ISSN:2231-0711
www.ijcset.net

An Approach of Cryptography for Web User Authentication using Secure Remote Password Protocol

Revati Raman Dewangan,
MPCCET, Bhilai (CG)
revati2004@gmail.com

Vivek Parganiha,
MPCCET, Bhilai (CG)
vivekparganiha@gmail.com

Deepali Thombre,
SSCET, Bhilai (CG)
deepthombre@gmail.com

***Abstract* - This research paper describes generation of a crypt key for user of web application using SRP techniques for the purpose of secure authentication in web. A numbers of web sites offer different kinds of users in world wide to access web application using unique user name and corresponding a password for securing them to others, even though they are now hacked by professional hackers. To avoid this kind of hacking of user's accounts; our approach is to provide a secure cryptography key using the techniques SRP (SRP-6) along with their username and password. This key will be unique for a particular user. Whenever user attempts to login the web application a new unique key will be generated by the application in each an every single attempt then the newly generated key will be validated by server side.**

In many web applications, it is desirable to have users log in by giving some unique login name and a password before accessing pages. There are many ways to implement this, each with different advantages and disadvantages. The considerations involved are complex enough and the majority of authentication systems in use on the web today have at least some fixable security weaknesses. There are two standard authentication systems which are described in the HTTP protocol documents: "basic authentication" which is supported by most browsers and HTTP servers, and "digest authentication" which isn't. The Secure Remote Password (SRP) protocol is an implementation of a public key exchange handshake described in the Internet standards working group request for comments 2945(RFC2945). This mechanism is suitable for negotiating secure connections using a user-supplied password, while eliminating the security problems traditionally associated with reusable passwords. This system also performs a secure key exchange in the process of authentication, allowing security layers (privacy and/or integrity protection) to be enabled during the session. Trusted key servers and certificate infrastructures are not required, and clients are not required to store or manage any long-term keys.

Index Terms— **Web Application authentication, RFC2945, HTTP server, Secure Remote Password (SRP) protocol, integrity protection.**

I. INTRODUCTION

In many web applications, it is desirable to have users log in by giving some unique login name and a password before accessing pages [1]. There are many ways to implement this, each with different advantages and disadvantages. The considerations involved are complex enough that I'd guess that the majority of authentication systems in use on the web today have at least some fixable security weaknesses. In the HTTP protocol documents: "basic authentication" which is supported by most browsers and HTTP servers, and "digest authentication" which isn't. I will then discuss various "do-it-yourself" alternatives to basic authentication, focusing on the three basic phases to the web authentication process:

1. **Logging in:** The user must be prompted for a login and password. Some program on the server must check these against a database to confirm that they are valid.
2. **User Tracking:** Normally there is no persistent connection between a user's browser and and your web server. If the web-site consists of more than one page, and if you don't want the user to have to log in again for each new page he looks at, we need some way to preserve the login information from page to page.
3. **Logging Off:** If we have a way to remember that a user is logged on, we also need a way to destroy that information when the user logs off.

Several commonly used server-side web development packages (such as Microsoft's Active Server Pages, Allaire's Cold Fusion, or Apache's Tomcat server) have authentication systems built in.

Authentication, authorization, and accounting (AAA) is a term for a framework for intelligently controlling access to computer resources, enforcing policies, auditing usage, and providing the information necessary to bill for services. These combined processes are considered important for effective network management and security.

As the first process, **authentication** provides a way of identifying a user, typically by having the user enter a valid user name and valid password before access is granted. The process of authentication is based on each user having a unique set of criteria for gaining access. The AAA server compares a user's authentication credentials with other user credentials stored in a database. If the credentials match, the user is granted access to the network. If the credentials are at variance, authentication fails and network access is denied.

Following authentication, a user must gain **authorization** for doing certain tasks. After logging into a system, for instance, the user may try to issue commands. The authorization process determines whether the user has the authority to issue such commands. Simply put, authorization is the process of enforcing policies: determining what types or qualities of activities, resources, or services a user is permitted. Usually, authorization occurs within the context of authentication. Once you have authenticated a user, they may be authorized for different types of access or activity.

The final plank in the AAA framework is **accounting**, which measures the resources a user consumes during access. This can include the amount of system time or the amount of data a user has sent and/or received during a session. Accounting is carried out by logging of session statistics and usage information and is used for authorization control, billing, trend analysis, resource utilization, and capacity planning activities.

Authentication, authorization, and accounting services are often provided by a dedicated AAA server, a program that performs these functions. A current standard by which network access servers interface with the AAA server is the Remote Authentication Dial-In User Service (RADIUS).

i. Basic Authentication

To use basic authentication, you must configure your HTTP server daemon to know that certain documents require authentication to access[2]. First, all documents to which access is to be restricted are placed in some common directory under your server's document root. That directory (and all beneath it) can be configured either by placing commands in a file named .htaccess that resides in that directory, or by placing the same commands in an appropriate <Directory> block in the global configuration file. The directives will be the same in either case, giving at least the following information:

- **Authorization Realm Name** - Some label which identifies which service this authorization is for.
- **User Database Name** - These describes where the database of valid users and user passwords is stored. (IMPORTANT: It should not be stored anywhere under the server's document root, since any data there could possibly be viewed by the user, and you don't want people viewing your password database).
- **Restricted Operation** - A list of which kinds of HTTP transactions authentication is required for.

ii. Digest Authentication

Digest authentication was added to the HTTP standard to provide a method of authenticating users without sending passwords over the network in clear text. This fixes the major security weakness in basic authentication.

Digest authentication, however, has only recently been beginning to catch on. Apache's web server has long included support for it, but until recently the only browser that implemented it was W3C's reference browser, Amaya. Now support for it has appeared in Internet Explorer 5.0, Mozilla 1.9.7, Netscape 7, Opera 4.0, and Safari 1.0. That's pretty much all current browsers. Microsoft's IIS 5.0 server also supports it.

For the most part, digest authentication works just like basic authentication. The browser requests a page, which is rejected. But the rejection message is a bit different, in that it says a digest authentication is required and also gives a string called a "nonce," which is some string (generally based on the time of day and the IP address of the requester) which is different for each request made.

As with basic authentication, the browser gets a password (either from the user or from its cache memory) Instead of just sending that information, the browser does the following:

1. Concatenates the user name, the authentication realm name and the password, and then computes an MD5 checksum of that whole string.
2. Concatenates the URL requested and the method for the request, and then computes an MD5 checksum of that string.
3. Concatenates the two previous checksums with the "nonce" string supplied by the server, and then computes an third MD5 checksum of *that* string.

The checksum resulting from the last step is sent with the request for the new page, as are the clear text of the login name and the nonce value.

Note* The MD5 Message-Digest Algorithm is a widely used cryptographic hash function that produces a 128-bit (16-byte) hash value. Specified in RFC 1321, MD5 has been employed in a wide variety of security applications, and is also commonly used to check data integrity.

II. CRYPTOGRAPHY

Cryptography is the science of *secret writing*. It's a branch of mathematics, part of *cryptology*. Cryptology has one other child, *cryptanalysis*, which is the science of breaking (analyzing) Cryptography [3].

The main security concerns of applications are addressed by cryptography. First, applications need assurance that users are who they say they are. Proving identity is called *authentication*. In the physical world, a driver's license is a kind of authentication. When you use a computer, you usually use a name and password to authenticate yourself. Cryptography provides stronger methods of authentication, called signatures and certificates[4].

Computer applications need to protect their data from unauthorized access. You don't want people snooping on your data (you want *confidentiality*), and you don't want someone changing data without your knowledge (you want to be assured of your data's *integrity*). Data stored on a disk, for example, may be vulnerable to being viewed or stolen. Data transmitted across a network is subject to all sorts of nefarious attacks. Again, cryptography provides solutions.

So what can you do with cryptography? Plenty. See Figure 1.

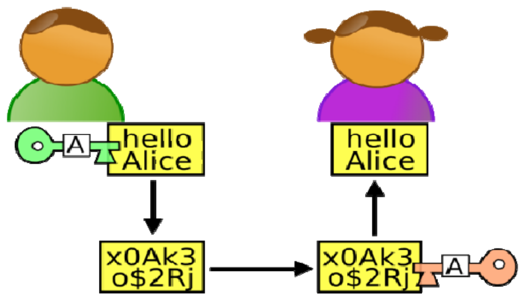

Figure 1. Cryptography

Here are just a few examples:

Secure network communications

Cryptography can protect your data from thieves and impostors. Most web browsers now support SSL , a cryptographic protocol that encrypts information before it is transmitted over the Internet. SSL allows you to buy things, using your credit card number, without worrying too much that the number will be stolen.

Secure hard disk

You can encrypt the files on your hard disk so that even if your enemies gain physical access to Your computer, they won't be able to access its data.

Secure email

Email is notoriously easy to steal and easy to forge. Cryptography can make it hard to forge e-mail and hard to read other people's messages. Although cryptography is heavily mathematical, there isn't much math in this book. One of the really nice things about the Java Security API is that, like any good software library, it hides a lot of complexity.

The Security API exposes concepts, like Signature and Cipher, and quietly deals with the underlying details. You can use cryptography effectively in a Java application without knowing too much about what's going on underneath the hood. Of course, this implies you need to trust Sun to write the Security API correctly.

III. Types of Cryptography

There are several ways of classifying cryptographic algorithms. For purposes of this paper, they will be categorized based on the number of keys that are employed for encryption and decryption, and further defined by their application and use [9]. The three types of algorithms that will be discussed are (Figure 2):

- Secret Key Cryptography (SKC): Uses a single key for both encryption and decryption
- Public Key Cryptography (PKC): Uses one key for encryption and another for decryption.
- Hash Functions: Uses a mathematical transformation to irreversibly "encrypt" information.

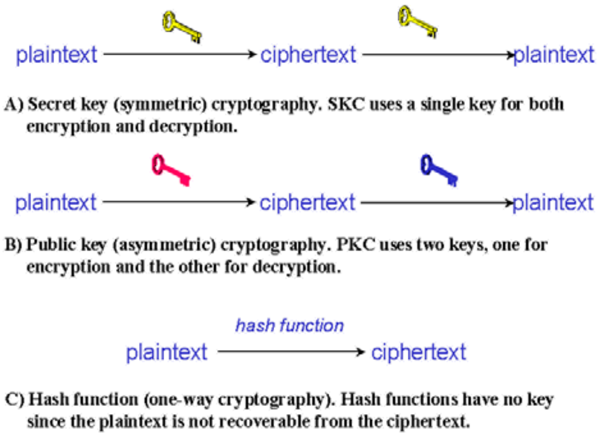
plaintext
ciphertext
plaintext
A) Secret key (symmetric) cryptography. SKC uses a single key for both encryption and decryption.
plaintext
ciphertext
plaintext
B) Public key (asymmetric) cryptography. PKC uses two keys, one for encryption and the other for decryption.
hash function
plaintext
ciphertext
C) Hash function (one-way cryptography). Hash functions have no key since the plaintext is not recoverable from the ciphertext.

Figure 2. Types of cryptography: secret-key, public key, and hash function.

1) *Secret Key Cryptography*

With *secret key cryptography*, a single key is used for both encryption and decryption. As shown in Figure 2 A, the sender uses the key (or some set of rules) to encrypt the plaintext and sends the ciphertext to the receiver. The receiver applies the same key (or ruleset) to decrypt the message and recover the plaintext. Because a single key is used for both functions, secret key cryptography is also called *symmetric encryption*.

Secret key cryptography schemes are generally categorized as being either *stream ciphers* or *block ciphers*. Stream ciphers operate on a single bit (byte or computer word) at a time and implement some form of feedback mechanism so that the key is constantly changing. A block cipher is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

2) *Public-Key Cryptography*

Public-key cryptography has been said to be the most significant new development in cryptography in the last 300-400 years. Modern PKC was first described publicly by Stanford University professor Martin Hellman and graduate student Whitfield Diffie in 1976. Their paper described a two-key crypto system in which two parties could engage in a secure communication over a non-secure communications channel without having to share a secret key.

PKC depends upon the existence of so-called *one-way functions*, or mathematical functions that are easy to computer whereas their inverse function is relatively difficult to compute. Let me give you two simple examples:

1. *Multiplication vs. factorization:* Suppose I tell you that I have two numbers, 9 and 16, and that I want to calculate the product; it should take almost no time to calculate the product, 144. Suppose instead that I tell you that I have a number, 144, and I need you tell me which pair of integers I multiplied together to obtain that number. You will eventually come up with the solution but whereas calculating the product took milliseconds, factoring will take longer because you first need to find the 8 pairs of integer factors and then determine which one is the correct pair.
2. *Exponentiation vs. logarithms:* Suppose I tell you that I want to take the number 3 to the 6th power; again, it is easy to calculate 3^6=729. But if I tell you that I have the number 729 and want you to tell me the two integers that I used, *x* and *y* so that $\log_x 729 = y$, it will take you longer to find all possible solutions and select the pair that I used.

3) Hash Functions

Hash functions, also called *message digests* and *one-way encryption*, are algorithms that, in some sense, use no key (Figure 2C). Instead, a fixed-length hash value is computed based upon the plaintext that makes it impossible for either the contents or length of the plaintext to be recovered. Hash algorithms are typically used to provide a *digital fingerprint* of a file's contents, often used to ensure that the file has not been altered by an intruder or virus. Hash functions are also commonly employed by many operating systems to encrypt passwords. Hash functions, then, provide a measure of the integrity of a file.

Hash algorithms that are in common use today include:

- *Message Digest (MD) algorithms:* A series of byte-oriented algorithms that produce a 128-bit hash value from an arbitrary-length message.
    - *MD2 (RFC 1319):* Designed for systems with limited memory, such as smart cards. (MD2 has been relegated to historical status, per RFC 6149.)
    - *MD4 (RFC 1320):* Developed by Rivest, similar to MD2 but designed specifically for fast processing in software. (MD4 has been relegated to historical status, per RFC 6150.)
    - *MD5 (RFC 1321):* Also developed by Rivest after potential weaknesses were reported in MD4; this scheme is similar to MD4 but is slower because more manipulation is made to the original data. MD5 has been implemented in a large number of products although several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996 ("Cryptanalysis of MD5 Compress").
- *Secure Hash Algorithm (SHA):* Algorithm for NIST's Secure Hash Standard (SHS). SHA-1 produces a 160-bit hash value and was originally published as FIPS 180-1 and RFC 3174. FIPS 180-2 (aka SHA-2) describes five algorithms in the SHS: SHA-1 plus SHA-224, SHA-256, SHA-384, and SHA-512 which can produce hash values that are 224, 256, 384, or 512 bits in length, respectively. SHA-224, -256, -384, and -512 are also described in RFC 4634.
- *RIPEMD:* A series of message digests that initially came from the RIPE (RACE Integrity Primitives Evaluation) project. RIPEMD-160 was designed by Hans Dobbertin, Antoon Bosselaers, and Bart Preneel, and optimized for 32-bit processors to replace the then-current 128-bit hash functions. Other versions include RIPEMD-256, RIPEMD-320, and RIPEMD-128.
- *HAVAL (HAsh of VAriable Length):* Designed by Y. Zheng, J. Pieprzyk and J. Seberry, a hash algorithm with many levels of security. HAVAL can create hash values that are 128, 160, 192, 224, or 256 bits in length.
- *Whirlpool:* A relatively new hash function, designed by V. Rijmen and P.S.L.M. Barreto. Whirlpool operates on messages less than 2^{256} bits in length, and produces a message digest of 512 bits. The design of this has function is very different than that of MD5 and SHA-1, making it immune to the same attacks as on those hashes (see below).
- *Tiger:* Designed by Ross Anderson and Eli Biham, Tiger is designed to be secure, run efficiently on 64-bit processors, and easily replace MD4, MD5, SHA and SHA-1 in other applications. Tiger/192 produces a 192-bit output and is compatible with 64-bit architectures; Tiger/128 and Tiger/160 produce a hash of length 128 and 160 bits, respectively, to provide compatibility with the other hash functions mentioned above.
-

IV. SECURE REMOTE PASSWORD PROTOCOL.

Secure Remote Password (SRP) is an ingenious authentication method where the password remains private to the user at all times and never has to be communicated beyond their computer; instead, what client and server exchange is a series of cryptographically secured messages. The **S**ecure **R**emote **P**assword protocol performs secure remote authentication of short human memorizable passwords and resists both passive and active network attacks. Because SRP offers this unique combination of password security, user convenience, and freedom from restrictive licenses, it is the most widely standardized protocol of its type, and as a result is being used by organizations both large and small, commercial and open-source, to secure nearly every type of human-authenticated network traffic on a variety of computing platforms.

Merits of the SRP protocol:

- Zero-knowledge password proof - the password remains private to the user at all times and is never shared with the authenticating server.
- Resistant to eavesdropping and man-in-the-middle attacks.

- Good resistance to offline dictionary attacks in case the server is compromised.
- May be used for mutual authentication and to establish a secret session key for encrypted communication.
- A mutually trusted third party is not required.

The Secure Remote Password protocol was devised by Tom Wu[7] during his work at Stanford University.

What is SRP?

SRP is a secure password-based authentication and key-exchange protocol. It solves the problem of authenticating clients to servers securely, in cases where the user of the client software must memorize a small secret (like a password) and carries no other secret information, and where the server carries a *verifier* for each user, which allows it to authenticate the client but which, if compromised, would not allow the attacker to impersonate the client. In addition, SRP exchanges a cryptographically-strong secret as a byproduct of successful authentication, which enables the two parties to communicate securely.

Many password authentication solutions claim to solve this exact problem, and new ones are constantly being proposed. Although one can claim security by devising a protocol that avoids sending the plaintext password unencrypted, it is much more difficult to devise a protocol that remains secure when:

- Attackers have complete knowledge of the protocol.
- Attackers have access to a large dictionary of commonly used passwords.
- Attackers can eavesdrop on all communications between client and server.
- Attackers can intercept, modify, and forge arbitrary messages between client and server.
- A mutually trusted third party is not available.

The idea behind SRP first appeared on USENET in late 1996, and subsequent discussion led to refined proposals in 1997 to address these security properties. This lead to the development of one of the variants of the protocol still in use today, known as SRP-3, which was published in 1998 after several rounds of discussion and refinement on cryptography-related newsgroups and mailing lists, and has withstood considerable public analysis and scrutiny since then. The technology evolved into a newer variant known as SRP-6, which maintains the security of SRP-3 but has refinements that make it more flexible and easier to incorporate into existing systems. Technical details of the actual protocol design are available from this site [10].

Competitive Analysis of SRP

This section lists some of the more popular authentication products and analyzes how their security compares to strong password mechanisms like SRP and, in some cases, how SRP can be used to add password security to existing infrastructures.

1.Multifactor Authentication

The strongest forms of authentication involve the combination of more than one authentication factor:

- What you know: Passwords, passphrases
- What you have: Hardware tokens, private keys
- What you are: Biometrics

When combined properly, these techniques force an intruder to compromise several factors before being able to mount a meaningful attack.

Cryptographic smart cards - These are physical tokens that contain a CPU and enough memory to store private keys and perform cryptographic operations like digital signatures with them. They are usually PIN-protected and offer some form of hardware-based tamper-resistance, which is supposed to make them useless without the human-memorized PIN.

Under the right circumstances, they offer a high degree of security. They are also expensive to issue, require the installation of special card readers, and are difficult to deploy on a large scale, which has been an impediment to their adoption, especially in the United States.

Arcot authentication - Arcot Systems offers a software-based alternative to hardware authentication tokens that bypasses many of the limitations faced by smart cards while maintaining the security properties of a multifactor system. By using a technique known as *cryptographic camouflage*, Arcot can store tokens entirely in software and protect them from brute force attack, a problem that plagues other software-based solutions.

2.Strong Password Authentication

Although most strong password systems are single-factor systems, they can be combined with an additional factor, like a software or hardware token, to construct a multifactor system. What distinguishes strong password systems from other, weaker one-factor methods is the level of security that they leverage from that one factor. A strong password system protects even low-entropy ("guessable") passwords from off-line attack, even against adversaries with complete access to the network. They also exchange a session key to enable both data confidentiality and integrity after authentication has been confirmed.

EKE, SPEKE - EKE, or **E**ncrypted **K**ey **E**xchange, was developed by Bellovin & Merritt in 1992, and is one of the earliest examples of secure password technology. David Jablon invented SPEKE (**S**trong **P**assword **E**xponential **K**ey **E**xchange) in 1996, as well as a variable-modulus variant of SPEKE while some people also refer to as PDM. Both EKE and SPEKE passwords that are "plaintext-equivalent" to the real password, which means that an intruder who breaks into a server protected by EKE or SPEKE and captures the password database would subsequently be able to impersonate all the users on the system. EKE and SPEKE have variants that guard against this attack, but only with a significant performance loss.

AMP, SNAPI, AuthA, OKE, etc. - The current groundswell of interest in password authentication technology has spurred a large number of proposals for new authentication protocols, all offering different combinations of security, performance, and license availability. Standards bodies like the IEEE P1363 working group have formed entire study groups to help sort out the veritable alphabet soup that has resulted.

3.Pseudo-Strong Authentication

This category of methods is called "pseudo-strong" because while they are better than plaintext passwords, they have also have some well-known security problems that make them vulnerable in real-world deployments. Another distinguishing characteristic of these methods is that they inhabit the "no-man's land" of the technology quadrant: There are methods out there that offer better security while being equally easy to use, and there are also methods out there that are equally strong yet easier to use. It should therefore come as no surprise that SRP-based variants of these methods are quickly coming to market, since they preserve the ease-of-use that makes products based on these methods easy to deploy and use, while offering better security against well-known attacks.

SSH Public Key Authentication - SSH, or "Secure Shell", is a protocol that uses various key exchange methods to encrypt session traffic for remote logins and TCP/IP port forwarding. Although it uses well-known algorithms to perform both session key establishment and session encryption, its security is heavily dependent on the security of the method used to authenticate the user. In many cases, the user authentication method is the weak link in the chain.

SSH Public Key Authentication is similar to Client-side SSL Certificate Authentication, as it shares the same vulnerability to a stolen-credential attack, and it also has the same difficulties in coping with *roaming users*, who frequently must log in from different locations. Some implementations of SSH allow the private key to be stored in hardware, which improves security in exchange for more deployment obstacles.

SSH Password Authentication - Nearly all implementations of SSH support what is known as "Password Authentication", in which the client sends its password directly to the server, hoping that the encrypted connection will protect it in transit. The security of this method is highly dependent on the particular method of server authentication being used. Keep in mind that an attacker who successfully bypasses server authentication in this case gets the *plaintext* passwords of any users who subsequently attempts to log in, and can do so undetectably.

One commonly-used method of server authentication is "ad-hoc" distribution of server public keys. The server sends the client its (non-certified) public host key, and the client uses this to encrypt a session key and send it to the server. The actual protocol is slightly more complicated, but what matters is that the client has no way of knowing if the host public key it received was in fact the right one, which makes this protocol susceptible to Man In The Middle (MITM) attacks in practice. Although the client tries to keep track of previously-received host keys, there is no way for it to know if a change in host keys is legitimate or the beginning of an attack, so users will in most cases either ignore the warning that SSH spits out or inundate the help desk (if there is one) every time a host key expires or the network configuration changes for any reason.

Some versions of SSH support PKI-based distribution of host keys, which improves the security of server authentication, but then requires the deployment of a PKI. Sites must then purchase server certificates from third parties like VeriSign or Thawte, or they must install and administer their own certification authorities. In any case, strengthening server authentication doesn't address the real problem with password authentication, which is that the user's password is being sent out in a reversibly-encrypted form.

Password-protected Client-side SSL/TLS Certificates - To initiate a conventional, server-authenticated SSL connection, a Web server provides a certificate from a well-known CA (Certificate Authority), which the Web browser can verify. SSL also provides a mode in which the client can send a certificate to the Web server, which can verify it and use its contents to authenticate the client. The corresponding private key, which resides on the user's PC, is protected with a passphrase. An attacker who captures this encrypted private key can brute-force the passphrase to obtain the private key and impersonate the user, which is the same problem that affects SSH Public Key Authentication. Likewise, it is possible to store these private keys in hardware tokens, which trades off security for convenience.

Default Preauthentication in Kerberos V5 - To fix the password security weaknesses in Kerberos V4, version 5 added *preauthentication*, which forces a client to prove knowledge of his password before the server starts an authentication session. The default form of preauthentication Kerberos V5 is an encrypted timestamp: The user converts his password into an encryption key, which the encrypts a binary representation of the current time. If the server is able to decrypt the client's message and obtain a timestamp within a given window, authentication proceeds. Unfortunately, an attacker who intercepts this message can perform an off-line brute-force attack against this message and obtain the user's password.

4.Weak Authentication

Although it would be nice to be able to say that weak authentication methods are mentioned here for historical interest, the truth is that an embarrassingly large amount of Internet traffic is secured with these legacy technologies. In some cases, there is a legitimate reason for making these choices - a common reason is that the

authentication must work without installing additional client software.
Unfortunately, users of these systems ultimately sacrifice security for the convenience of backward-compatibility. If it is necessary to settle for the lowest common denominator to satisfy all users, then it is in all users' best interests to raise that lowest common denominator to support a level of security that includes strong password authentication. The "worst offenders", in no particular order:

- Clear text passwords (unsecured telnet, rlogin)
- Encoded passwords (HTTP Basic Authentication)
- Classic challenge-response protocols (HTTP Digest Authentication, Windows NTLM Authentication, APOP, CRAM, CHAP, etc.)
- One-Time Password schemes based on a memorizable secret (S/Key, OPIE)
- Kerberos V4.

V. Concept of SRP for securing the connection

Goals: To secure against [11]
1. Passive adversary
2. Active adversary (they cannot recover the password)
3. Adversary cannot do an offline dictionary attack to recover password.
4. Adversary with access to server's DB cannot recover the password.

How are passwords stored inside a computer? See Table 1.

Table 1 How are password Stored inside a computer.

Id, pass →	Id	Pass
	Rev	Hash(pass)
	Alice	Hash(pass)

But if we assume there are 1,000,000 distinct password combinations, then the attacker with access to a computer, can create a hash of all the password's and use it to attack this computer. Worse yet, he can attack any other computer which uses the same hash function, like same OS.
Solution for the above problem:
Use a random salt (128-bit number) in addition to the hash. Having a user-wise Salt, will require an attacker to generate a per-user/per-machine dictionary attack to crack the password. See Table 2.

Table 2 Using a salt

Id, pass	Id	Pass
	Rev	S,Hash(S,pass)
→	Alice	S,Hash(S,pass)

Secure Remote Password Protocol

- Global Parameters

I. N – large prime
II. Base g
III. Parameter k = 3 (for which discrete log is not known) see Table 3 and Table 4

Table 3. Registration Phase

User	Client	Server

Id,pass →	Pick a random S.
	X=Hash(S,Pass)
	V= g^x mod N →
		id,S,V
		
		ID | Pass
		Id | S,V

Table 4 Login Phase

User	Client	Server

Id,pass →	Pick a random 'a'
	A = g^a mod N →
	X= H(s,Pass)
	u= H(a,b)
	S=$(\beta - k.g^x)^{(a+ux)}$
	K=H(S)
	←
		Id, A
		pick a random 'b'
		Lookup id from table and compute
		B = kV+g^b mod N
		s, B
		S=$(\alpha.v^u)^b$
		K=H(S)

An Example wrong mutual authentication phase sees Table 5

Table 5 wrong mutual authentication

User	Client	Server
	← Hello,	MAC(K, "Hello")

Problem: attacker who just knows id, can send (id, α) and can do the offline dictionary attack on the "Hello" message, verifying MAC.The actual problem is server authenticates first after the shared key.
Solution: Have client authenticate first with the shared key. See table 6. Now they share a key K, which can be used for an IND-CCA2 encryption scheme. How Keys Match?see Table 7

Adversary Advantages and Attempt to Break

1) Attacker impersonating as client
In the "s , β" step, he needs to solve discrete log and get the random number b. But as β is randomly distributed, he needs to check with the server by sending M1, if it is correct. He has

only one chance to guess the password and the hash and verify in a single interaction. He cannot do an offline dictionary attack.

Table 6 Client authentication

USER	CLIENT	SERVER
	M1= H(H(ID)||S||	A||B||K) $\longrightarrow$
	$\longleftarrow$ M2=	H(A||M1||K)

Table 7 Key Match

$S_{client} = (\beta - k.g^{x})^{(a+ux)}$ $= (KV + g^{b} - KV)^{(a+ux)}$ $= g^{ab}\, g^{ubx}$
$S_{server} = (\alpha.V^{u})^{(b)}$ $= (g^{a}.g^{xu})^{(b)}$ $= g^{ab}\, g^{ubx}$

2) Attacker impersonating as Server

The attacker can send a guess of V as "s , β". (The salt s can be obtained by impersonating as client from the actual server).Now, Still he has only one guess to generate V, as there is only one chance for verifying M1

Table 8 Attacker impersonating as Server

$S_{client} = (\beta - k.g^{x1})^{(a+ux)}$ $= (Kg^{x1} + g^{b} - KV)^{(a+ux)}$ $= g^{ab}\, g^{ubx1}$
$S_{server} = (\alpha.V^{u})^{(b)}$ $= (g^{a}.g^{x1u})^{(b)}$ $= g^{ab}\, g^{ubx1}$

3) Man-In-Middle attack

Say the Attacker is acting as MIM and intercepting communication and impersonating the client to server and server to client, then he either need to guess a from α or guess b from β. Both of this requires solving discrete log problem.

Note * This was the over all concept of the SRP.

VI. CONCLUSION

Till the end of the chapter V we discussed the various aspect and different scenario of security, security threats and solution goal of security. We also discussed the technology to avoid the hacking by professional hacker. It was our literature paper, in our next paper we are going to implement a web based application, In which the newly key will generated each attempt using cryptography with SRP method and this key along with the password will validated by the server side application.

REFERENCES

[1]http://tldp.org/HOWTO/Secure-Programs-HOWTO/web-authentication.html

[2] http://unixpapa.com/auth/

[3] http://www.cacr.math.uwaterloo.ca/hac/

[4] C. ADAMS AND H. MEIJER, "Security related comm.- ents regarding McEliece's public-key cryptosystem", *Advan-ces in Cryptology–CRYPTO '87 (LNCS 293)*, 224–228, 1988.

[5] S. BERKOVITS, "How to broadcast a secret", *Advances in Cryptology–EUROCRYPT '91 (LNCS 547)*, 535–541,1991.

[6] N. Haller and R. Atkinson. On Internet Authentication. Naval Research Laboratory, October 1994. Request For Comments (RFC) 1704.

[7] The Secure Remote Password Protocol, Thomas Wu, Computer Science Department Stanford University.

[8] R.H. Morris and K. Thompson. Unix password security. Communications of the ACM,22(11):594, November 1979.

[9] *www.giac.org/cissp-papers/57.pdf*

[10] http://software.dzhuvinov.com/nimbus-srp.html

[11] M. Steiner, G. Tsudik, and M. Waidner. Refinement and extension of encrypted key exchange. ACM Operating systems Review, 29(3), July 1995

[12] The law of cryptography with java code by Neal R. Wagner.

[13] A. Menezes and S.A. Vanstone. Elliptic curve cryptosystems and their implementations. Journal of Cryptology, 6(4):209{224, 1993.

[14] N. Haller and R. Atkinson. On Internet Authentication. Naval Research Laboratory, October 1994. Request For Comments (RFC) 1704.

[15] S.C. Pohling and M.E. Hellman. An improved algorithm for computing logarithms in gf(p) and its cryptographic signi_cance. IEEE Transactions on Information Theory, 24(1):106{111, January 1978.

[16] security engineering: a guide to building dependable distributed systems by *zhqm zmgm zmfm* g. julius caesar *xyawo gaooa gpemo hpqcw ipnlg rpixl txloa nnycs yxboy mnbin yobty qynai* john f. kennedy

[17] Lecture Notes on Cryptography by Shafi Goldwasser and Mihir Bellare.