ONE IMPLEMENTATION OF A PROTOCOL FOR GENERATION AND DISTRIBUTION OF CRYPTOGRAPHIC KEYS

Dalibor Marijančević*,
Saša Adamović,
Milan Milosavljević

Singidunum University,
Belgrade, Serbia

Abstract:
This paper discusses a cryptographic protocol for a secure session key generation, whose design is based on the principles of perfect cipher systems. The need for this type of protocol in modern computer networks arises for several reasons. The most important ones are associated with a requirement for high-level security and a simple implementation (important cryptographic steps are executed in a controlled environment). The protocol is described in detail, as is its implementation. The paper further examines the protocol's advantages, limitations, as well as its applications.

Keywords:
key, generation, protocol.

1. INTRODUCTION

Perfect forward secrecy - PFS is an imperative for secure communication. The essence of PFS access is the session key [1]. The usual scheme through which the session key is generated is the application of asymmetric cryptography. However, since we do not know what the future brings and whether at some point the factoring of large primes on which asymmetric cryptography relies will become trivial, it is necessary to find a new way to exchange session keys.

Interesting is the fact that the present is the future of the past, or what was impossible to attack 15 years ago, today it can be simply feasible [2]. The attacker can be passive, record communication, and wait for a technology to emerge that will enable them to decrypt the content.

What is described above justifies the need for a protocol that will provide a strong session key that does not rely on the assumption that there is no feasible attack. This paper will propose a scheme providing the specified conditions.

2. OVERVIEW IN THE FIELD OF RESEARCH

Among protocols that employ the password as part of the key material the most widely used is the SRP - Secure Remote Password protocol [3].

Correspondence:
Dalibor Marijančević

e-mail:
dalibor.marijancevic.11@singimail.rs

DOI: 10.15308/Sinteza-2018-9-16

The SRP protocol was rolled out as one implementations of AKE (Asymmetric Key Exchange)[4]. Considering the then existing Encrypted Key Exchange (EKE) platform [5], the author concluded that this is a symmetrical protocol, which means that the same element needs to be kept in two places. This leads to the situation where both parties must be responsible for the exchange and prevention of unauthorized access.

Using this approach, he author concluded that it would be a better solution if both parties in the communication derive the secret, and then passed it as an argument to a hash function in order to generate a verifier that would then be forwarded to another participant. The verifier needs to be safely guarded against the attacker in order to prevent the dictionary attack. However, even in the case of compromise, a password that corresponds to a given verifier is still required. Such a scheme allows the password never to leave the user's environment, meaning it is not distributed via a communication channel.

3. THEORETICAL ELEMENTS

In analyzing the SRP protocol and other similar solutions which use a previously shared secret, we have determined that all those come with certain limitations, or weak points that can affect the security of the key exchange.

One weakness - the user's password - has already been addressed in by the existing implementations.

Another weakness becomes evident during the process of random number generation. Even though random number generation is executed on the server for the most part, some elements of the process take place in the app on the user's end. With that in mind, we have identified a need for a protocol that would not delegate any part of the process to the user.

The next problem relates to the reliability of asymmetric cryptography. Although there is still no cracking algorithm, it has not been proven that such an algorithm does not exist. As in the previous case, the need arises for a scheme, that will eliminate the use of such a calculation method for the session key. In asymmetric cryptography, there is definitely question of extensive CPU time which is needed for such operations. It should be noted, however, that protocols based on these operations can be used as a message-protecting shield, so that at an early stage of the protocol execution it can be determined whether an active attack has occurred.

The protocol or rather its simulation, which served as a demo is a protocol with playing cards [6].

Playing Cards Protocol

Protocol with playing cards is a protocol that uses four cards, two pairs of the same rank each. For simplicity, we will use the King and the Queen. The requirement is that paired cards are of different colors. Cards are shared secretly, but the exchange of key information is public.

The protocol begins with the dealing of the cards. The protocol assumes that the cards are dealt by an individual trusted by all participants. Cards are even dealt to the attacker. When cards are dealt legitimate participants publicly reveal the colors of their cards. If the colors match, it means that both legitimate players have different cards. Before the protocol is run, the rule is established that if the first participant holds a Queen, the key bit is zero, otherwise it is one.

If the colors differ, legitimate participants cannot make any definitive conclusions about the cards, while the attacker faces this uncertainty every time the colors of the cards held by legitimate participants match. In case of mismatch, cards are folded and the new round begins.

Interpretation of Cards in Bits

In order to use this protocol, cards should be presented as bits. One of possible implementations is a bigram where one bit denotes a color, while the other represents the king or queen. In this way, a set of ten bits represents five cards.

Card Dealing

The next obstacle is the very act of dealing of the cards. Since the assumption of each protocol is that the only secret value is the key, the situation presented in the protocol is not possible in a real-life scenario. From this perspective, it is obvious that the attacker gets all the necessary data through the communication channel as the cards are dealt. In addition, the protocol assumes that there is enough knowledge of one card so that it can be decided whether it will be used and what is the value of the key bit if a round is taken into account. For this reason, we will introduce some minor changes to the protocol.

The first change refers to a trusted person who deals the cards. This, in our case, will be a server. Assuming that the server has access to a source with good entropy, in this way, we get cards generated in a random way.

The following change refers to the dealing of the cards itself. At this point a user password comes in, or any secret value shared between the server and the user before the communication commences. Although the server has assumed the role of the dealer, it will generate its own cards only. The most important role of these server cards is that they represent random values, which will only amplify the entropy of the key. The user's cards represent a previously shared secret. The server sends its cards through the communication channel, while both participants already have their users' cards.

The last change pertains to the announcement of colors. This is no longer necessary, since both legitimate participants already have both sets of cards on themselves. The attacker knows what the server cards are, but since they do not know the users' cards and the public announcement of color has been removed from the protocol, there is no way for the attacker to find out how the key will be formed.

Protocol Setup

Taking into account all the points described above, we can discuss the setup of the protocol.

- Legitimate participants exchange secrets before commencing communication.
- In every communication, the server will generate its cards in a random way.
- A previously shared secret constitutes users cards .
- Only server cards are sent via the communication channel.
- Both participants generate the key in their environment.

By generating a session key in this way, we have avoided the following:

- Sending previously exchanged secret via a communication channel.
- The user has no obligation to generate random values that are part of the key material.
- User's password as a material for the key. Server cards, assuming they use the appropriate source, increase the key's entropy.

4. PROTOCOL IMPLEMENTATION

For the implementation of the protocol, it is necessary to consider primarily the keeping of a previously exchanged secret, as well as the establishing of communication. Since the primary use of the protocol is to generate a secret session key between the server and the client, or the client application, we will direct our attention to such a scenario.

The server side in all scenarios may remain unchanged, while on the client side this may be a web browser or a native application. Although web browser is also a native application, the mode it is used for communication with the server is in most cases different. The main difference is data storage.

When we use the web browser, the only memory we can use comes in form of cookies or local storage. This places a limit on the available options, especially when the client first reaches out with a request. The server has no way of knowing who the user is and it must very carefully consider if it will respond. After the first successful session, a cookie can be stored in the user's web browser, and the next time the user requests access, the server will know who the client is based on the cookie. However, cookies may be time-limited. Furthermore, web browsers support private sessions, after which all communication data is deleted. Finally, the user can delete a cookie.

In the instance of OS access, a native application could integrate parameters necessary for the protocol as part of the installation. In this case, a previously shared secret can be used as a key that the user does not need to memorize. This, from the standpoint of entropy, will serve better for the session key than the user's password. From this aspect, a native application is considered a better solution, but often it is impossible to use them because the user still has to install them, and a certain number of users will not accept that. Another aspect that can reduce the level of security is the implementation itself. The scheme in which a secret is kept on the client's operating system must be carefully considered. If it is stored as open text, malicious software can acquire sufficient privileges and reach the key itself.

Secret Keeping

Here we will primarily discuss how a secret is kept on the server's end. The secret is kept on the server with a username and a salt. The salt is necessary to prevent password disclosure in case a database with user data has been compromised. A salt is generated specifically for each user. The password and salt are passed together as an argument of a hash function. For this process, slow algorithms should be used so that the time needed for negotiation is extended.

The system can be additionally secured if the salt and the result of a hash function are stored in different databases. The result is a previously shared secret and in this way we have secured the user's cards. Now we may face a problem that the user does not know the salt. Since the salt is not a key, it can be public information. However, some are of the opinion that the salt should not be public information, or that the system will be safer if the salt is not public. A compromise is that the server should send salt only if the user does not have it. This would minimize the risk of revealing the salt.

Use of Shared Secret and Protocol Packet

Let us assume that SHA-512 algorithm is used for the hash function. The result of this algorithm is 64 bytes. Since the protocol defines cards as a pair of bits, each of these bytes will be converted into bits and merged into one string. The length of that string N is an important parameter of the protocol, since it essentially represents the number of cards the user has.

To prevent the attacker from getting the information about the length of the string, each packet will have 512 bits. The server will always generate cards within the length of 512 bits, but the protocol will only use the first N bits. This feature will be further used to send parameters in the first packet which will additionally increase the uncertainty of the session key.

It is important to note that that each packet of the protocol sent by the server is assembled from random bits, and then, if necessary, some bits change depending on the needs in the current phase of the protocol.

Protocol Parameters

In order to further add to the uncertainty of the current session key, we will introduce several parameters. The server sends the protocol parameters in the first packet of cards by placing them in positions that are immediately after the position representing the length of the user's cards.

The first parameter is the rule on whether the key bit is zero or one depending on which card the server or user has. We will generate this rule as a random one bit value that will determine whether the king represents zero or one.

The next parameter is the key length. AES (Advanced Encryption Standard) algorithm is mostly used as an algorithm to encrypt communication with a session key. Bearing in mind that AES supports 128, 160, and 256-bit keys, a minimum of two bits representing this parameter is required, where numbers 0, 1, and 2 in the binary presentation would represent the key length accordingly.

The third parameter refers to the number of rounds. If participants in the communication want to hide the information on how many rounds is necessary to generate the key, a third parameter can also be added, which will represent the number of packages to be sent after key generation, but not considered. This can be achieved by setting a rule to always send the same number of packets, but this could be considered an unnecessary cost for smaller keys.

Initial Packet and Salt

We have already mentioned that the salt can be public information, so the server can send a salt to the client in its first response. However, if we do not want to display it in any subsequent key exchange, when the client sends a request to the server next time, the server must know that the salt is already saved. For this purpose, the client, depending on the current state, will generate one or two random values, one as their challenge to the server and the other one as a replacement for a salt if necessary. We should bear in mind that one of the principles of the protocol is that the client should not generate random values that are used as key material, so we will not use them for these purposes.

The first situation is that in which the client does not have a salt. In this case, it generates two random values and passes them as an argument to a hash function. The first packet will be a user name, a challenge and the result of a hash function. When a server receives a message, it will pass the challenge and the salt associated with the username as the arguments to hash function. Since the result will be different, that means that the client does not have a saved salt on their end. Thus, the first response from the server side on its initial bits will contain a salt and will not be used as a cards material.

The other situation is that the client has a salt. Then, on the client's end, one random value is sufficient, which will be a challenge, and the result of a hash function will be obtained when the challenge and salt are used as arguments. In this way, the server will receive information that the client has a salt and the first package will be the beginning of dealing the cards.

The challenge sent by the client is used for mutual authentication that should verify whether the key is properly generated.

File Storage Service

When we considered the weaknesses of asymmetric cryptography, we indicated that what is currently impenetrable, in the future, with the growth of processing power or the application of some new algorithms, can become solvable. A passive attacker can record communication and wait for something to be developed for a feasible attack.

Such a scenario proposes a new use for the protocol. In the same manner that an attacker can record communication, the server can record encrypted files. In this case, the server could have the same information as the attacker, that is, it would only know its cards. The previously shared secret would only be used for authentication and communication, and another one could be used for encryption, but it would be known only to the user. The role service would have is to provide a source with good entropy in the form of its cards and send it to the client. The server, together with the files, would also store the appropriate cards. Everything else would take place on the client's end.

This setup poses a problem of salt as an asset that increases the uncertainty of the password. The client can use the same salt as the one used for key exchange, but that is in conflict with the rule that a special salt is used for each password, which is in place because the user might not be cautious enough and may enter the same password as the one used for key exchange. We can infer from this that both the salt used for key exchange and the salt for encryption should be stored on the server. This places responsibility on the user not to use the same password for file encryption, if they do not want to allow the server to have insight into the key. Under such circumstances, the server would be in the same position as the attacker who came into possession of the database with the user's data, that is the salt and the result of the selected hash function. Another way is that the server generates a salt which would be stored on the client's end. However, users expect to be able to access their files from different platforms, so this can be a problem from the aspect of user experience.

Protocol Flow

Bearing in mind the foregoing points, the flow of the protocol is the following:

- H - selected hash function
- R – client's challenge
- k – username or any other unique identifier
- t – a secret
- s – salt
- s' – replacement for the salt
- n – number of rounds, which depends on the currently held cards and the third parameter of the protocol
- K – session key
- E, D – encryption, decryption with the selected algorithm with symmetric key
- r - unused bits of server cards
- Rs – server's challenge

Client:

– calculates H(R, s) or H(R, s')

-- >(k, H(R, s) / H(R, s'), i)

Server:

– generates first cards and defines protocol parameters

– calculates H(i, s(k)) and compares the result with what was sent by the client

– if the comparison is positive, calculates the client cards and at the specified place it replaces the bits with parameters

< -- packet(cards, parameters)

– if the comparison is negative, replaces the bits with salt

< -- packet(cards(salt))

Client:

- depending on the state calculates salt or receives the first packet and begins calculating the key

Server:

< -- packet(cards) * n

Authentication:

Server:

<-- E(K, H(R, r), Rs)

Client:

- D(E(K, H(R, r), Rs))

-- >E(K, H(Rs))

Independent Use of the Protocol

Looking at the protocol flow, we see that the user presents himself and sends a challenge. All other messages are sent by the server. If the PKI infrastructure is not used together with the protocol, an attacker may step in between, alter messages and send their bits, but eventually they must send a message encrypted with a session key, and this will not be possible.

In case that someone tried to misrepresent the server, they would be able to get a salt, and only if the web browser is used as a client application. Today's web services use different ways to determine if something unusual happened in a given session. If the behavior deviates from the norm, the user will be prompted to change the password, and will receive a new salt in the process. However, even if such an attack went unnoticed, the salt is by definition public information, so it does not represent a major vulnerability.

If a false request is sent through a native application, it is assumed that the client already has a salt and the server will not accept a client request that does not contain it. In addition, through the native application, the creator has several ways to prevent such a scenario by introducing additional keys that will identify the application itself and its state.

An Example of a Key Exchange

The password that is used is "alisinatajnalozinka".

The result of a hash function is:

-84 -12 -41 28 106 33 -36 94 -122 113 62 37 -82 103 -124 -55 -48 -125 9 -15 -97 72 51 31 -94 -46 52 -107 -71 29 -92 -90 25 -1 -80 68 -112 -43 15 -40 -106 61 105 35 -119 -80 102 113 72 20 32 10 -43 -54 -102 -89 83 122 88 52 -40 -62 7 6

When the bytes are turned into bits to obtain the client cards, the length of client cards is 384.

Here is an example of calculation of the key, where the first row represents the client's cards, the second is for server's cards and the third row is the key.

```
11 00 11 10 00 10 01 11 10 00
01 01 01 00 00 00 11 00 00 10
-- 1  -- -- 0  -- -- -- -- --
```

5. SCOPE OF APPLICATION

The scope of application of the protocol can be categorized based on the technologies or the type of client application which is used. The first type is a web browser, the other is a native application.

The protocol we discuss in this paper does not require any parameters other than the ones defined in the specification and as such it can be used for the exchange of the session key. The communication between participants does not reveal much, so the passive attacker can only get the username and, if it is the first request, the salt. The attacker can also get the username in a different way, while the salt is considered public information. With this in mind, we should say that the protocol does not provide any protection against recording of the traffic. Asymmetric cryptography may further protect such communication. When using proxy servers, we place our trust in the service provider, so we believe that the service provider will not abuse these two pieces of information.

Web Browser

The advantage of a web browser is that the application as such can be used for products from different companies. The user just needs to type the address of the service he wants to access and the content is available. Also, if you need to do a specific task, browsers support add-ons that can provide functionalities the browser vendor has not included in the code. From the perspective of security, browser vendors follow the development of technologies and protocols and browsers are generally updated regularly, so that they can respond to threats against security of user data.

However, the advantage that comes with the fact that one application is sufficient to use all services, also brings some disadvantages. An application that uses the browser must use the security protocols that the browser supports and must rely on them. One of the most important mechanisms is SOP (Same Origin Policy), which prevents requests from being sent towards third-party resources that are not allowed [7]. The browser also takes care of accessing the computer, and it keeps separated the data coming from applications in different domains, even if they are executed simultaneously. Additional protection is offered in that absolutely no access is permitted to the file system unless explicitly granted by the user for a specific real-time action. Software such as Flash Player or Java Applet, through which the vendor grants access to the computer as if it were a native application, is gradually phased out, since it represents a major security risk to the user. A single flaw in the Java programming language relative to SOP and security vulnerability in the SSL 3.0 protocol led to the BEAST (Browser Exploit Against SSL / TLS) attacks[8] where the attacker can decrypt the communication between a browser and web server.

That said, it should be noted that the memory the browser can use is limited. Therefore, all the values that are required for the application to run, some of which may be secret, usually come via a communication channel. This opens the possibility for data leakage.

Considering the above limitations, and in case we want to provide a user-friendly interface, we have to move forward with a user's password as a previously shared secret. In most cases, this means that the previously exchanged secret will not have satisfactory entropy, and therefore we must implement the steps outlined in the protocol to enhance security from this aspect.

As far as the memory is concerned, everything that the application has stored in the browser, either in the cookie or in the local storage of the browser, can be either accidentally or intentionally deleted by the user in one move. The browser grants the user full control over the content that is stored. Having control is good from the point of view of security and malicious software, but this also imposes limitations on the useful software.

Native application

Native applications offer a range of options. A key with good entropy can be sent at the time of installation which, if coupled with a user password, should provide good protection. Another benefit is that the application could authenticate itself with the server for a specific user.

On the other hand, unlike the browser, the user has no control of what the application is doing on the computer. Although operating system vendors are striving to protect users from malicious applications, the risk is much higher than it is for browsers where damage, in most cases, cannot occur if the user does not do something in that regard.

Using Protocol with Asymmetric Cryptography

By using the PKI infrastructure, where the user encrypts their first packet with a server public key and then the server sends cards encrypted with its secret key, user data can be additionally protected. The protocol does not have a rule as to whether the server must send all of its cards at once or in multiple separate messages. However, in the scenario where asymmetric cryptography is used, the server can generate cards, calculate the key and an authentication message, and encode everything with its secret key. This would require only one step of complex computational asymmetric cryptography operations on both ends.

In this scenario, the attacker could get server cards and authentication messages from the server, but with that information they could not learn anything else.

Key Exchange Between End Users

In the previous chapter, we discussed the encryption of files and their storage in a remote location. In one setup, the server does not have to know anything about the password used as a key material, i.e. it is only a source of cards with good entropy. This scenario can be used to exchange the key between two users. The protocol can be used in two ways. One would be that there is no record of the client on the server, and the other that there is such an account.

It is assumed that both clients share a common secret. These two clients may also be devices that belong to the same user who needs to transfer something from one device to the other or to connect them to communicate among themselves as is the case with SOHO (Small Office Home Office) networks or car keys.

A commonality of both approaches is that one of the clients generates a challenge and forwards it to another client through a channel different from the communication channel. When the server receives the same challenge from both ends, it will send the same cards to both addresses, so that the clients will use them to generate a session key.

If the protocol were to be used for ad hoc communication, the challenge should be a short sequence. The reason is that the other client must manually enter that value into the application they will use.

Scenario Without User Account

In the scenario without a user account, there is a problem of salt and user identification in case when the service is used via web browser. In case of a native application, all necessary values can be generated and stored together with the installation on both ends.

When it comes to the browser, both clients must define a name in advance by which the server could identify them. The protocol could be further enhanced by asymmetric cryptography, as it would thus protect the name and challenge.

The second problem is salt. Although the primary role of the salt is to avoid storing a user's password in plain text, in the discussed protocol, the salt serves as a value that will enhance the uncertainty of the password. In each exchange, the server would have to generate a salt and pass it on to the users along with the cards.

Scenario With User Account

With this setup, an initial request is somewhat different. The packet contains everything that the original protocol does, and users must also exchange the challenge. An addition is the username for the communication. When a server receives both requests, it adds a record that defines a link between the two clients, and joins the common salt. The salt is further treated as in the original protocol.

6. CONCLUSION

In the paper, a protocol for a session key exchange is presented in order to achieve perfect forward secrecy. The protocol as the key material employs a previously shared secret and array or random bits generated by the server side. It is not necessary to generate random values with satisfactory entropy on the user's end that would be used as a key material. The only random value is the challenge that is used at the last stage of the protocol to authenticate participants for communication and authorization of messages.

REFERENCES

[1] W. Diffie, P.C. van OOrschot, M.J. Wiener, "Authentication and authenticated key exchanges", Designs, Codes and Cryptography", Kluwer Academic Publishers, vol. 2, pp. 107-125, Jun 1992.

[2] 13 Authors, "Factorization of a 768-bit RSA modulus", CRYPTO'10 Proceedings of the 30th annual conference on Advances in cryptology, Santa Barbara, CA, USA — August 15-19, pp. 333-350, 2010.

[3] T. Wu, "The SRP authentication and key exchange system", 2000. [Online]. Available: https://tools.ietf.org/html/rfc2945

[4] T. Wu, "The secure remote password protocol", Computer Science Department, Stanford University, 1997

[5] S.M. Bellovin, M. Merritt, "Encrypted key exchange: password-based protocols secure against dictionary attacks", IEEE Symposium on research in security and privacy, pp. 72-84, 1992

[6] M. Milosavljevic, S. Adamovic, Kriptologija 2, Belgrade: Singidunum University, 2014.

[7] A. Barth, "The Web origin concept", 2011. [Online]. Available: https://tools.ietf.org/html/rfc6454

[8] T. Kurokawa, R. Nojima, S. Moriai, "On the security of CBC Mode in SSL3.0 and TLS1.0", Journal of Internet Services and Information Security (JISIS), vol. 6, pp. 2-19, 2016