Improved User Authentication in Off-The-Record Messaging

Chris Alexander Ian Goldberg

David R. Cheriton School of Computer Science
University of Waterloo
Waterloo, ON, Canada N2L 3G1
{cialexan,iang}@cs.uwaterloo.ca

ABSTRACT

Instant Messaging software is now used in homes and businesses by a wide variety of people. Many of these users would benefit from additional privacy, but do not have enough specialized knowledge to use existing privacy-enhancing software. There is a need for privacy software to be easy to understand, with complicated cryptographic concepts hidden from the user. We look at improving the usability of Off-the-Record Messaging, a popular privacy plugin for instant messaging software. By using a solution to the Socialist Millionaires' Problem, we are able to provide the same level of privacy and authentication as in older versions of OTR, but we no longer require that the user understand any difficult concepts such as keys or fingerprints.

Categories and Subject Descriptors

K.4.1 [**Management of Computing and Information Systems**]: Public Policy Issues—*Privacy*; E.3 [**Data**]: Data Encryption; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Authentication*; H.4.3 [**Information Systems Applications**]: Communication Applications—*Computer conferencing, teleconferencing, and videoconferencing*; C.2.2 [**Computer-Communication Protocols**]: Network Protocols—*Applications*

General Terms

Security, Algorithms, Human Factors

Keywords

Authentication, fingerprints, instant messaging, socialist millionaires' protocol

1. INTRODUCTION

As pointed out by Whitten and Tygar [15], it is very difficult to develop security software for the general population. Many of the basic tools used, including keys, fingerprints and digital signatures, are difficult for an uninitiated user to understand. Further, when faced with such puzzling terms, many users will sooner ignore the security features of a program than take the time to determine their proper use.

This is especially true for Instant Messaging (IM) software. IM clients allow users to chat with other known users online, and are easy enough for children to understand and use effectively. As such, they have become a very popular way for people to communicate both at home and, increasingly, at work. They are used by a wide variety of people, most of whom know very little, if anything, about security and privacy, and it is unlikely that all such people will learn the basics of public-key cryptography before installing their first IM client. Therefore there exists a need to provide secure software that protects people's privacy, while at the same time hiding cryptographic details from the user, presenting instead a user interface that is simple and intuitive.

There are serious consequences if this need is not met. As an example, consider the case of a man-in-the-middle (MITM) attack. Here, two honest parties, Alice and Bob, wish to chat with each other. Unbeknownst to them, a third party, Eve, wishes to learn the contents of their conversation. So when Alice and Bob establish a connection, Eve simply ensures that they both establish connections with herself instead of each other. Eve may now pass messages from her connection with Alice to her connection with Bob, so no messages go missing, but they are all read by Eve before reaching their intended destination.

IM networks have an inherent weakness towards MITM attacks. Since popular IM protocols tend to route all messages through a central server, MITM attacks from this central point can be very effective. Even if messages are properly encrypted, such attacks can succeed. For example, using Trillian's SecureIM feature encrypts messages but does not perform any authentication at all, so MITM attacks are not detected. Another good example is the Jabber protocol, where messages are only encrypted between a client and the central server, not from client to client. Thus an attack at the server would still recover the contents of every message.

That said, there are good solutions available that prevent MITM attacks and many others. One such solution, called Off-the-Record Messaging (OTR) [1], is freely available online and compatible with many of the popular IM protocols used today. Unfortunately, it requires users to verify each other's fingerprints in real time, which is outside of the comfort zone of many others. Other tools are no better. PGP requires even more knowledge of public-key cryptography than OTR, and is difficult for new users to understand [15]. In this paper, we offer an improvement to OTR that would not affect its security and privacy properties, but allow it to be used properly with a much smaller amount of prior knowledge.

Section 2 explains the goals and history of the OTR protocol,

WPES'07, October 29, 2007, Alexandria, Virginia, USA.

Section 3 explains the basic usability shortcoming of OTR thus far, and Section 4 gives the protocol used to address this shortcoming. Section 5 examines how to use the protocol securely, Section 6 describes the actual modification of OTR and Section 7 offers our conclusions.

2. OTR MESSAGING PROTOCOL

In this section we describe the evolution of the Off-the-Record Messaging protocol, from its inception to the present day.

2.1 Original OTR Protocol

The original OTR protocol was presented by Borisov, Goldberg, and Brewer in 2004 [1]. It was motivated by the idea of two people, say Alice and Bob, conversing face-to-face in a private room. In this case, Alice may be assured that no one else may hear what she says to Bob. She may also be assured that there is no evidence of the conversation outside of Bob's memory, so Bob will be unable to prove to anyone what she said. Alice is then free to say whatever she wishes, without having to worry about her words later being used against her.

Today, many social and business conversations occur not in person, but over the Internet using IM. OTR was developed as a way to give the same level of privacy to an IM conversation as to a face-to-face one.

To begin with, OTR uses a Diffie-Hellman (DH) key exchange to establish a shared secret between Alice and Bob [5]. After this exchange, they can use their shared secret to encrypt their messages, preventing any adversary from learning about their conversation or launching a man-in-the-middle (MITM) attack. However, a MITM attack is still possible during the DH key exchange itself. To prevent this, Alice and Bob sign their DH key exchange messages using long-lived public keys. If Alice and Bob possess public/private key pairs (v_A, s_A) and (v_B, s_B) respectively, then the key exchange runs as follows:

$$\begin{aligned} A \to B &: Sign_{s_A}(g^x), v_A \\ B \to A &: Sign_{s_B}(g^y), v_B \end{aligned}$$

If Bob knows Alice's public key v_A in advance, then he can check that the first message he receives does indeed come from Alice, and vice versa. Then both parties can compute the shared secret as g^{xy} and use it to communicate securely.

At this point Alice and Bob may begin sending each other encrypted messages. In order to limit the amount of information that is compromised if an adversary determines the shared key, Alice and Bob re-key as frequently as possible. Every time that Alice receives a message from Bob, she generates a new DH exponent and adds it to her next outgoing message. Bob does the same, and pairs of these new exponents are used to create new DH keys. Since the keys change so quickly, key IDs are used to make sure that both parties can identify which pair of exponents were used in any given message. As new keys are created, old keys are securely erased, rendering any messages signed under the old keys unreadable even to Alice and Bob. Since the keys consisted of randomly chosen DH exponents, they cannot be recreated by an attacker at a later date, regardless of how much other information is known to the attacker. This procedure gives OTR the property of perfect forward secrecy (PFS), ensuring that future key compromises cannot reveal the contents of old messages.

With this machinery in place, it should be infeasible for an adversary to learn the contents of messages passed between Alice and Bob. However, it is still possible for an adversary to change the contents of messages in transit. In order to detect this, OTR messages are authenticated using SHA1-HMAC [13]. The MAC key used is a hash of the decryption key for that message. This way, anyone who can decrypt the message can also forge a correct MAC tag, but an adversary unable to decrypt messages cannot.

The last desirable feature that was put into OTR was deniability. So far, the protocol ensures that only Bob can read Alice's messages, but we have not discussed the situation where Bob attempts to prove the contents of the conversation to a third party. First of all, simply by using symmetric encryption and HMAC, OTR guarantees that all proper messages were generated by someone who knew the secret key; that is, either Alice or Bob. On its own, this prevents Bob from proving anything, as he could simply have produced all of the messages himself. However, OTR goes beyond this.

The specific encryption scheme used is AES in counter mode, which is a stream cipher. This means that an adversary may alter a bit of a ciphertext to cause a corresponding change to that bit in the plaintext. Also, after a key is deleted, the corresponding MAC key is published. This means that after the fact, anyone could alter a ciphertext to be anything they wished and then generate a valid MAC for it. This expands the deniability of Alice's messages; as soon as a MAC key is published, any message using that key could have been generated not only by either Alice or Bob, but also by anyone casually observing their communication. Any link between Alice and the messages she sends under OTR is thus broken.

2.2 Attack on OTR version 1

About a year after the original publication, Di Raimondo, Gennaro, and Krawczyk [4] found an attack on OTR version 1. They pointed out that Diffie et al.'s identity misbinding attack [6] would work on the initial DH key exchange in OTR. This attack allows an adversary Eve to interfere with the initial key exchange in such a way that Alice and Bob still reach the same key at the end of the protocol, but Alice believes that she is talking to Bob while Bob believes that he is talking to Eve.

To do this, Eve runs a MITM attack, starting simultaneous conversations with both Alice and Bob. Messages from Alice to Bob are replaced with identical messages signed by Eve, while messages from Bob to Alice carry their original signatures. The exchange given in [4] is:

$$\begin{aligned} A \to E &: g^x, Sign_{s_A}(g^x), v_A \\ E \to B &: g^x, Sign_{s_E}(g^x), v_E \\ B \to E &: g^y, Sign_{s_B}(g^y), v_B \\ E \to A &: g^y, Sign_{s_B}(g^y), v_B \end{aligned}$$

Here, Alice still receives g^y signed by Bob and correctly assumes that she is talking to him. Bob, on the other hand, receives g^x signed by Eve and starts out assuming that he is talking to Eve when he is actually talking to Alice.

Several existing protocols that defeat the identity misbinding attack were suggested as possible improvements to OTR. We describe here the first such protocol, a version of SIGMA [12]. Here, each party waits until the shared secret g^{xy} is determined, and then sends an encrypted message identifying themselves. Since Eve does not know the shared secret, she should be unable to continue her MITM attack, and both Alice and Bob correctly determine each other's identities. The exchange given in [4] is:

$$\begin{aligned} A \to B &: g^x \\ B \to A &: g^y \\ A \to B &: A, Sign_{s_A}(g^y, g^x), MAC_{K_m}(0, A), v_A \\ B \to A &: B, Sign_{s_B}(g^x, g^y), MAC_{K_m}(1, B), v_B \end{aligned}$$

Again, the MAC key K_m is a hash of g^{xy}, so it is unknown to Eve. By including this MAC, SIGMA prevents the identity misbinding attack.

2.3 OTR version 2

OTR version 2 was released in 2005. The largest change in version 2 was the reworking of the initial authenticated key exchange (AKE). In response to the attack mentioned above, the AKE was changed to a SIGMA variant, as suggested. Instead of using the exact protocol recommended in [4], however, OTR adopted a variant that also hides the public keys of the participants from passive adversaries. Where the public keys were formerly sent in the clear, they are now encrypted using the DH shared secret.

The protocol still works by first establishing an unauthenticated DH channel and then performing the authentication inside that channel. The channel itself uses a 64-bit secure session id based on the shared secret, which is short enough to be vulnerable to brute-force attacks. As a result, an initial commitment is used to ensure that neither party can base their choice of g^x on the other party's value of g^y. The first few steps in the AKE are then:

Alice:

1. Randomly selects a 128 bit value r
2. Randomly selects a 320 bit value x
3. Sends $AES_r(g^x)$, SHA-256(g^x) to Bob

Bob:

1. Randomly selects a 320 bit value y
2. Sends g^y to Alice

Alice:

1. Computes $s = (g^y)^x$
2. Sends r to Bob

Bob:

1. Decrypts g^x using r
2. Verifies that g^x agrees with SHA-256(g^x) received earlier
3. Computes $s = (g^x)^y$

The purpose of r in the above steps is to satisfy an engineering restriction: many IM protocols enforce a maximum size on messages. Ideally, Alice's first step 3 would simply send SHA-256(g^x) to Bob as a commitment, and Alice's second step 2 would send g^x to open the commitment. However, Alice's second step 2 will be combined with her first message in the steps below, which leaves no room to send all of g^x. So instead, we hide the value of g^x by encrypting it with r and sending it in the first message, and reveal r in Alice's second message. Note in particular that the encrypted version of g^x does *not* act as a commitment on its own; the hash is still performing that function.

At this point, the shared secret s has been established. To perform authentication, Alice and Bob use SHA-256 hashes of s with various prefixes to determine a series of MAC and AES keys. These keys are then used to encrypt and verify the integrity of the information exchanged during the rest of the AKE. If (v_A, s_A) and (v_B, s_B) are Alice and Bob's public/private key pairs, then the protocol continues:

Alice:

1. Computes MAC keys a_1, a_2, b_1, b_2 and AES keys a_3 and b_3
2. Selects $keyid_A$, a serial number to associate with g^x
3. Computes $M_A = MAC_{a_1}(g^x, g^y, v_A, keyid_A)$
4. Computes $X_A = v_A, keyid_A, sign_{s_A}(M_A)$
5. Sends $AES_{a_3}(X_A), MAC_{a_2}(AES_{a_3}(X_A))$ to Bob

Bob:

1. Computes MAC keys a_1, a_2, b_1, b_2 and AES keys a_3 and b_3
2. Uses a_2 to verify $MAC_{a_2}(AES_{a_3}(X_A))$
3. Uses a_3 to decrypt $AES_{a_3}(X_A)$ and obtains $X_A = v_A, keyid_A, sign_{s_A}(M_A)$
4. Computes $M_A = MAC_{a_1}(g^x, g^y, v_A, keyid_A)$
5. Uses v_A to verify $sign_{s_A}(M_A)$
6. Selects $keyid_B$, a serial number to associate with g^y
7. Computes $M_B = MAC_{b_1}(g^y, g^x, v_B, keyid_B)$
8. Computes $X_B = v_B, keyid_B, sign_{s_B}(M_B)$
9. Sends $AES_{b_3}(X_B), MAC_{b_2}(AES_{b_3}(X_B))$ to Alice

Alice:

1. Uses b_2 to verify $MAC_{b_2}(AES_{b_3}(X_B))$
2. Uses b_3 to decrypt $AES_{b_3}(X_B)$ and obtains $X_B = v_B, keyid_B, sign_{s_B}(M_B)$
3. Computes $M_B = MAC_{b_1}(g^y, g^x, v_B, keyid_B)$
4. Uses v_B to verify $sign_{s_B}(M_B)$

At the end of this protocol, Alice and Bob possess a shared secret s and have exchanged key IDs so they can begin the constant rekeying process described in 2.1. Alice also knows Bob's public key v_B, and is convinced that Bob knows the corresponding private key s_B. Bob has a similar assurance about Alice. Further, all of these values were encrypted, so they are concealed from passive adversaries.

3. AUTHENTICATION PROBLEM

Although the AKE in OTR is quite good in theory, it still suffers from a practical drawback. In all of the schemes mentioned so far, it is assumed that Alice and Bob know each other's public keys before the AKE begins. If they do not, then Eve can pretend to be Bob and use her own key pair for signing, and Alice will have no way to spot the deception.

This makes MITM attacks launched from a central IM server particularly effective. If Eve controls the IM server, then she can impersonate Alice to Bob and Bob to Alice for every conversation over that server, learning the contents of every encrypted message sent. Unless Alice and Bob know each other's public keys, this attack will never be detected. In fact, there exists a plugin for Jabber servers to automatically launch MITM attacks on OTR conversations, called `mod_otr` [8].

3.1 Solving the authentication problem

To prevent this attack, each OTR user maintains a store of their buddies' public keys. If a buddy starts a new conversation using a familiar public key, then OTR finds the match in the store and authenticates automatically. If a match is not found in the store, for example during the first conversation with a new buddy, the user is prompted to verify that the key is correct. In this case OTR displays the fingerprint (SHA-1 hash) of the unfamiliar key, and asks the user to verify its authenticity out-of-band.

While this approach certainly works, it has several drawbacks. First, it requires that users understand the basics of public-key cryptography; if the user does not know what a fingerprint is, asking them to verify one can only lead to confusion. Once the user gets past the initial verification, however, all the messy cryptographic details are hidden again and the user can benefit from OTR without any idea how it works. Ideally, we would like the buddy verification to be every bit as easy to use as the rest of OTR.

Second, the current OTR framework requires that fingerprint verification be done in real time. Since the key pairs used are specific to OTR, users will generate new keys every time they install OTR on a new computer, or reinstall it on their present one. This means that even for well-known buddies, occasional and unpredictable fingerprint verifications are necessary. And since the fingerprints are essentially random, there is nothing that the user can do ahead of time to speed up the verification. We would like to allow almost all the work of verification to be done offline, at the users' convenience, to make the process less onerous for them.

If we are to hide the potentially confusing fingerprints from OTR users, we must find a secure yet intuitive substitute. Note that any given Alice and Bob starting an IM session already have many shared secrets that have nothing to do with cryptography. For example, if they meet in person and talk about OTR, they can easily pick a shared password with which to identify each other later. Even if they don't pick a password, they know many details that would be difficult for an attacker to guess. For example, where did Alice and Bob first meet? Who introduced them? When did they last meet in person?

A simple way to leverage this information would be for Alice and Bob to simply ask each other some questions of this form after the AKE has taken place and the conversation is encrypted. If each answers to the other's satisfaction, the conversation may continue. This has severe privacy concerns, however: if Eve is launching a MITM attack and she asks the first verification question, she can force Alice to reveal an arbitrary secret fact known only to Alice and Bob. She may be detected in doing so, but she will still have gained important information of her choosing. We must be more careful to use the shared information in a way that will not reveal it to attackers.

4. SOCIALIST MILLIONAIRES' PROTOCOL

Fortunately, a very similar problem has already been studied in great detail. Our problem is simply a rephrasing of the Socialist Millionaires' Problem, in which two millionaires wish to know whether they happen to be equally rich [10]. This is itself a variant of Yao's original Millionaires' Problem [16], where the millionaires wish to know who is richer without revealing any other information about their wealth.

An efficient solution to the Socialist Millionaires' problem was developed in 2001 by Boudot, Schoenmakers, and Traoré [2]. Under the Decision Diffie-Hellman assumption, their protocol allows two millionaires to learn only whether their fortunes are equal, while revealing nothing about the fortunes themselves.

We adapted this Socialist Millionaires' Protocol (SMP) to work with OTR. Here, instead of comparing money, we are comparing the shared information between Alice and Bob. Now an attacker Eve can only learn information from Alice if she is correctly able to guess the answer on the first try—otherwise, the protocol terminates and she learns nothing. Of course, Eve could still attempt to impersonate Alice to Bob and vice versa, passing SMP information along unchanged in an attempt to get it to match. For this reason, OTR actually compares a SHA-256 hash of the session ID, the two parties' fingerprints, and the original secret between Alice and Bob. Any MITM attack made by Eve will cause Alice and Bob to hash different fingerprints, and so the protocol will simply fail.

Another important property of this protocol is that if Alice and Bob's secrets are different, then neither Alice, Bob, nor any observer learns any information about those secrets, save that they are different. This allows Alice and Bob to use secrets with very low entropy, since each guess by a MITM would have to be performed online with either Alice or Bob.

For efficiency, when we altered the SMP for OTR we reduced the number of messages as much as we could, and we omitted the optional sections relating to fairness. Our modified protocol is described below.

4.1 Setup

All computations in this protocol are done in a group G of large prime order q. The exact group used is the 1536-bit modulus group defined in RFC 3526, also known as Diffie-Hellman Group 5 [11]. The generator g_1, known to both Alice and Bob before the protocol begins, is equal to 2. Alice knows a secret x and Bob knows a secret y, both of which are elements of $\mathbb{Z}_q$. The goal of the protocol is to determine whether $x = y$.

A variety of zero-knowledge proofs are also required in the protocol. Simply put, a zero-knowledge proof allows Alice to demonstrate the correctness of a certain fact to Bob without revealing any additional information. For example, in the context of SMP, Alice must at times demonstrate that a certain exponent used in her calculations has not changed throughout the protocol, without revealing the value of the exponent itself. All values passed between Alice and Bob are accompanied by these zero-knowledge proofs that show that the protocol is being followed faithfully. For a description of the specific proofs used, see section 2.3 of [2] or the shorter explanation given in the appendix; all of the proofs used are simple and efficient.

4.2 Generator Selection

Two additional generators, g_2 and g_3, are required for this protocol. Both are created through a DH exchange. Alice chooses $a_2 \in_R \mathbb{Z}_q$ and Bob chooses $b_2 \in_R \mathbb{Z}_q$. They then exchange $g_1^{a_2}$ and $g_1^{b_2}$ and each computes $g_2 = g_1^{a_2 b_2}$.

They then repeat this process, choosing new values a_3 and b_3 and exchanging $g_1^{a_3}$ and $g_1^{b_3}$ to get $g_3 = g_1^{a_3 b_3}$. Finally, Alice and Bob save the values of a_3 and b_3 used in the generation of g_3 for use later in the protocol.

4.3 Blinding x and y

Alice picks an $a \in_R \mathbb{Z}_q$ and computes $(P_a, Q_a) = (g_3^a, g_1^a g_2^x)$. Bob chooses a b and (P_b, Q_b) similarly. P_a, Q_a, P_b, and Q_b are then exchanged.

4.4 Revealing whether $x = y$

Alice uses her value of a_3 from 4.2 to compute $R_a = \left(\frac{Q_a}{Q_b}\right)^{a_3}$. Bob similarly computes $R_b = \left(\frac{Q_a}{Q_b}\right)^{b_3}$ and the values are exchanged. Alice and Bob may now compute $R_{ab} = R_a^{b_3} = R_b^{a_3}$.

Now both parties know that:

$$\begin{aligned} R_{ab} &= \left(\frac{Q_a}{Q_b}\right)^{a_3 b_3} \\ &= (g_1^{a-b} g_2^{x-y})^{a_3 b_3} \\ &= g_3^{a-b} g_2^{(x-y)a_3 b_3} \\ &= \left(\frac{P_a}{P_b}\right) \left(g_2^{a_3 b_3}\right)^{(x-y)} \end{aligned}$$

So to check whether $x = y$, they need only check whether $R_{ab} = \left(\frac{P_a}{P_b}\right)$. Note that no party knows the value of $\left(g_2^{a_3 b_3}\right)$, and that that value is a random generator of G. Therefore, if $x \neq y$, $\left(R_{ab} \cdot \frac{P_b}{P_a}\right)$ will be a random element of G, and if $x = y$, it will be 1. Since the randomness of $\left(R_{ab} \cdot \frac{P_b}{P_a}\right)$ does not depend on the entropy of x and y, this protocol works well even if that entropy is very small. Thus, even secrets such as common English words are acceptable.

5. SECURITY OF SMP

A full security analysis of the above protocol is given in [2], where it is shown that the zero-knowledge nature of the protocol means that attackers cannot learn anything about x and y apart from their equality, unless they can also solve the Decision Diffie-Hellman problem. Knowing this, if Alice runs the protocol, she may be certain that no attacker has learned any information about her secret, apart from whether it matches the attacker's online guess. In particular, the attacker cannot do an offline dictionary or brute-force attack against the secret, even if it has very low entropy. Therefore, every incorrect guess of the secret will cause an online execution of the SMP to fail; this will alert the users to the attacker's presence if the attacker does not successfully guess the secret in the first couple of tries.

Since the secrets used in OTR contain the fingerprints of both Alice and Bob, as well as the user input, an attacker may not simply relay SMP information between Alice and Bob in an attempt to get it to match. Therefore, after a successful run of the protocol, Alice knows that an attacker may neither learn her secret nor rely on a third party's knowledge of the secret. That is, for an attack to succeed, the attacker must actually know the secret.

Now if Eve wants to defeat the authentication mechanism, she has a couple of options. First, she may allow Alice to select a shared secret but try to learn the secret before it is used. Second, if the secret has not yet been selected, she may attempt to persuade Alice to use a secret which is already known to Eve. The difficulty of these tasks depends on the method that Alice and Bob use to determine their secrets.

5.1 Choosing a secret over a secure channel

Suppose that Alice and Bob choose their secret over a secure channel, for example, in person. In this case, Eve will have no way to influence the selection of the secret, nor any idea what secret has been chosen. At best, she can try to convince Alice to "remind" her of the secret, or to choose a new one instead. If Alice agrees, but does not wish to decrease the security of her secret, then she should deliver the new secret or the reminder over the original channel as well. Then Eve will be stuck in the same position as before.

The advantage of this method is that an arbitrary secret can be selected when Alice and Bob happen to meet, even if they do not have their OTR fingerprints with them at the time, or especially if one or both of them have never used OTR at all at that point.

On the other hand, if Alice decides to negotiate a new secret over an insecure channel, she should take care to follow the rules given in the next section.

5.2 Choosing a secret over an insecure channel

Although meeting in person is more secure, it may not be practical if Alice and Bob live far apart or rarely visit each other. Even if Alice and Bob can meet in person, or share another secure method of communication, it may be more expensive, time-consuming, or simply inconvenient, than using an insecure channel such as the unauthenticated OTR conversation itself.

Suppose that Alice and Bob are chatting online using OTR and decide to run the SMP, but have not previously selected a secret and possess no channel more secure than their current conversation. They can still select an appropriate secret in this case, but they will be more vulnerable to attack.

In this case, if Eve is attempting to impersonate Bob or run a MITM attack, she has a few options. The most direct is to attempt to use a secret that has been sent in the contents of the OTR conversation itself. However, the online instructions that accompany OTR specifically warn against sending the secret over the Internet, whether over IM or email. If Eve requests that Alice do so, Alice should become very suspicious of her.

Instead, Alice should send enough information over OTR to identify the secret to Bob, but not to a stranger. A suitable hint might be: "The secret will be the name of the movie we watched last week." In this case, if Eve is a stranger, she will not know the secret unless she happened to be present when they watched the movie. Likewise, a stranger would be unable to select a secret and give Alice a hint such as the one above, that is tied to a private experience.

At this point, the only way that Eve could attack Alice is if she had managed to learn some very specific details about Alice's life. Then she could attempt to choose a secret based on those details, and send Alice an appropriate hint, similar to the one above. To guard against this, Alice should make sure that she authenticates Bob using a secret of her own choosing; this can be done either by running the protocol twice, once with a secret chosen by her, and once with a secret chosen by the other party, or else by combining the two secrets into a single protocol run. Now if Eve wishes to attack Alice, a few details are insufficient; she must know quite a large amount of information to be able to guess a secret chosen by Alice instead of herself.

5.3 Security results

It is clear that establishing a secret over a secure channel will make Alice secure against all attackers, subject to the hardness of the Decision Diffie-Hellman problem at the heart of SMP. If she wishes to use a more convenient but less secure channel, she should never give out the secret itself over the channel, and she should make sure that at least one run of the protocol uses a secret of her own choosing. In this case, Alice will be secure against all automated attacks, as well as attacks from adversaries without an intimate knowledge of her experiences with Bob.

This is much more secure than the approach using fingerprints, which as `mod_otr` demonstrates, is often insecure against automated attacks by adversaries with absolutely no private knowledge about Alice [8], as users simply fail to properly authenticate the fingerprints over a separate channel.

Figure 1: The dialog box used to enter the SMP secret.

6. IMPLEMENTATION

6.1 Modifications

The original version of OTR was implemented as a plugin for the popular IM client gaim. As gaim has been replaced by pidgin [7] due to a trademark dispute with AOL [17], OTR naturally migrated to become a plugin for pidgin instead. OTR itself required only two major changes.

The first major change was the addition of message fragmentation. Most IM protocols have a maximum allowable size for any transmitted message, usually on the order of 1–2 kB. Since some of the messages exchanged during the SMP are larger than this, it was necessary to include a mechanism to fragment and reassemble any messages that would otherwise exceed the maximum size. This is invisible to the user, who will simply no longer experience "Message too large" errors.

The other major change was the addition of a Socialist Millionaires' authentication mechanism. The familiar fingerprint verification popup may still be accessed through an "Advanced" option, but the regular authentication mechanism is now the SMP. Selecting "Authenticate Buddy" from the menu brings up the dialog box shown in Figure 1.

Here the user does not need to understand any cryptographic ideas; they need only to understand the idea of a secret or a password. Instructions about when to authenticate and how to choose a reliable secret are available through the "What's This?" tab and on the main OTR website [9]. These instructions include the recommendations given in Section 5.

6.2 Performance

When implementing the SMP, messages were combined for efficiency reasons whenever possible. For example, Alice begins by sending her halves $g_1^{a_2}$ and $g_1^{a_3}$ of the generators, and Bob can not only reply with his own halves $g_1^{b_2}$ and $g_1^{b_3}$ of the generators, but he may also compute and send his values of P_b and Q_b. The compressed protocol takes 5 steps, with Alice executing odd numbered steps and Bob executing even numbered ones. Messages are sent during the first four steps of the protocol; step 5 merely checks proofs and computes the final result, but does not produce any values to transmit to Bob. On a computer with a 3 GHz processor running Linux, Alice's steps took a total of 1213 ± 3 ms of computation time, while Bob's steps took a total of 1212 ± 4 ms. That the times are the same for Alice and Bob is not surprising, as they perform essentially the same computations. Thus, the full protocol takes about 2.4 seconds of processing time plus the time to transmit four messages. A progress bar is displayed during the execution of the SMP in order to keep the user apprised of how much of the protocol has been completed.

7. CONCLUSION

We have implemented an enhancement for OTR which allows users to perform proper authentication without exposing them to terminology with which they are likely to be unfamiliar, such as "fingerprints". We expect this enhancement will make OTR more accessible to the general population, and easier to use to protect sensitive information.

As future work, we plan to perform a user study to evaluate our design. Such a study on the new elements of the user interface may be instructive, as it could identify refinements in structure or language that would make the authentication process even more straightforward and intuitive.

Acknowledgements

We would like to thank Steven J. Murdoch and the anonymous referees for their helpful suggestions for improving this paper. We would also like to thank the Natural Sciences and Engineering Research Council of Canada for supporting this research with a Discovery Grant and an Undergraduate Student Research Award.

8. REFERENCES

[1] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-Record Communication, or, Why Not To Use PGP. In *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society*, pages 77–84. ACM Press, October 2004.

[2] Fabrice Boudot, Berry Schoenmakers, and Jacques Traoré. A Fair and Efficient Solution to the Socialist Millionaires' Problem. *Discrete Applied Mathematics*, 111(1–2):23–36, 2001.

[3] David Chaum and Torben Pryds Pedersen. Wallet Databases with Observers. In *Proc. of Advances in Cryptology—CRYPTO '92, Lecture Notes in Computer Science 740*, pages 89–105, 1992.

[4] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Secure Off-the-Record Messaging. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, pages 81–89. ACM Press, November 2005.

[5] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, IT-22(6):644–654, 1976.

[6] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes and Cryptography*, 2(2):107–125, 1992.

[7] Sean Egan et al. Pidgin. http://pidgin.im/pidgin/home/. Accessed June 2007.

[8] Olivier Goffart. mod_otr — Man in the Middle module for Off-The-Record. http://ejabberd.jabber.ru/mod_otr. Accessed June 2007.

[9] Ian Goldberg and Nikita Borisov. Off-the-Record Messaging. http://otr.cypherpunks.ca/. Accessed June 2007.

[10] Markus Jakobsson and Moti Yung. Proving Without Knowing: On Oblivious, Agnostic and Blindfolded Provers. In *Proc. of Advances in Cryptology—CRYPTO '96, Lecture Notes in Computer Science 1109*, pages 186–200, 1996.

[11] Tero Kivinen and Mika Kojo. More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE). RFC 3526, http://www.ietf.org/rfc/rfc3526.txt, May 2003.

[12] Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' Approach to Authenticated Diffie-Hellman and Its Use in IKE Protocols. In *Proceedings of CRYPTO '03*, pages 400–425, 2003.

[13] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, http://www.ietf.org/rfc/rfc2104.txt, February 1997.

[14] Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, 1991.

[15] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999.

[16] Andrew Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[17] Ziff Davis Publishing Holdings Inc. AOL dispute forces GAIM to become "Pidgin". http://www.desktoplinux.com/news/NS3029376172.html. Accessed June 2007.

APPENDIX

A. ZERO-KNOWLEDGE PROOFS IN SMP

As mentioned above, the solution to the SMP that we use relies on a variety of zero-knowledge proofs. There are in fact three such proofs, associated with the three steps of the protocol given earlier.

Each proof requires one or more evaluations of a hash function which is diversified to prevent Alice and Bob from copying each other's proofs. In OTR, we use SHA-256 with a unique prefix. In the following, let $h_n(x)$ represent hashing x with prefix n, that is, $h_n(x) = \text{SHA-256}(n, x)$.

A.1 Generator Selection

When Alice sends Bob her half of g_2 as $g_1^{a_2}$, she wants to assure him that she knows the corresponding value of a_2. To do this she uses Schnorr's protocol [14]. Alice selects $r \in_R \mathbb{Z}_q$ and computes $W = g_1^r$, $c = h_1(W)$, and $D = r - a_2 c \bmod q$. Alice sends the triple $(g_1^{a_2}, c, D)$ to Bob, who accepts only if $c = h_1(g_1^D (g_1^{a_2})^c)$.

When Bob sends his half of g_2 to Alice, he computes a similar proof using b_2 and h_2 in place of a_2 and h_1 respectively. This process repeats to generate g_3, this time using hash functions h_3 and h_4 and exponents a_3 and b_3.

A.2 Blinding x and y

Alice next selects an a and computes $(P_a, Q_a) = (g_3^a, g_1^a g_2^x)$. To prove that she has followed the protocol, she must allow Bob to verify that the same exponent a was used in computing both P_a and Q_a, and that she knows the secret x. We use Boudot, Schoenmakers and Traoré's extension to a protocol by Chaum and Pedersen to do so [2, 3]. Alice selects $r_1, r_2 \in_R \mathbb{Z}_q$ and computes $W_1 = g_3^{r_1}$, $W_2 = g_1^{r_1} g_2^{r_2}$, $c = h_5(W_1, W_2)$, $D_1 = r_1 - ac \bmod q$, and $D_2 = r_2 - yc \bmod q$. She then sends (P_a, Q_a, c, D_1, D_2) to Bob. Bob is convinced only if $c = h_5(g_3^{D_1} P_b^c, g_1^{D_1} g_2^{D_2} Q_b^c)$.

Bob behaves similarly except that he uses y instead of x and h_6 instead of h_5.

A.3 Revealing whether $x = y$

Finally, Alice computes $R_a = \left(\frac{Q_a}{Q_b}\right)^{a_3}$. She would like to prove to Bob that the value of a_3 was the same one that was used to compute g_3. For this we use the unmodified protocol from Chaum and Pedersen [3]. Alice selects $r \in_R \mathbb{Z}_q$ and computes $W_1 = g_1^r$, $W_2 = \left(\frac{Q_a}{Q_b}\right)^r$, $c = h_7(W_1, W_2)$, and $D = r - a_3 c \bmod q$. Alice sends (R_a, c, D) to Bob, who accepts only if $c = h_7\Big(g_1^D (g_1^{a_3})^c, \left(\frac{Q_a}{Q_b}\right)^D R_a^c\Big)$.

Bob does a similar calculation using b_3 and h_8.