokens that could be installed in Workstations, PDAs and smartphones.

o ***Registration Phase***

The user is given a SecurID token, which is associated with his/her identity in the RSA ACE/Server. Then the user chooses a ***userID*** and a ***PIN*** (Personal Identification Number) that has to be kept secret.

o ***Storage of Information***

The RSA ACE/Server has to be installed and the ***user ID***, the ***PIN*** and the token code (serial number) are stored together. These values associate the user with the token via the PIN.

o ***Authentication Phase***

SecurID consists of a hardware token with a value that changes every 60 seconds or so. The token is synchronized with an RSA ACE/Server, which validates the authentication attempt. When you are prompted for authentication, you will be given a passcode prompt. Depending on the type of SecurID card you have, you will either type in a PIN (four to eight alphanumeric digits in length) followed by the six-digit number currently displayed on your SecurID card, or you will enter the PIN on your SecurID card, press the diamond key, and type in the number displayed on the SecurID card. Because the SecurID card and ACE/Server are in sync, the ACE/Server knows what the SecurID card should read at any given moment.

o ***Comments and Variations***

There are also software implementations of the SecurID tokens that can be installed in many devices (WorkStations, PDAs, SmartPhones, …).

A critical point of this approach is the storage of the PIN and the token code. They have to be shared by the client and the server and subsequently have to be stored in a safe way. Another consequence is that the administrator of the ACE/Server knows all the ***(PIN, userID, token code)*** combinations and is then able to impersonate every user in the system. We conclude that the ACE/Server is a very critical component.

In some applications, security tokens are combined with User/Password Authentication. In these cases, the SecurID is used to provide the "Operational Passwords" to the client as in the OTP scheme.

Compass Bank (http://www.compassweb.com), for example, uses RSA securID to protect Wire Transfers, after a previous User/Password authentication.

There are some other solutions that use Smart cards but they are essentially asymmetric cryptography approaches which will be discussed in a later section.

**Symmetric key Authentication**

In this scheme, the password is replaced by a symmetric key that is shared by the user and the system. As in the SecurID approach, the server has to keep safe all the symmetric keys of the user and is also able to impersonate any user. We have to trust in the server considerably when using this approach.

o ***Registration Phase***

In the registration phase, the user and the system agree on a secret key and on a symmetric encryption algorithm. Of course, the user also has to choose an ID.

o ***Storage of Information***

The server has to store a cryptographic key for each user in the system. In the password based solutions, passwords were never known by the server, instead, the server stored hash functions of the passwords.

In this scheme, the server must know the secret key and can then impersonate the user, thus the server must be a Trusted party for all users. If this is not the case, a Third Trusted Party is necessary in the system to authenticate the user with the server.

- ***Authentication Phase***

The authentication challenge is to cipher a random number sent by the server and the response should be the correct cipher text. In an example:

If Alice is trying to authenticate Bob, she sends to Bob a random number $N_A$. Then, Bob encrypts $N_A$ with key shared by Alice and Bob, $K_{AB}$, obtaining the cipher text $E_{K_{AB}}(N_A)$, and sends it back to Alice.

When Alice receives $E_{K_{AB}}(N_A)$, she computes $N_A' = D_{K_{AB}}(E_{K_{AB}}(N_A))$. If the key is correctly shared ($K_{AB}$ is the same for Alice and Bob) then $N_A' = N_A$ and supposing that only Alice and Bob know $K_{AB}$, Alice has Authenticated Bob. If $N_A' \neq N_A$ then Bob is not who is he claiming to be.

This the basic version of the protocol, which is flawed. To define a correct protocol we should use Manipulation Detection Codes (MDC) instead of a cryptographic function.

- ***Comments and Variations***

As previously mentioned, if the server is not fully trusted there is a need for a TTP. The Woo-Lam protocol provides a simple solution using a TTP.

1. Alice $\rightarrow$ Bob: ***Alice***;
2. Bob $\rightarrow$ Alice: $N_B$;
3. Alice $\rightarrow$ Bob: $E_{K_{AT}}(N_B)$;
4. Bob $\rightarrow$ Trent: $E_{K_{BT}}(Alice, E_{K_{AT}}(N_B))$;
5. Trent $\rightarrow$ Bob: $E_{K_{AT}}(N_B)$;
6. Bob decrypts the cipher text using $K_{BT}$.

This is a very complex approach that is used in many security protocols and applications. For example, the Kerberos system makes uses of this protocol (with some modifications) to provide authentication using symmetric cryptography

## AUTHENTICATION TRANSFERENCE AND DELEGATION

In some cases, It may be desirable for users to be able to transfer authentication to other users. For example, in a company, the Director could change over time, then when a new director comes, the old one has to transfer his authentication tokens (passwords, OTP lists, secret keys, etc.) to the new Director. This is a previous concept for Delegation. We will see how each of the previous mechanisms manages the transference of Authentication.

- ***User/Pass.*** If you give your password to some other user, he gains the same access as you to the system. One possible mechanism to limit this access temporarily could be to change the password, but if the system allows users to change their own passwords, you could lose access to it, if the user to whom you have lent your password, changes it before you do so. Then the User/Pass scheme is not suitable for delegation, given that when you transfer your User/password to someone, you may lose it forever.
- ***One Time Password (OTP).*** If you only give or transfer one session key, the user can not access the system more than once, this could be a short time or a long time depending on the maximum time per session of the server. But in any case he gets full access to the system, as he can not be distinguished from the legitimate user.

- ***Authentication Tokens.*** You can lend your SecurID to another user and when it is returned to you, you recover access to the system. Meanwhile you do not have the SecurID, so you can not access the system. The two users, the grantor (i.e. the person who grants the permission) and the grantee (i.e. the person to whom it is granted) can not access simultaneously but they can not be differentiated from each other thus this not a safe way to transfer authentication.
- ***Symmetric key Authentication.*** As with the Password based approach, if you lend your secret key to a third party you lose control over your authentication.

Therefore we need another way of authentication to provide delegation capabilities, and public key cryptography offers us the perfect solution.

**Asymmetric key Authentication**

Asymmetric cryptography, also known as Public Key Cryptography (PKC), provides a powerful mechanism for authentication that is highly suitable for Delegation. In this scheme, each user has a pair of keys $(\boldsymbol{K_{pr}}, \boldsymbol{K_{pb}})$ (In fact, keys are included in certificates that have more information about the users) named private and public key respectively. The main requirement for asymmetric cryptography is that $\boldsymbol{K_{pr}}$ can not be derived from $\boldsymbol{K_{pb}}$ in polynomial time and that both keys are inverses of each other in the following sense:

$$\boldsymbol{D}_{\boldsymbol{K}_{pb}}(\boldsymbol{E}_{\boldsymbol{K}_{pr}}(\boldsymbol{M})) = \boldsymbol{M} \tag{0.1}$$

Let us think of a situation in which Alice is trying to authenticate Bob and where Alice knows Bob's public key, then the authentication scheme is as following:

1. Alice sends Bob a random message ***M***
2. Bob encrypts the message ***M*** with his private key and gets $\boldsymbol{E}_{\boldsymbol{K}_{pr}^{B}}(\boldsymbol{M})$.
3. Bob sends $\boldsymbol{E}_{\boldsymbol{K}_{pr}^{B}}(\boldsymbol{M})$ to Alice.
4. Alice calculates $\boldsymbol{M}' = \boldsymbol{D}_{\boldsymbol{K}_{pb}^{B}}(\boldsymbol{E}_{\boldsymbol{K}_{pr}^{B}}(\boldsymbol{M}))$

If $\boldsymbol{M}' = \boldsymbol{M}$ then, by equation $(0.1)$ Alice can make sure that the message could have originated only from Bob and nobody else, since Bob keeps his private key secret and nobody can derive Bob's private key from the public key. The message $\boldsymbol{E}_{\boldsymbol{K}_{pr}^{B}}(\boldsymbol{M})$ is the digital signature of ***M***.

**Delegation**

Suppose Alice is trying to authenticate Jack and suppose Jack delegates Bob.

1. Alice sends Bob a random message ***M***, but she thinks she is talking to Jack
2. Bob sends Jack the message ***M***, asking for delegation on this session.
3. Jack sends to Alice a delegation Statement, $\boldsymbol{E}_{\boldsymbol{K}_{pr}^{J}}(\text{"}\boldsymbol{I\_trust\_Bob}\text{"}, \boldsymbol{K}_{pb}^{B}, \boldsymbol{M})$. He can send the message via Bob in case they are not in the same comunication channel.
4. Alice decrypts the message with the Jack's public key and gets $(\text{"}\boldsymbol{I\_trust\_Bob}\text{"}, \boldsymbol{K}_{pb}^{B}, \boldsymbol{M})$.
5. If ***M*** remains the same in the received message, Alice understands that Bob can act on behalf of Jack and then tries to authenticate Bob using the previous protocol with $\boldsymbol{K}_{pb}^{B}$.

This is a basic academic example of delegation protocol that needs Jack to be online. An offline version can be considered in which Alice does not require the random message $\boldsymbol{M}$ to be in Jack's response but the drawbacks are that Jack can not change his opinion on Bob. Another modification, to avoid this problem of offline delegation, could be to include the validity interval of the delegation statement in the cipher text:

$$E_{K^{J}_{pr}}(\text{"}I\_trust\_Bob\text{"},\ form = 15/02/2005, to = 17/03/2005, K^{B}_{pb}, M)$$

Jacks delegates to Bob, but Bob may also delegate to Frank, this situation produces a sequence of statements of the form:

$$E_{K^{Grantor}_{pr}}(\text{"}I\_trust\_Grantee\text{"}, K^{Grantee}_{pb}, M)$$

these sequences are called delegation paths.

The main advantages of Public Key Cryptography over the classical Authentication Methods are:

- The public key is the only piece of information needed to authenticate the users, and this information is not sensitive. It can be posted on any web page
- Public Key Cryptography allows delegation thanks to the use of delegation paths or chains of trust.

But there are also some disadvantages:

- Asymmetric is expensive, both in storage space and computation time.

## CONCLUSIONS AND FUTURE WORK

PKC is an authentication method that has to be combined with an authorization policy to obtain a functional system. More Modern approaches like, Privilege Management Infrastructure (PMI), try to solve the problem of authorization in only one phase. It combines the authorization policy with the authentication information in Attribute Certificates.

## REFERENCES

[1] F.T. Gramp and R.H. Morris. Unix operating system security. ***AT&T Bell Laboratories Technical Journal***, 63(8):1649-1672, October 1984.

[2] Lamport, L., "Password Authentication with Insecure Communication", Communications of the ACM 24(11), 770-772, November 1981.

[3] Haller, N., "The S/Key One-time Password System", Proceedings of the Symposium on Network \& Distributed Systems Security, pages 151-157, February 1994.

[4] RFC 2222 - Simple Authentication and Security Layer (SASL) (http://www.ietf.org/rfc/rfc2222.txt)

[5] RFC 2444 - The One-Time-Password SASL Mechanism (http://www.ietf.org/rfc/rfc2444.txt)

[6] RFC 2289 - A One-Time Password System (http://www.ietf.org/rfc/rfc2289.txt)

[7] RFC 1760 - The S/KEY One-Time Password System (http://www.ietf.org/rfc/rfc1760.txt)

[8] RFC 1321 - The MD5 Message-Digest Algorithm (http://www.ietf.org/rfc/rfc1321.txt)

## ABOUT THE AUTHOR

PhD. Student Isaac Agudo, Department of Computer Systems, University of Malaga, E-mail: isaac@lcc.uma.es.