Two Factor Zero Knowledge Proof Authentication System

Quan Nguyen
Mikhail Rudoy
Arjun Srinivasan

6.857 Spring 2014 Project

Abstract

It is often necessary to log onto a website or other system from an untrusted device using an untrusted connection. In this paper, we propose a login protocol that can be implemented to allow security in this situation. The protocol is a two factor authentication scheme which utilizes zero knowledge proofs to convince a server, over an unsecure connection, that the user knows his password without compromising that password. The protocol never requires the user to enter any sensitive information on the (potentially compromised) device that they are logging in on. We analyze the security of this protocol, noting which vulnerabilities it protects against and which it is susceptible to. We also describe our implementation of this protocol, which can be seen at ***http://tfzkp.herokuapp.com****.*

Introduction

It is often the case that logins to secure systems are made from untrusted devices, such as public kiosks or otherwise possibly compromised systems. There are many ways these systems can be compromised. One common way is through the use of a hardware or software keylogger. Hardware keyloggers act as devices that monitor keyboard and other input from input devices to the system. These allow an adversary to have full knowledge of all data transferred from the user to the system, including login credentials. Software keyloggers behave similarly, but are pieces of malicious software that use the network to relay information back to an attacker. If any sort of information is sent to the untrusted system, there is no way to avoid a keylogger from tracking this data.

In addition, even systems that are not compromised at the software or hardware level are subject to human intervention, including shoulder surfing. This involves a human or camera behind a user as he or she types in login credentials. This is especially prevalent in public areas, such as outdoor ATMs. Some solutions developed include cameras placed to track suspicious behavior using computer vision techniques; however, these are not reliable, as the shoulder surfer does not necessarily have to be human, but can be any sort of monitoring device.

The only way to reliable and practical way to log in to systems in these situations involve not passing any sensitive information through the untrusted device. For this reason, we look to the combined use of zero knowledge proofs and two factor authentication.

In this paper, we propose a protocol that is meant to solve all of the problems described above. In particular, the protocol is intended to allow a user to log in to an account on an untrusted device while in the presence of keylogging and shoulder surfing using an unsecured network. The protocol allows the user a trusted device to aid in his log in which can also be shoulder surfed and also connects over an unsecured network, but is guaranteed not to be keylogged or otherwise compromised.

Background

To combat the security vulnerabilities detailed above, a few security techniques can be combined. **Two Factor Authentication** is a method commonly used by internet services to provide an extra layer of security in addition to the standard password used as login credentials. It employs a secondary device, such as a phone, that the user must have in his or her possession to complete the authentication process. In services such as Google's GMail, two factor authentication works using a phone's SMS capabilities to send text messages from authentication servers with access codes to the phone. This is combined with the password, so authentication requires both *knowledge* and *possession* of a trusted device.

Two factor authentication still requires sending a password over a network, however. To combat this, we introduce the concept of zero knowledge proof. Zero knowledge proof is a type of proof that is compete and sound and has the zero knowledge property. The completeness and zero knowledge properties amount to the idea that the prover can only successfully convince the verifier of the fact being proved if and only if the fact is true. The zero knowledge property states that the verifier learns nothing from the proof other than the fact that proven fact is true. What we use in our protocol is a zero knowledge proof that the prover knows a secret key; because of the zero knowledge property, the key is not compromised, only the fact that the prover knows the key.

Protocol

We can now describe the protocol that we developed to address the problems introduced earlier. This protocol involves two separate stages. The first stage is the signup stage, during which the user creates an account with a website or other service. The second stage is the login stage, and consists of the steps necessary to log in.

Before we describe the particulars of these two stages, we will introduce the principals in this scheme:

User: The user is the individual who wishes to securely maintain and access an account with one or more services.

Trusted Device (TD): The trusted device is a device that belongs to the user. It must be a mobile device such as a phone, tablet, or laptop because it will be required every time the user attempts to log in. We assume that the trusted device is not compromised.

Untrusted Device (UD): An untrusted device is a device that the user wishes to use in order to log into an account that the user has. The untrusted device is, as the name implies, not trusted; it is assumed to be keylogged and shoulder surfed.

Server: A server in this scheme is the device in charge of a particular service. All interaction with the service is achieved through communication with the server.

Adversary: In this scheme we consolidate all possible malicious actions under a single hypothetical individual who we refer to as the adversary.

Account Creation Protocol

Before the user can log into an account with a service, the user must have an account with that service. Setting up an account is actually a very simple process.

To begin, the user selects a username and communicates it to the server via the trusted device. If this username is already taken, the user is informed with an error message, and the user must make another attempt. When the user identifies an unused username, the process can continue.

In the next step, the user selects a password and enters it into the trusted device. The trusted device uses that password to (deterministically) generate a secret key; for example, the binary representation of the password can be interpreted as a large number and that number can be used as the secret key. The trusted device then randomly generates and saves a secret key for itself. The public keys associated with each of the two secret keys are then computed. After that, the username and both public keys are sent to the server, which stores this data. This set of steps is summarized in figure 1 below.

It is important that, once this process is complete, the trusted device deletes all references to the user's password or to the associated public key.

In our scheme, we assume the discrete logarithm assumption. This allows us to keep the private key, public key pairs in a particular format: a private key is an integer x and the associated public key is the triple (g, g^x, p) where p is a large prime and $g \in Z_p^*$.

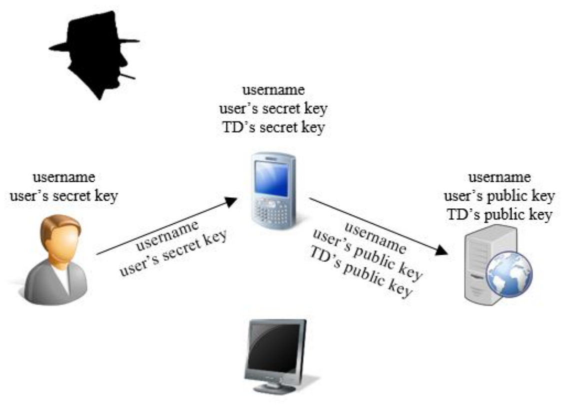


Figure 1: the setup process summarized

Login Protocol

The user, having created an account may find that he wishes to log in from an untrusted device.

He begins by selecting the service that he wishes to log into on his trusted device. The device sends a message, digitally signed with the device's secret key, to the server indicating the intent of the user (as identified by his username) to log in. The server verifies the signature, and either accepts the signature as valid, or rejects it as invalid. In the former case, the server saves in its records that a login attempt has begun. Just in case, this login attempt expires within a minute if it is not continued. The steps so far have been summarized in figure 2.

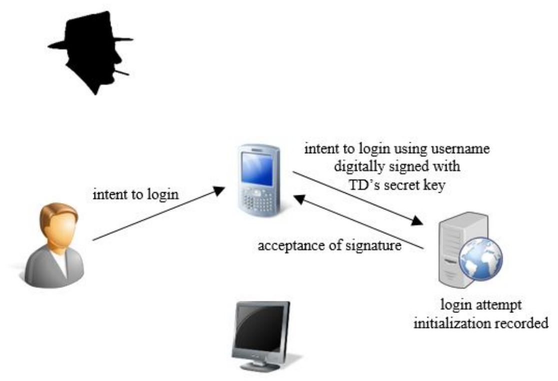


Figure 2: the first steps to logging in

After that, the trusted device requests that the user enter his password. Once the password is entered, the trusted device computes the user's secret key from the password. Next, the trusted device runs through several rounds of zero knowledge proof with the server to demonstrate knowledge of the user's secret key to the server. Every message sent should be signed with the trusted device's secret key such that all messages not sent by this device can be ignored. Suppose we let x be the user's secret key. The server has access to g and g^x, so the proof necessary is the proof of knowledge of discrete logarithm. A zero knowledge proof of this kind is easy to do, as seen in [?]. The next figure demonstrates these steps.

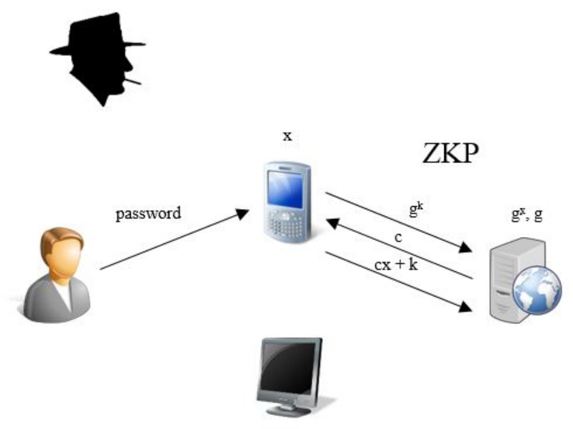


Figure 3: the user tells the trusted device his password, and the trusted device proves knowledge of the password to the server

When the server is sufficiently convinced that the trusted device currently knows the password, it generates a random token and associates that token with this login attempt. Once again, if the token remains unused for too long, the login attempt expires. The token is encrypted with the trusted device's public key and sent to the trusted device. The trusted device decrypts the message and displays the token on screen. The user then enters his username and the token into the untrusted device which sends the data on to the server. These steps are shown in figure 4.

The server checks that the user has a login attempt in progress and verifies that the token entered matches the one associated with that login attempt. If everything checks out, the untrusted device is informed that it successfully "logged in half way", and the token is removed from the records; as a result, the token can only be used once. If verification fails (for example if the adversary shoulder surfs and enters that token first), the untrusted device is informed of this fact. In either case, the untrusted device then

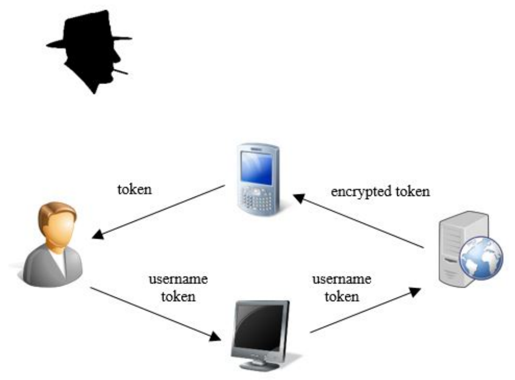


Figure 4: the user uses the token that is sent by the server to "login half way" on the untrusted device

displays to the user whether or not he has "logged in half way." The user uses this information to inform the trusted device of whether the first half of logging in was successful. The device in turn, forwards this answer to the server. If the answer is no, the login attempt is aborted and must be restarted. If the answer is yes, the untrusted device must be the single device that is logged in half way, and the protocol can continue. These steps are summarized in figure 5.

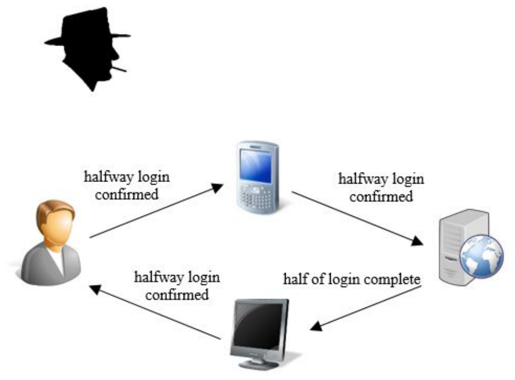


Figure 5: the user confirms to the server that it is the untrusted device that has successfully logged in half way and not a device in use by the adversary

At this point, the trusted device does another few rounds of ZKP resulting in the server sending another encrypted token. Once again, the trusted device decrypts the token and the user copies it to the untrusted device. This time, when the untrusted device sends the data to the server, the server verifies not only that the token is correct, but also that the device attempting to log in is the unique device that is "half way logged" in. At that point, the device's halfway login is upgraded to a full login, and the server can serve the untrusted device the content that the user is attempting to access. Once the second token is printed on screen, the trusted device removes all record of the user's password or secret key. See figure 6 for a summary of these final steps.

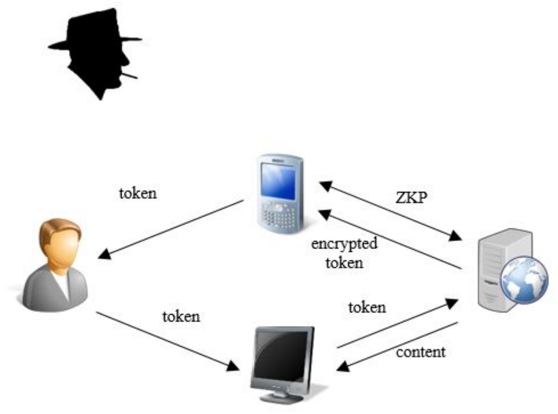


Figure 6: a second round of zero knowledge proof and a second token which will only be accepted from the "halfway logged in" device are used to complete the login attempt

Security

The above protocol is meant to solve the problems of keylogging, shoulder surfing, and unsecure networks. In addition, we claim that the protocol is a two-factor authentication scheme, meaning that both knowledge of the password and ownership of the device are strictly necessary to log in. We discuss the protocol's security with respect to each of these issues below.

Secure Password Storage

Password systems are generally implemented by storing hashed passwords. This requires the user to directly send the server his password, and unfortunately, in the real world passwords are often leaked. The proposed protocol avoids this issue by never giving the server the users' passwords in the first place. Instead, the server is provided with public keys, which as the name implies can be made public without a loss of security. In this sense, the proposed protocol is an improvement over hashing based password schemes because it allows for more secure password storage.

Two Factors are Necessary to Log in

It is clearly true that if the user has both knowledge of the password and ownership of the trusted device, the above protocol can be used to successfully log in. We want to make sure, however, that only a user with both of these factors can successfully log in.

Whenever the server receives a message regarding a login attempt associated with a particular username, the server uses the public key that it has stored to verify that the message is signed with the secret key associated with the trusted device associated with that username. If verification of the signature fails, the server simply ignores the message. Thus, since a successful login attempt requires a lot of communication between the server and the user's trusted device, a successful login attempt is impossible without knowledge of the trusted device's secret key.

Since we trust the trusted device to be secure, we are assuming that the adversary cannot steal the secret key from the trusted device. Since encryption of messages and signing of messages can be accomplished without compromising the security of the secret key, the adversary cannot gain the trusted device's secret key by eavesdropping on legitimate login attempts made by the user [**?**]. Thus, we conclude that only the trusted device has access to the trusted device's secret key, and so only with ownership of the trusted device can a login attempt succeed.

We also know that the zero knowledge proofs have only a negligible probability of success if the prover (the trusted device in this case) does not actually have access to the knowledge; this is the soundness property of zero knowledge proofs. Thus, a login attempt can succeed only if the trusted device has access to the user's secret key. However, since the trusted device always deletes all references to the user's password or secret key between login attempts, the only way for the trusted device to have the user's secret key is for the user's password to be entered at the start of the login attempt. Thus, in addition to ownership of the trusted device, knowledge of the password is required in order to successfully log in.

We see that the protocol described is a two factor authentication scheme and therefore has the extra layer of security associated with such a scheme.

Keylogging

One of the assumptions made is that the untrusted device is being keylogged. In that case, we see that over the course of a login attempt, the data leaked consists only of the two tokens and the user's username.

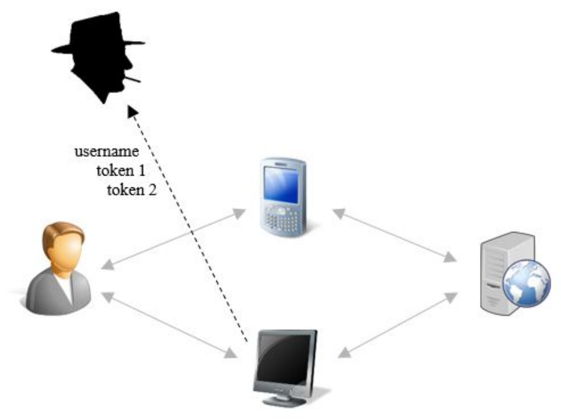


Figure 7: over the course of a login attempt, this data could be leaked to the adversary via keylogging

The protocol does not rely on keeping the user's username secret, and in fact it is expected that a user's username will be publicly known. The adversary can gain no advantage from learning that piece of information.

The two tokens are both randomly generated and provide the adversary with no information on the user's password of the trusted device's secret key. The adversary can gain no information from learning the two tokens. Beyond that, however, the adversary also cannot make use of the tokens in any way. If the adversary attempts to use the second token to log in then he will not be successful; this is because only the unique computer that is "halfway logged in," the untrusted device that the user is using to login, can finish the login attempt. If the adversary attempts to use the first token to log in, then he will successfully log in half way. However, in that case the user will be unable to log in half way, and as a result the log in attempt will be canceled without the adversary having a chance to keylog or eavesdrop a second token with which he can conclude his login attempt.

Clearly, the adversary can gain no advantage from learning the values of the two tokens.

Unsecured Network

Assuming that the network is unsecured, we want to know whether the adversary can learn anything useful by eavesdropping on the network. Eavesdropping on the network allows the adversary access to communications between the server and the trusted device and between the server and the untrusted device; in effect, the adversary is able to read everything sent and received by the server.

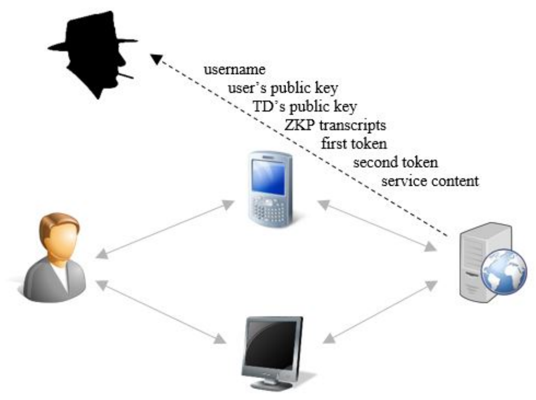


Figure 8: the adversary can access all of the communication that the server participates in by eavesdropping on an unsecure network

During the setup phase, the server and the trusted device agree on a username, and the trusted device sends the server both public keys. As mentioned before, none of the security of this protocol relies on keeping any of these three data secret, so the adversary gains nothing by eavesdropping on this phase.

During a login attempt, the trusted device always digitally signs messages to the server with the device's private key. Viewing these signatures does not help the adversary because digital signatures are designed to not be forgeable [**?**].

In the protocol, after first contacting the server, the trusted device uses a Zero Knowledge Proof to establish its knowledge of the user's password. The zero-knowledge property of zero-knowledge proofs allows this conversation to occur on an unsecure network while revealing to any observer (such as the adversary) only that the user's trusted device has been given the user's password.

Next, the server sends an encrypted token to the trusted device. We can select an encryption method that is IND-CCA2 secure, so that even if the adversary learns both the token and its encryption, the secret key used remains secure. Even if the encryption fails to hide the value of the token, tokens can only be used once, so a use by the adversary would be obvious to the user. Thus, the adversary must either watch passively as the user uses the token (thereby making knowledge of the token completely irrelevant) or actively attempt to use the token first, in which case the user will abort the login attempt. In either case, the adversary gains nothing in this step.

Finally, another round of ZKP and another transmitted token round off the login attempt. The Zero Knowledge Proof remains equally secure in this part of the protocol as it was the first time, and the token, even if its value is decrypted, is useless to the adversary because it can only be used with the user's untrusted device.

After the login attempt succeeds, the adversary will be able to eavesdrop on the content that the user is trying to access. While this is not a good thing, it is to be expected in a situation as hostile as we are dealing with, and services using this protocol should be set up so as not to reveal any sensitive information after login.

Shoulder Surfing

Exactly the same way that eavesdropping on the network amounts to reading all of the server's communications, shoulder surfing, in the worst case, amounts to viewing all of the user's communications.

Over the course of the protocol, shoulder surfing can reveal the user's username and the two tokens used for this particular login attempts. As already mentioned, none of this information is of any use to the adversary: the username is already public, and the protocol only generates a second token if the first is not stolen; stealing the second token can only be of use to the adversary in exactly the situation in which the token is not generated.

The real vulnerability here is that a shoulder surfer can see the user type his password into the trusted device. This would be a serious issue, but there are several mitigating factors.

First of all, even if the user's password is stolen, the fact that this is a two-factor authentication scheme allows the user some additional security. If the user suspects that his password has been stolen, he should change to a different password as quickly as possible,

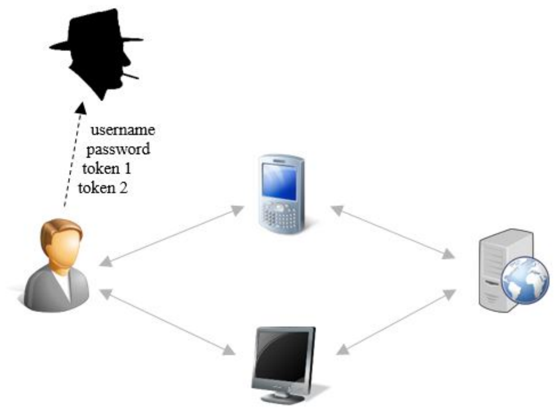


Figure 9: in the worst case, shoulder surfing allows an adversary to see everything that the user enters into any device and everything that the devices inform the user of

but as long as he still has his trusted device, the user is safe from the adversary accessing his account.

The second mitigating factor is that shoulder surfing is difficult to do without being noticed and can be avoided, as far as the trusted device is concerned, with a little effort. The untrusted device does not necessarily belong to the user; if it is in a public place, the adversary can spend time and effort in advance on setting up and hiding a camera which has a good view of the untrusted device. With the trusted device, the adversary does not have this option; since the trusted device belongs to the user, its exact location at the time of use will not be known ahead of time, and as a result the adversary has a much more difficult time successfully shoulder surfing. Taking this to its natural conclusion, a concerned user can simply cover his device (with a piece of clothing for example) while entering his password and severely restrict the adversary as a result.

Vulnerabilities

We have described several key ways in which the protocol described is secure, but unfortunately there are vulnerabilities that it is susceptible to. Before implementing this protocol, it is key to consider the vulnerabilities described below (as well as any other plausible breaches of security) and protect against them in some way.

Trusting the Trusted Device

We assume in analyzing this protocol that the trusted device is uncompromised. If this is not the case, depending on the details, we can run into a range of different problems.

In the worst case, the adversary has complete access to everything on the device. Even if the adversary has access only to the memory of the device, he is able to steal enough data to successfully log in using the user's account. In order to log in under a particular username, one needs knowledge of the user's password and ownership of the user's trusted device. The adversary can steal the user's password during any log in attempt, since for the duration of that attempt the user's password is stored on the device. The adversary can also steal the trusted device's secret key at any time, since it is also stored on the device, allowing him to prove ownership of the user's trusted device without actually owning it.

A less extreme case is that the trusted device is compromised, but not to the extent of giving the adversary full access. One possibility is that the device is keylogged. Keylogging the device allows the adversary direct access to the user's password. Given that fact, using this scheme with a keylogged trusted device is no better than using a more standary two factor authentication scheme where the username and password are entered directly into the (presumably keylogged) untrusted device.

Even without the adversary intentionally taking action to compromise the trusted device, certain issues with the device can cause problems with this scheme. For example, it could be that when this protocol is implemented, user password data is not perfectly removed between login attempts. This is a very plausible situation, even without malicious intent on the part of the person implementing this scheme, since it is possible for the data to be recoverable from the device's memory even after the data has been deleted. The problem in this situation is that logging in no longer requires two factors: possession of the device is still necessary, but now possession of the device also directly grants knowledge of the user's password. With only one factor, the protocol becomes much less secure; simply stealing the user's trusted device is sufficient to access their account.

Denial Attack

In our analysis above we assume that an adversary's intent is to access the user's account. However, this might not be the case.

For example, it could be that the adversary simply intends to be as much of a nuisance to the user as possible. Unfortunately, an adversary with all of the resources we have granted him can be a very effective nuisance. By stealing the first token (which can be done several ways including shoulder surfing), the adversary can log in halfway himself rather than allowing the user to do so. This forces the user to restart the log in attempt. By doing this repeatedly, the adversary can effectively deny the user access to their own account.

Active vs Passive Attacks

Another assumption we have made is that the adversary's control of the untrusted device and of the unsecured network are passive. All of our analysis so far interprets the fact that the untrusted device is untrusted to mean that the device is keylogged, and interprets the fact that the network is unsecured to mean that the adversary can eavesdrop on the network. However, it could be the case that the adversary has more control over both of these components.

Here is an extreme example of what the adversary could do: the adversary could allow the user to log in on the untrusted device without interference, at which point the adversary would simply take command of the untrusted device (which is logged into the users account) away from the user. As you can see, a user must trust the untrusted device at least some minimum amount before agreeing to log in on that device.

If the adversary has control over the network (as opposed to just eavesdropping capability), he can organize a very similar attack, even without having control over the untrusted device.

The adversary does this by organizing a man in the middle attack which places his own device between the untrusted device and the server. In effect, the user, believing himself to be logging the untrusted device into his account will actually be logging the adversary's device in instead. He will be unable to tell the difference (until the login attempt is complete) because the untrusted device will display a copy of what the adversary's device displays: a computer in the middle of logging in.

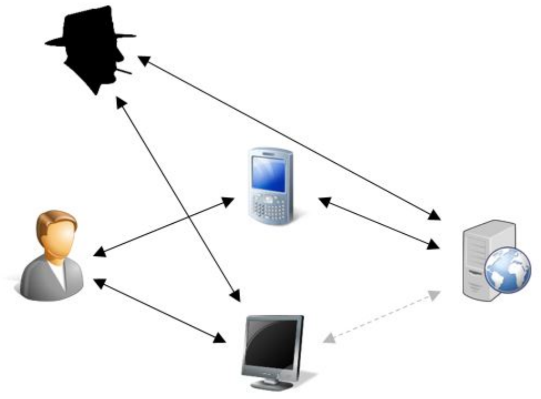
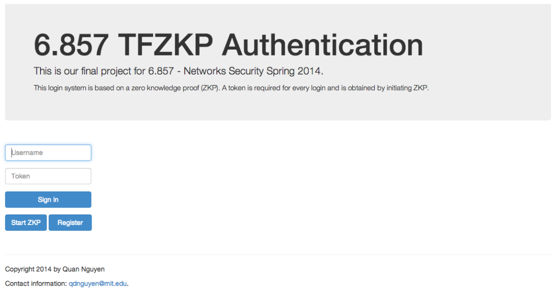


Figure 10: an adversary can fool the user into logging the adversary's device into the user's account by hijacking the signal between the server and the untrusted device

Implementation

Our implementation was in the form of a web frontend, hosted on Heroku. The backend was written in Python with Flask and all of the zero knowledge computation was done in Javascript on the trusted device and Python on the backend server. There were twenty rounds of zero knowledge proof done with our protocol (small enough to make the login process time efficient) and our prime used was 1000667 with generator $G = 98$.

In a complete implementation of the protocol described, the user's trusted device would use an app to interface with the server. In our more simplified implementation, we only use a website. In the simplified system, setting up an account involves accessing the website through the trusted device and registering for an account. The initial page of the website is shown below in figure 11, and the registration page below that in figure 12.


6.857 TFZKP Authentication

This is our final project for 6.857 - Networks Security Spring 2014.

This login system is based on a zero knowledge proof (ZKP). A token is required for every login and is obtained by initiating ZKP.

Copyright 2014 by Quan Nguyen

Contact information: qdnguyen@mit.edu.

Figure 11: Login page

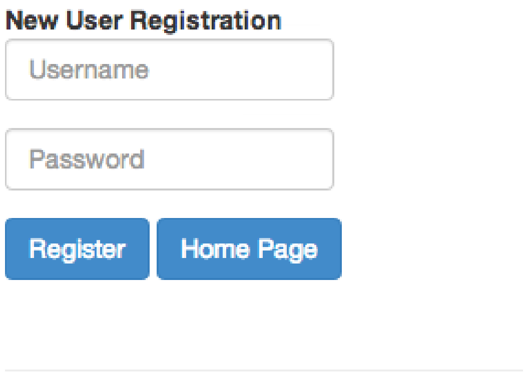


Figure 12: Registration page

Using this registration page through the trusted device allows a user to create an account associated with a username, password, and trusted device's secret key. The username and password are provided and remembered by the user. The trusted device's secret key is generated at random and stored in a cookie on the trusted device. Unfortunately, this is not the most secure storage, but we allow this in our simplified implementation.

A more important hole that this difference introduces is based on the fact that browsers occasionally clear cookies. In our case, this means that the user's trusted device will occasionally and almost spontaneously stop being recognized as that user's trusted device. We did not implement a fix, but an easy one exists. Users of this purely web-based system would be provided an opportunity to reset their authentication via email. This would be much like the standard "reset password by email" option for normal password schemes. The user's email would be sent a link that could be used to essentially redo the registration phase. In a situation where the cookie storing the trusted device's secret key is removed, the user would simply need to re-set-up the device as the trusted device by resetting their authentication data.

After a user is registered and wishes to log in, the protocol requires them to start the zero knowledge proof process through the trusted device and to generate a token which the user can then use to log in on the untrusted device. The page shown in figure 13 is the page that the user accesses with his trusted device.

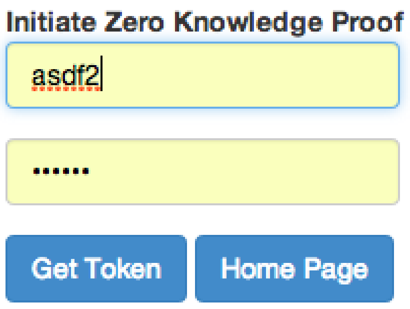
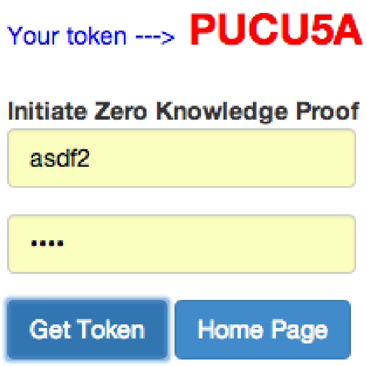


Figure 13: Get ZKP token page

Once the server is convinced of the trusted device's knowledge of the user's password, it sends the user his token via the webpage on the trusted device. The token is of length 6 because that's small enough for the user to easily input from a trusted device (like a phone) into a separate computer, but large enough to remain reasonably secure. This is shown in figure 14.


Your token ---> PUCU5A

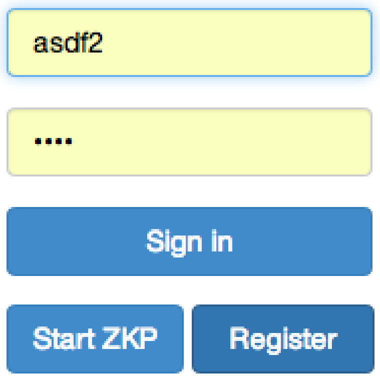
Initiate Zero Knowledge Proof

asdf2

••••

Figure 14: Received first token page

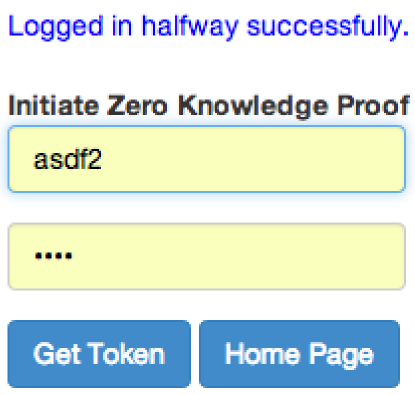
The user then inputs this token into the untrusted device along with his username. This can be seen in figure 15.


asdf2

••••

Figure 15: Sign in with token page

After that, the user's login is halfway successful. According to the protocol, the user must generate another access token to prevent shoulder surfers from simply stealing the first and accessing the account. Requiring two tokens at different steps of the process ensures that only one untrusted device is used during the login process.


Logged in halfway successfully.

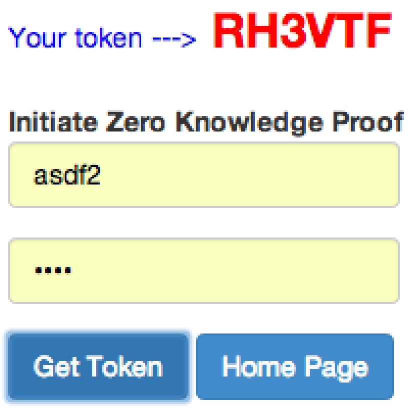
Initiate Zero Knowledge Proof

asdf2

••••

Figure 16: Login halfway successful page

The second token is generated on the trusted device exactly as before.


Your token ---> RH3VTF

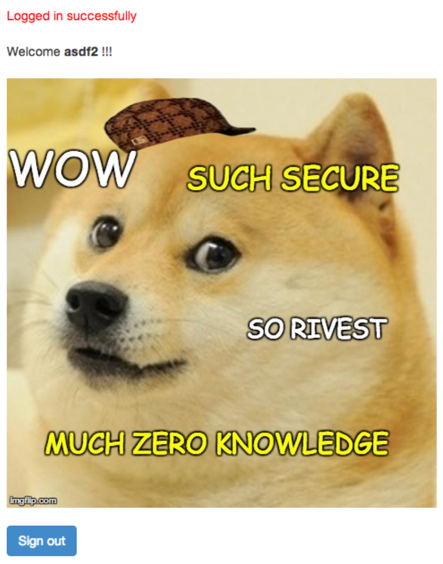
Initiate Zero Knowledge Proof

asdf2

••••

Figure 17: Received second token page

Entering the second token results in a successful login on the untrusted device, as seen in figure 18.



Logged in successfully

Welcome **asdf2** !!!

Figure 18: Login successful page

References

[1] David Chaum, Jan-Hendrik Evertse, Jeroen van de Graaf, René Peralta, *Demonstrating Possession of a Discrete Logarithm Without Revealing it*. Advances in Cryptology — CRYPTO' 86, Lecture Notes in Computer Science Volume 263, 1987, pp 200-212.

[2] Jeremy Clark, *Elgamal and CPA-Security, Zero-Knowledge*. Concordia Institute for Information Systems Engineering, Scribe Notes, March 14, 2013.

[3] Oded Goldreich, Yair Oren, *Definitions and properties of zero-knowledge proof systems*. Journal of Cryptology, 199424, Volume 7, Issue 1, pp 1-32.