A Secure Zero-Knowledge Two-Factor Authentication Protocol

Albert Chalom
Nathan Wolfe
achalom@college.harvard.edu
nwolfe@college.harvard.edu

Abstract

We present a two-factor authentication protocol for a password and fuzzy password (where the original and current fuzzy password might have some noise and therefore only need to be close rather than match). Our protocol keeps the password and fuzzy password confidential from the server with the server only storing a verifier of the password and fuzzy password. Use cases of this protocol could be that a user wants to authenticate him/herself to a server with a password and fingerprint scan, but does not wish the server to learn his/her password and fingerprint scan.

We provide an authentication protocol and then prove security of our protocol. The protocol is based on generic two-party secure computation, but we also provide a lightweight computation method using Yao's Garbled Circuits specifically for our protocol. We end our paper by discussing what assumptions an optimal protocol must make to be secure, and show how our entropy assumptions are close to optimal.

1 Background

1.1 Key Exchange

In 1976 Diffie and Hellman created the first protocol to allow two users to create a secure channel on an untrusted network[4]. Prior to this protocol, if two entities wished to create a secure channel, they would have had to agree on a shared key earlier. This protocol, now known as a Diffie-Hellman key exchange, allows two users to agree on a key through an untrusted network, such that any adversary cannot learn the key without exponential computational power. This now forms the basis for many authentication protocols that served as inspiration for our paper.

1.2 Password authenticated key exchange

In 2000, Boyko et al.[3] provided a PAKE (Password authenticated key exchange) protocol that allows a two users to create a shared key if they agree on a password, or a client and server to create a shared key if the client provides a proper password. Their protocol also protects against an adversary who monitors the communication network from being able to run an off-line dictionary attack to discover the users password.

Importantly, they offered an extension to the protocol, called PAK-X, where only one participant stores the password as plaintext, and the other user stores a verifier, keeping the password hidden from the server. That extension inspired our protocol.

1.3 Fuzzy password authentication

1.3.1 Fuzzy passwords

Fuzzy passwords are a generalization on passwords, where the authenticator allows for the fuzzy password to be noisy. When a user wants to authenticate him/herself they would provide a fuzzy password that just has to be close enough to the original fuzzy password. Oftentimes Hamming distance will be used to measure closeness, and a verifier will output success if the fuzzy password has a small Hamming distance from the original (determining whether two n length strings match in all but δ locations). However, any definition of closeness can be used.

Fuzzy passwords are useful for using biometric information as a source of authentication. When a person repeatedly takes their finger print or iris scan, the scans might not be identical, and the fuzzy pass-string generated will have some noise. Therefore, an authenticator would want to verify that two fingerprints or iris scans belong to the same person, but allow for the fuzzy passwords not to be complete matches.

1.3.2 Fuzzy Password Authenticated Key Exchange

Dupont et al.[5] provide a scheme for authenticated key exchange provided that two fuzzy passwords are close enough. Their work was the first instance of allowing these fuzzy passwords to be low entropy to prevent an eavesdropping adversary from using an off-line dictionary attack to learn the fuzzy password. However, this scheme requires both parties to know the fuzzy password, and if two fuzzy passwords are close enough it allows either party to learn one bit of the other entity's fuzzy password.

1.4 Motivation

Existing schemes exist allowing two-factor authentication, password authentication, and fuzzy password authentication. However, fuzzy password authentication schemes require the server to store the the fuzzy password directly. We wanted to create a scheme that doesn't require the server to store the fuzzy password in the plain. Use cases for this might be a user who does not want the server to know his/her biometric information. Even if the user trusts the server, a user would want a scheme that keeps his/her biometric information safe even if the server gets hacked. We therefore wish to provide a two-factor authentication protocol for a password and fuzzy password where neither the password nor fuzzy password is stored on the server.

We do not focus on the key exchange aspect here, as we assume the user and client can use any secure existing key exchange protocol or even password key exchange protocol and then perform this authentication step and halt communication if authentication fails.

2 Overview

Section 3 will describe our assumptions, section 4 will present our scheme, section 5 will prove correctness and security, and section 6 will show possible implementations of the necessary secure two-party computation. Section 7 is a discussion of our password entropy requirements, compared to the optimal case. We will then discuss future work that can be done to extend our work further.

3 Assumptions

Our scheme requires a user to provide a normal (non-fuzzy) password with high entropy and to have a way of generating a fuzzy password (likely a fingerprint scanner).

We assume that the client and server have established confidential and integral communication using a protocol such as TLS/SSL. We also assume that the server has proven its identity by means of a certificate. However, the client is not yet authenticated; the server wants to determine their identity.

We assume that we have a pairwise independent hash function, and access to two-party secure computation. We define two-party secure computation as follows:[1]

Two-party secure computation

Let F be a two party function. We say a protocol is secure for calculating F if there are parties 1 and 2 with inputs x_1 and x_2 and outputs y_1 and y_2 respectively, such that for every efficient adversary A there exists an efficient "ideal adversary" S such that for every set of inputs by honest users the following distributions are computationally indistinguishable

- The tuple (y_1, y_2) where A controls some adversarial parties and the protocol is executed as above. We also allow the ith party to abort the protocol, and denote this by $y_j = \top$ for $j \geq i$.
- The tuple y_1, y_2 that is created by an ideal adversary as follows
  1. We let S choose x_i for all users controlled by the adversary and define $(y_1', y_2') = F(x_1, x_2)$.
  2. For $i = 1, 2$ we allow S to choose to abort the protocol at the ith stage. If the adversary aborts the remaining users learn nothing, and if the adversary doesn't abort user i learns y_i'
  3. Let k be the last stage that wasn't aborted. For all honest users $i \leq k$ we set $y_i = y_i'$, for all honest users $> k$ we set $y_i = \top$ and we let all users the adversary controls to set y_i however they like.

We will provide some possible implementations of such two-party computation in section 6.

Required Encryption/Decryption scheme

We also require that our Encryption/Decryption functions be CPA-secure and our decryption function output ERROR if a user tries to decrypt using the wrong key. This assumption of outputting error isn't new, but for concreteness, we provide a possible encryption scheme, though many others can be used as well.

Let S, V be a secure MAC, and E, D be a encryption scheme. We will now construct and encryption scheme E', D' that with high probability will only decrypt if the correct key is used. We define $E'_{k1,k2}(m) = E_{k1}(m, S_{k_2}(m))$. Then we let $m, \sigma = D_{k1}(c)$. We define $D'_{k1,k2}(c)$ to output m if $V(m, \sigma) = 1$ and ERROR otherwise.

We now claim that if an invalid key is used, this will output error with high probability. To see this we will consider the following two cases where $k1$ and $k2$ are incorrect, and show how this will lead to an error.

Case 1: Incorrect k1. In this case, since an incorrect decryption key is used, $D_{k1}(c)$ should decrypt to a string that is computationally indistinguishable from uniform random. Then, if we break this random string into m, σ the probability that $V(m, \sigma) = 1$ where both m and σ are random is negligible.

Case 2: Incorrect k2. In this case, since an incorrect verification key is used, the probability that $V_{k2}(m, \sigma) = 1$ is negligible. This follows from the fact that if a key is uniformly chosen from random, the probability that $V(m, \sigma) = 1$ is negligible.

4 Scheme

We will present a scheme under the assumption that we have secure two-party computation, and in section 6 we will provide possible implementations of secure two-party computation.

Our scheme requires a user to have a password and a way of capturing biometric information. When creating an account, a user would choose a password p and a capture their biometric information, (often a fingerprint) fp. The user would then send $c = E_{H(p)}(fp)$ to the server where E is a secure encrypting function and H is a hash function $\{0,1\}^* \rightarrow \{0,1\}^n$. The server keeps c as a verifier. The user must remember p, but doesn't have to remember fp.

Then when a user wants to authenticate itself it knows its password p and it can make a new (close) fingerprint scan fp'. Then using secure two-party computation, the user can provide $H(p)$ and fp' and the server can provide c. The computed function determines whether fp' is close to $D_{H(p)}(c) = fp$ without revealing any additional information to the user or the server. The securely computed function outputs 1 if fp and fp' are close and 0 if they are not close or the decryption with key $H(p)$ gives ERROR.

5 Analysis

For our scheme to be useful, it must have the properties of a zero-knowledge proof. That is, it must be complete, sound, and zero-knowledge. We consider the client to be the prover and the server to be the verifier.

5.1 Proof of completeness

Completeness means that if both parties are honest and the network allows both the user and server to compute the function then the server will always verify the user (assuming the right password and two close enough fingerprints were provided).

To see this we note that when the honest user creates an account, they provide the server with $E_{H(p)}(fp)$ and then the honest server correctly stores this. Then when the user wishes to authenticate, he/she supplies the key $H(p)$ to the secure computation protocol. Then by construction the two-party computation function will output 1 if fp' is close to fp.

5.2 Proof of soundness

Soundness means that if a user trying to authenticate him/herself provides an incorrect password or a fingerprint that is not close enough to the original fingerprint then the probability that the server will authenticate the user is negligible. More formally, following the two party secure definition provided above, we can assume that the authenticating user is under adversarial control and the server is honest. We only consider the case of having the adversary control the user and assume that the server is honest (if the server were malicious it could just arbitrarily authenticate anyone).

We will now model our verification function as $F(p, fp, c)$ where the user provides the password p and fingerprint fp and the server provides the original encrypted fingerprint c. In the two party computation definition provided above, $x_1 = p||fp$ and $x_2 = c$. We define $F(p, fp, c) = (1, 1)$ iff fp is close to $D_{H(p)}(c)$ and $F(p, fp, c) = (0, 0)$ otherwise. We will now consider the following two cases of the the user attempting to authenticate with an incorrect password and then with an incorrect fuzzy password.

Case 1: Incorrect password

Let p be the password and p' be the incorrect password the user provided. Then, because H is a pairwise independent hash function, $Pr[H(p) = H(p')] = \frac{1}{2^n}$ where $n = |H(p)|$. Except with negligible probability, $D_{H(p')}(c)$ will output `ERROR` as an incorrect key was used for decryption. Then because our decryption fails, `ERROR` is not close to fp so by construction $F(p, fp, c) = (0, 0)$

Case 2: Not close enough fuzzy password

In this case, F correctly decrypts c. However, in this case fp and fp' are not close enough, and by construction of our function F if fp and fp' are not close enough $F(p, fp, c) = (0, 0)$.

In both these cases we showed that if the honest server provides the correct input x_2, the two party function $F(p, fp, c) = (0, 0)$. We will now show that if we use a secure two-party computation protocol for F that authentication would fail.

Following our protocol definition above, the adversarial user has the option of halting the protocol (in which case that adversarial user never gets authenticated), or setting $y_2 = F(x_1, x_2)$[2]. Since we showed in both cases above $F(x_1, x_2) = (0, 0)$ then except with negligible probability the server will receive 0 or $\top$, and the adversarial user will be unable to authenticate itself with an invalid password.

5.3 Zero knowledge

Zero knowledge implies that the server cannot learn anything about fp, fp', or $H(p)$ and no malicious user can use the protocol to learn $p, H(p), fp$ or fp'. This follows from the definition provided earlier of a secure two-party computation protocol where the only thing each user learns is either the output of the function being computed y_i or that the protocol was aborted $\top$.

We will show possible implementations of two-party protocols that provide zero knowledge.

6 Possible 2-Party Computation Implementations

6.1 Introduction to Yao's Garbled Circuits (YGC)

Bellare et al.[2] set forth a formalization of garbling schemes. The premise is that we have two parties, call them Party 1 and Party 2. Party 1 has a function f; Party 2 has an input x; they wish to evaluate $f(x)$. Bellare et al. demonstrated a garbling scheme with *privacy, obliviousness, and authenticity,* guaranteeing that $f(x)$ is calculated correctly, such that neither party gains information about the inputs f, x. However, parties 1 and 2 could select f or x in a malicious way, respectively.

In our authentication protocol, we let Party 1 be the server and Party 2 be the client. The server is not concerned about the client choosing a malicious input x; the client merely provides a password and fingerprint, and if these are wrong the only consequence is that the authentication fails. However, the client should be concerned about whether f is the right function. If f is constructed maliciously, $f(x)$ could reveal information about the client's password or fingerprint.

The usual garbling scheme used is called *Yao's Garbled Circuits*. Party 1 sends a "garbled" circuit F to Party 2. Party 2 obtains a likewise-garbled version of its input, X, through oblivious transfer with Party 1. Finally, Party 2 evaluates on F, X to obtain $f(x)$. [7]

Now we focus our attention on how to guarantee that the garbled circuit F computes a function that is not malicious.

6.2 General YGC for malicious adversaries

Lindell and Pinkas[6] present a YGC secure two-party computation using the "cut-and-choose" technique. In this protocol, Party 1 provides λ versions of the garbled circuit. Party 2 can then use some of these replicas to check if the garbled function is correct. By this method, the probability of cheating is exponentially small in λ.

Unfortunately this method is not that efficient. To achieve a cheating probability of $2^{-\lambda}$, the number of circuits is multiplied by λ, and the number of oblivious transfers is multiplied by $\lambda/2$.

6.3 Lightweight YGC for this instance

We can optimize for our authentication scenario by making a security concession.

6.3.1 Scheme

Party 2, the client, wants to authenticate; say their authenticating data is x. Say the server, Party 1, has f that determines the authentication, with $f(x) = 1$ a successful login and $f(x) = 0$ a failure.

Now let the server choose random $z \in \{0, 1\}^m$. Define a new function g:

$$g(x) = \begin{cases} 0 & \text{if } f(x) = 0 \\ z & \text{if } f(x) = 1 \end{cases}$$

Now the server can let the client securely compute $g(x)$; the server will consider the client authenticated if they can determine the secret z.

Let H_1 be a secure collision-resistant hash function. The server sends the client $H_1(z)$ as a commitment.

Now we use YGC; the server provides a single garbled circuit G and lets the client evaluate with their garbled input X. The client obtains $z' = g'(x)$ where g' is the function actually encoded in G. The client keeps the YGC output private.

The client checks if $H_1(z') = H_1(z)$. If so, the client responds with z', authenticating, otherwise, the client aborts.

6.3.2 Correctness

By construction, the client authenticates successfully if $f(x) = 1$ and both parties are honest.

This scheme is sound toward malicious clients; if $f(x) = 0$, the client has no way of determining z. The client has $H_1(z)$, but since H_1 is a secure hash function, the client cannot determine z from this efficiently.

6.3.3 Security

What security does the client have against a malicious server? The garbled circuit G may actually encode any function g' by the server's choice.

The client only has two possible responses: z', or abort. The client only responds z' if $H_1(z') = H_1(z)$. Since H_1 is collision-resistant, the probability that $z' \neq z$ is negligible.

So, except with negligible probability, the client responds with z or aborts. The server generated z originally, so that response contains no new information. The only information to be gleaned is in whether or not the client aborts.

Thus, with every authentication interaction, a malicious server may learn one bit of information by observing whether the client aborts. That is our security concession.

However, if the client is quick to cut ties with a server it suspects is cheating, the amount of information the server learns will be limited. If the client ever has to abort an authentication interaction, it may conclude that the server is cheating. Thus, it makes no more interactions with that server.

The malicious server thus may learn information only until the first time the client aborts. For instance, if the server learns bits of the client's password one at a time, with each bit having probability $1/2$ of 0 or 1, the average number of bits learned, including the aborted step, will be 2; the probability of learning k bits or more is 2^{1-k}. For long enough passwords, this information given up may not be an issue.

6.3.4 Performance

The upside of this security concession is that this computation scheme only requires one exchange of a garbled circuit. Compare this to the fully secure approach in Section 6.2, which uses λ circuits to get a $2^{-\lambda}$ probability of cheating.

One possible downside of this scheme is since we use fuzzy passwords, it is possible that the client just takes a bad scan of their fingerprint, which results in the client aborting and cutting ties with the server, when in reality the server was honest.

7 Entropy Discussion

If the server gets hacked, in order to prevent against a dictionary attack we require that the user's password is high entropy. This assumption is nearly optimal; any system that wishes to be secure against a dictionary attack, should the server be compromised, would require a high entropy source. We can formalize this:

If a server wishes to authenticate a user, we can model this as a function $f(u, s)$ where u is some data the user provides (such as a password, fingerprint, etc.) and s is some data the server provides (such as a copy of the password, a hash of the password, etc). Then $f(u, s) = 1$ authenticates the user, and $f(u, s) = 0$ rejects the authentication attempt.

Theorem 7.1. If a server is compromised such that an adversary learns f and s, and an authentication $f(u^*, s) = 1$ exists, then the adversary can perform an off-line dictionary attack to find u' such that $f(u', s) = 1$; i.e., an attack by guessing all possible values of u, without using any network interactions.

Proof. An adversary given f and s can evaluate $f(u, s)$ for all values of u, without using the network. Then, since $f(u^*, s) = 1$ exists, if the adversary does this enough times, it will find a u' such that $f(u', s) = 1$. If u^* is a password from a low entropy distribution it is likely that $u' = u^*$, but in any case this will allow the adversary to authenticate as the user. □

7.1 Relevance to our scheme

One flaw with our scheme is that we require the user password to be high entropy rather than the combined user input (i.e. the password and fingerprint combined) to be high entropy to provide protection should the server get hacked. This follows from the fact that we encrypt the fingerprint data with a hash of the password, so if an adversary with the server's data discovers the hash of the password they can learn the user's fingerprint. In an ideal scheme, the verification for the password and fingerprint would be separate such that learning one of these two factors as well as the data stored on the server, does not allow an adversary to learn the other.

However, while our scheme is not ideal, we do note that if we assume the data on the server is kept confidential, then if an adversary learns a user password from some other source (say a user uses the same password for all their accounts) then this still does not help the adversary learn the user's fingerprint. This is because the adversary does not have a copy of the encrypted fingerprint from the server, and all authentication communication is done through a secure channel. Thus, the adversary still cannot impersonate the user through two-factor authentication.

8 Future Work

As mentioned before, one flaw with our scheme is that to protect against a dictionary attack should the server be compromised, we require that passwords are high entropy. However, in the ideal case we would just require that either the password or the fuzzy password are high entropy. Future work could provide an independent verifier for both a password and fuzzy password without storing the actual password and fuzzy password, or prove that this is impossible.

Another flaw with our scheme is that two-party computation is slow in practice and ensuring that it is zero-knowledge is particularly slow. Future work can either improve generic performance of a zero-knowledge two-party computation scheme, or find how our scheme can specifically be optimized even if two-party computation is still slow for the general case.

References

[1] Boaz Barak. Multiparty secure computation I: Definition and Honest-But-Curious to Malicious complier. CS127 Lecture Notes 2018

[2] Mihir Bellare, Viet Tung Hoang, Phillip Rogaway. Foundations of Garbled Circuits. CCS 2012

[3] Victor Boyko, Philip MacKenzie, Sarvar Patel. Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. Eurocrypt 2000

[4] Whitfield Diffie, Martin. E Hellman. New Directions in Cryptography. IEEE 1976

[5] Pierre-Alain Dupont, Julia Hesse, David Pointcheval, Leonid Reyzin, Sophia Yakoubov: Fuzzy Password-Authenticated Key Exchange. IACR Eurocrypt 2018

[6] Yehuda Lindell, Benny Pinkas. An Efficient Protocol for Secure Two-Party Computation in the Presence of Malicious Adversaries. Journal Of Cryptology, April 2015.

[7] Sophia Yakoubov. A Gentle Introduction to Yao's Garbled Circuits. 2017