Three-Party Password-Based Authenticated Key Establishment Protocol

Kartik Singhal, 150050025
Mukesh Pareek, 150050049
November 25, 2018

We have implemented the Three-Party Password-Based Authenticated Key Establishment Protocol Resisting Detectable On-Line Attacks due to Wang et al [1]

1 ABSTRACT

Efficient secure communication over public networks prone to adversarial attacks, is an important problem in network security. The whole of the working of internet is based on the security of protocols used for communication. Public Key Encryption schemes although secure for such applications are not very efficient as most of the existing protocols are computationally expensive. The popular solution is to use PKE to establish a common secret key for Symmetric Key Encryption and then use fast SKE schemes for Data Encryption. Such key exchange protocols are well studied under various situations and constraints and still continue to be interesting for cryptographers.
Password-Based Authenticated Key Exchange(PAKE) refers to the practical scenario where two or more parties have knowledge of their corresponding passwords (agreed on in advance or provided by a centralised authority) and establish cryptographic keys communicating over an adversarial network. The passwords are commonly spread over a considerably small set of strings which are human-memorable without any external hardware assistance and have considerably less randomness. The aim of such protocols is to prevent any adversary listening to the conversation from guessing the passwords or obtaining any non-trivial information about the established key.

Three-Party Password-Based Authenticated Key Establishment Protocol:[2] Humans can only remember so many passwords. When you are interacting with the vast community of users throughout the world, it's important to have common passwords which can be used with multiple users. Three-party protocols refer to the scenario where all users have passwords which are shared with a centralised server. For security reasons, the server most commonly stores a function of the password for authentication. Any two parties willing to communicate securely authenticate themselves with the server and the three parties exchange messages such that the two parties end up with a symmetric key for encryption and any adversary listening on the network obtains no information about the key.

1.1 Security Requirements of PAKE

- **Session Key Security :** The session key established through the protocol should only be known to the two clients.
- **Forward Secrecy :** If the passwords are revealed in the future, the session keys established in the past sessions should still be secret.

1.2 Possible attacks on PAKE

- **Man in the middle attack :** An adversary mediating the communication between any two parties and listening as well as altering the messages should not be able to fool the parties into believing that a secure protocol has been executed.
- **Replay Attack :** The attacker can resend messages sent previously.
- **Offline Dictionary Attacks :** An attacker eavesdropping on the communication can guess passwords and perform computations offline to verify the passwords. These kind of attacks are undetectable by the server and other parties, and hence we can't put a limitation on number of failed guesses.
- **Undetectable Online Dictionary Attacks :** The protocol might allow undetectable online guesses in some situations. Although this requires participation from the server and the parties, failed attempt will not get noticed as some party might not be able to distinguish a malicious request from an honest one.
- **Detectable Online Dictionary Attacks :** The attacker guesses the password and verifies using the response from parties and server, but a failed attempt will be noticed and the parties can take action accordingly.

1.3 What's Different in this Protocol?

As mentioned above, the PAKE protocols are well studied by cryptographers. A number of Three-Party PAKE protocols have been proposed which satisfy different security and efficiency requirements. The protocol we have implemented has two novel features:

- **Capturing the Human Participation :** As claimed by the author, none of the protocols propsed before this, capture this property that both the client machines are being operated by human users. This can be used to make the protocol more secure and efficient. This property is captured by using CAPTCHA.

- **Low Computational Power of Client Machines :** Here we work with the assumption that the client machines are light-weight machines and are unable to perform computationally expensive operations whereas the server is assumed to be computationally powerful. Here partial authentication is done by human participators which reduces the complexity. This setting eliminates a lot of protocols which employ additional precautions for resisting on-line detectable attacks which are costly.

1.3.1 Hard AI Problem and CAPTCHA

We defer the formal definition of an hard AI problem and suggest the reader to refer the original paper. Intuitively, an AI problem is called to be hard, if it can be solved with considerable success probability by most humans, and is hard for computationally bounded machines. A CAPTCHA(Completely Automated Public Turing test to tell Computers and Humans Apart) is a program which generates such hard AI problems which humans can pass but machines can only do so with negligible probability. Captchas are in use at numerous places. We have used a freely available library[3] for captcha implementation which we have modified for our use. Solving the problem corresponds to recognizing a distorted image of a string which is hard for machines.

2 The Protocol

2.1 Protocol Setting, Computational Assumptions and Notations

- **S** is a trusted server which runs on an independent machine and has the passwords for clients.

- **A** and **B** are the identities(strings) of two clients, ClientA and Client B, which have shared a password with the trusted server. ClientA and ClientB wish to establish a common secret key.

- **G** is the Quadratic Residue group(QR_P^*) where $P = 2903$ is a safe prime. Observe that **G** is a cyclic group of order 1451. $g = 3$ is a generator of **G**. As P is a safe prime, the Computational DDH assumption holds in **G**.

- $E_K(M)$ denotes the encryption of a message M and $D_K(C)$ the decryption of a message C using an SKE scheme with key K. We use AES128 block-cipher in CBC mode as the SKE scheme.

- **H(M)** is the hash of message M using a one-way hash function. We are using MD5 hash with a 16-byte output as the hash function.

- pw_u denotes the Key derived from password of user U, using a key-derivation function. We use pw_u = H(pw), i.e. the MD5 hash function as the key-derivation function.
- $\phi(r,t)$ is the captcha function, which takes in a string r and randomness t, and outputs the captcha. For our implementation, $\phi(r,t)$ takes in 15 lettered strings and outputs the GIF encoding of the captcha image.
  We restrict the range of characters allowed in the captcha string to elements of the set $S' = S/\{e,g,z\}$ where S is the set of small-case English alphabet. We avoid the particular letters because of limitations of the captcha generator. Ω_n is the set of all possible strings over S' of length n. For demo purpose we choose n = 15. $|\Omega_n| = 23^{15} > 2^{67}$

2.2 Description of the Protocol

Note : The group G, the generator g, the Encryption scheme and hash function to be used in the protocol are public and known to all the parties. The identities A and B are also known to the server and both the parties.
The protocol proceeds in the following steps:

- A → picks $x \in_R Z_P^*$
- A → computes $M_1 = E_{pw_a}(g^x)$
- A → sends M_1 to B

- B → picks $y \in_R Z_P^*$
- B → computes $M_2 = E_{pw_b}(g^y)$
- B → sends M_1 and M_2 to S

- S → obtains $g^x = D_{pw_a}(M_1)$
- S → obtains $g^y = D_{pw_a}(M_2)$
- S → picks $s_1, s_2 \in_R z_P^*$
- S → computes $K_{AS} = g^{xs_1}$
- S → computes $K_{BS} = g^{ys_2}$
- S → picks $r \in_R \Omega_n$
- S → generates $captcha1 = \phi(r, t_1)$
- S → generates $captcha2 = \phi(r, t_2)$
- S → computes $M_3 = E_{K_{BS}}(captcha1)$

- S → computes $M_4 = E_{pw_b}(g^{s_2})$
- S → computes $M_5 = E_{K_{AS}}(captcha2)$
- S → computes $M_6 = E_{pw_a}(g^{s_1})$
- S → sends M_3, M_4, M_5 and M_6 to B

- B → computes $g_{s_2} = D_{pw_b}(M_4)*$
- B → computes $K_{BS} = g^{ys_2} = (g_{s_2})_y$
- B → computes $captcha1 = D_{K_{BS}}(M3)$
- B → takes $r1$ = input from user, the string visible in the captcha1
- B → computes $M_7 = H(1||r1||B||A)$
- B → sends M_5, M_6, M_7 to S

- A → computes $g_{s_1} = D_{pw_a}(M_6)*$
- A → computes $K_{AS} = g^{xs_1} = (g_{s_1})_x$
- A → computes $captcha2 = D_{K_{AS}}(M5)$
- A → takes $r2$ = input from user, the string visible in the captcha2
- A → checks if $M_7 = H(1||r2||B||A)$
- A → computes $M_8 = H(1||r2||A||B)$
- A → computes $SK = H(2||r2||A||B)$
- A → sends M_8 to B

- B → checks if $M_8 = H(1||r1||A||B)$
- B → computes $SK = H(2||r1||A||B)$

2.3 Implementation Details

1. We use the general purpose cryptographic c++ library, Libgcrypt for encryption and hashing
2. We use the Mersenne Twister(MT1937) PRNG for picking elements uniformly at random from a group
3. The modified captcha creation function adds randomness using the above mentioned PRNG which works as the random distortion parameter t_1 and t_2 to generate different captcha image each time for the same string
4. The captcha images are generated and stored as ".gif“ files on the user machine, we have not implemented a GUI of our own for captcha visualisation and the user will have to use the system image viewers for the same

3 Security Analysis

3.1 Meeting Security Requirements

- **Session Key Security :** We assume that the server is trusted. From the computational DDH assumption, security of the AES encryption and one-wayness of the hash function, it is clear that any adversary listening on the communication channels, knows nothing non-trivial about the finally established common key.
  The adversary cannot obtain the string r from $M_7 = H(1||r||B||A)$ or $M_8 = H(1||r||A||B)$ due to hardness of inverting the hash function.
  The adversary cannot decrypt the encoded captcha images without knowing the secret keys, which are derived from the password, and are distributed randomly as the key-derivation function is a hash function.

- **Forward Secrecy :** Since we assume that the DDH assumption holds on the group G, it's hard to retrieve g^{xs_1} and g^{ys_2} even after knowing the password after the protocol begins as the knowledge of password only allows the adversary to decrypt and obtain g_{s_1}, g^{s_2}, g^x and g^y, which reveal nothing about k_{AS} and K_{BS} required to decrypt the captcha images

3.2 Resistance against various attacks

- **Man in the middle attack :** As the identities of the two parties are known to the server, and the output of the key-derivation function is uniformly distributed, any attacker without the knowledge of passwords can't fool the server and clients.

- **Offline Dictionary Attacks :** As the password pw_a is only used to encrypt g_x and g_{s_1}, offline dictionary attacks might help only if the attacker is able to retriece $K_{AS} = g^{xs_1}$ from these, which is hard because of the DDH assumption. Hence the protocol is immune to offline password guessing attacks.

- **Detectable and undetectable online attacks :** As recognizing the captcha strings requires human involvement, the comparison of the captcha images for verification of password guesses is slowed down making brute force guessing and verifying infeasible. Hence the protocol is immune to online password guessing attacks.

4 CONCLUSION

As commonly found in the literature, detectable online password guessing attacks are generally overlooked while designing three-party PAKE protocols, and it is believed that additional precautions like delaying response, introducing exponentially increasing delays with each failed attempt or locking the password after some fixed number of failures are necessary to resist such attacks. While these additional precautions are sometimes costly, and might be impractical to implement on light-weight client machines, they are also prone to Denial of Service(DoS) attacks.
In this paper, the authors have combined the popular hard Artificial Intelligence problem, CAPTCHA with cryptographic assumptions (like DDH) and techniques (SKE, One-way hash function) to show that it is possible to achieve resistance against detectable online password guessing attacks along with other known attacks without any additional precautions.

5 REFERENCES

[1] **Three Party Password Based Authenticated Key Establishment Protocol Resisting Detectable On Line Attacks**, Weijia Wang, Lei Hu, 2013.
[2] **A Survey on Three Party Password Based Authenticated Key Exchange (3 PAKE) Protocols**, Shivani Bhatia, Rekha Saraswat, 2013.
[3] **CAPTCHA library anti bot image generator**, Ivan Tikhonov
[4] **WH-3PAKE Github Repository**, Kartik Singhal, Mukesh Pareek, 2018