Off-the-Record Communication, or, Why Not To Use PGP

Nikita Borisov
UC Berkeley
nikitab@cs.berkeley.edu

Ian Goldberg
Zero-Knowledge Systems
ian@cypherpunks.ca

Eric Brewer
UC Berkeley
brewer@cs.berkeley.edu

ABSTRACT

Quite often on the Internet, cryptography is used to protect private, personal communications. However, most commonly, systems such as PGP are used, which use long-lived encryption keys (subject to compromise) for confidentiality, and digital signatures (which provide strong, and in some jurisdictions, legal, proof of authorship) for authenticity.

In this paper, we argue that most social communications online should have just the opposite of the above two properties; namely, they should have *perfect forward secrecy* and *repudiability.* We present a protocol for secure online communication, called "off-the-record messaging", which has properties better-suited for casual conversation than do systems like PGP or S/MIME. We also present an implementation of off-the-record messaging as a plugin to the Linux GAIM instant messaging client. Finally, we discuss how to achieve similar privacy for high-latency communications such as email.

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*

General Terms

Security

Keywords

Perfect Forward Secrecy, Deniability, Private Communication

1. INTRODUCTION

Originally a medium for the transfer of technical information, data, and research, the Internet has grown rapidly over the last decade to become the basis for a wide variety of forms of communication, ranging from electronic commerce, to the sharing of music and video, to social conversation.

Along with the growing population of the Internet came growing concern over the security of the data flowing across it. Your online communications could be observed by any number of third parties on their way to their destinations. Even data residing on your own PC could be vulnerable if you were unlucky enough to open the wrong email attachment.

The protections developed were twofold: use firewalls and host security to lock down the endpoints, and use *cryptography* to protect the information in transit. Popular cryptographic systems, such as SSL [9], PGP [6, 33], and S/MIME [4], were developed and used to protect diverse forms of data.

The majority of electronic commerce is protected by SSL. What about social communication? Some of it takes place over email, for which PGP and S/MIME are common tools of protection. And an increasing portion of it uses instant messaging protocols, such as AIM [3], MSN [20], ICQ [16], and many others. To protect instant messages there are several alternatives. Trillian [31] was the first to have a widely deployed solution for users of its client, called SecureIM, which provided basic secrecy but no authentication. AOL followed with its own protocol [2], functionally similar to S/MIME. One can also use PGP in an instant-messaging code with the appropriate glue logic. [14] However, these approaches offer very different security and privacy properties. Which is right?

In this paper, we examine what kind of privacy is necessary for social communications. We argue that not only must encryption be used to hide the contents of the conversation, but also, the encryption must provide *perfect forward secrecy* to protect from future compromises. Additionally, authentication must be used to ensure that the person on the other end is who they claim to be. However, the authentication mechanism must offer repudiation, so that the communications remain personal and unverifiable to third parties. Only with these properties can privacy similar to real-world social communications be achieved.

However, none of the mechanisms currently used for social communications have all of these properties. PGP and S/MIME provide encryption and digital signatures, but the encryption keys are typically long-lived, and the digital signatures are non-repudiable. The Trillian SecureIM scheme provides perfect forward secrecy, but performs no authentication at all. We therefore propose a new protocol for

WPES'04, October 28, 2004
Copyright 2004 ACM 1-58113-968-3/04/0010 ...$5.00.

protecting social interactions in the context of instant messaging.

In section 2 we motivate our privacy objectives. Section 3 gives an overview of relevant cryptographic primitives, and section 4 contains an exposition of our off-the-record instant-messaging protocol. In section 5 we describe our implementation of this protocol in a common instant messaging system. In section 6, we consider how to achieve similar privacy in the high-latency context of email. Finally, we review some related work in section 7 and in section 8 we conclude.

2. MOTIVATION

When Alice and Bob are talking in person, it is easy to keep their conversation private. Alice can make sure no one is around, and, with the exception of a hidden tape recorder, she can be reasonably sure that no one else will hear the conversation. Further, the only evidence anyone can obtain of the conversation is Bob's word about what happened. Such private, off-the-record conversations are common and useful in both social and business contexts. There is even a recognized need to have similar private conversations by telephone — it is illegal to tap or record a phone conversation without the parties' consent or a court order.

What happens when Alice and Bob want to have such a private conversation online? Today, being somewhat crypto-savvy, they would use PGP. Alice encrypts her messages to Bob's public encryption key, and signs them with her own private signature key. That way, only Bob can read the messages, and Bob is assured that Alice is the one who sent them.

Unbeknownst to Alice and Bob, however, the eavesdropper Eve is listening (good thing they used crypto!) and storing all of the encrypted messages, which she can't read.

Some time later, Eve manages to obtain Bob's private key, for example though a black bag job [12], Magic Lantern [28], or a subpoena. Eve now can read *all of Bob's past email* that she's collected over the years. In addition, Eve has evidence in the form of a cryptographic digital signature that Alice was the one who sent the messages.

This doesn't sound like a private conversation at all! *After the fact*, a cryptographically verifiable transcript of Alice and Bob's conversation has been recovered.

2.1 What went wrong?

You could say that Bob losing control of his private key was the problem. But with today's easily-compromised personal computers, this is an all-too-likely occurence. We would really prefer to be able to handle such failures gracefully, and not simply give away the farm.

There were two main problems:

- The compromise of Bob's secrets allowed Eve to read not only future messages protected with that key, but past messages as well.
- When Alice wanted to prove to Bob that she was the author of the message, she used a digital signature, which also proves it to Eve, and any other third party.[1]

When we think about private messages in the context of social conversation, we really want a system with different properties: we want only Bob to be able to read the message, and Bob should be assured that Alice was the author; however, no one else should be able to do either. Further, after Alice and Bob have exchanged their message, it should be impossible for *anyone* (including Alice and Bob themselves) to subsequently read or verify the authenticity of the encrypted message, *even if* they kept a copy of it. It is clear that PGP does not provide these desirable properties.

This paper introduces a protocol for private social communication which we call "**off-the-record messaging**". The notion of an off-the-record conversation well-captures the semantics one intuitively wants from private communication: only the two parties involved are privy to the contents of the conversation; after the conversation is over, no one (not even the parties involved) can produce a transcript; and although the participants are assured of each other's identities, neither they nor anyone else can prove this information to a third party. Using this protocol, Alice and Bob can enjoy the same privacy in their online conversations that they do when they speak in person.

3. CRYPTOGRAPHIC PRIMITIVES

In this section, we outline the cryptographic primitives we will use to achieve our goal of off-the-record communication.

- **Perfect forward secrecy** will be used to ensure our past messages cannot be recovered retroactively.
- **Digital signatures** will be used so that Bob knows with whom he's communicating.
- **Message authentication codes** will be used to prove Alice's authorship of a message to Bob, while at the same time *preventing* such a proof to third parties.
- **Malleable encryption** will be used to provide for forgeability of transcripts, repudiation of contents, and plausible deniability.

3.1 Perfect forward secrecy

The most obvious feature we need from our off-the-record messaging system is confidentiality: only Alice and Bob should be able to read the messages that make up their online conversation. Since we assume everything transmitted over the Internet is public information, we need to use encryption. Now our problem is reduced to ensuring that the decryption keys for the messages never fall into hands other than Alice's and Bob's.

Alice's and Bob's abilities to safeguard their decryption keys becomes paramount. If at any later time, some decryption key is revealed, perhaps by breaking into one of their computers, or through legal or coercive means, any messages — past, present, or future — encrypted for that key would no longer be secure.

We circumvent this problem by using short-lived encryption/decryption keys that are generated as needed and discarded after use. These keys also have the property that it is impossible to rederive them from any long-term key material.

A setup such as this provides a property known as **perfect forward secrecy** [15]: once Alice and Bob both discard any given short-lived key, there is no longer any amount of information that can be collected through any means to

[1] Note that if Alice had *not* signed the message, then third parties would not have proof of Alice's authorship of the message, but then neither would Bob.

recover the key, and thus decrypt messages encrypted with that key.[2]

Not only will Eve be unable to reconstruct the key, but Alice or Bob themselves will be unable to read those past messages. This strong property ensures the confidentiality behaviour desired in off-the-record communication.

To provide perfect forward secrecy, we use the well-known Diffie-Hellman key agreement protocol [10].[3] Diffie-Hellman allows two parties communicating over a public channel to agree on a shared secret, without revealing it to an eavesdropper. Briefly, the key agreement starts with some public parameters — a prime p and a generator g of a subgroup of Z_p^* of large prime order. Alice and Bob pick two numbers (the *private keys*), x_A and x_B respectively, and they transmit g^{x_A} and g^{x_B} (the *public keys*) over a public channel. Alice can then compute the shared secret $g^{x_A x_B} = (g^{x_B})^{x_A}$; Bob can compute the same secret as $(g^{x_A})^{x_B}$. This now-shared secret is used to create the short-lived encryption key. However, it is presumed to be intractable for Eve to compute the secret, since x_A and x_B are unknown to her.

3.2 Digital signatures and non-repudiation

Digital signatures are a popular means of authenticating the author of a message; they have a number of important properties. Since digital signatures use public key cryptography, it is not necessary for every pair of communicating parties to maintain a long-term shared secret; instead, every party needs to have a single public key that is known to everyone else and used to verify their signatures. Therefore, n parties only need $O(n)$ instead of $O(n^2)$ keys, and the public keys need not be kept secret. Some popular digital signature algorithms include RSA [29] and DSS [24].

In addition, these signature keys can be *long-lived* keys, unlike the short-lived encryption keys, above. The reason is that if Bob verifies Alice's signature on a piece of data, and then the next week, Alice's signature key is compromised, it doesn't affect the fact that that old signature was valid. On the other hand, if an encryption key is used to protect a piece of data, and then the next week, the encryption key is compromised, that old data is no longer protected.

Since compromises of signature keys can't affect old data the way compromises of encryption keys can, it is acceptable to keep the same signature key around for a long time; you never protect any additional data by changing your signature key the way you do by changing your encryption key. In addition, it is *desirable* to keep your signature keys around for a long time, since that simplifies *key distribution*: making sure all of your friends have a correct copy of your signature key.

Another important consequence of digital signatures is that a digital signature may be verified by anyone, and as such can be used to prove to a third party that Alice signed a message, without Alice's cooperation.

This last property is known as *non-repudiation* — Alice is unable at a later time to disclaim authorship of a message that she signed. As we motivated in the previous section, this is not a desirable property of private communications. Alice may not want to empower Bob with the ability to prove to third parties about what she told him in private; this concern is amplified by moves of many governments to associate legal power with digital signatures. Even if Alice trusts Bob, such trust may be compromised by someone breaking into Bob's computer, or legal proceedings forcing Bob to give up past messages from Alice. The burden of non-repudiation will limit what Alice may be comfortable with saying, a restriction undesirable for simple private communication between two parties.

We want repudiability: no one should be able to prove Alice sent any particular message, whether she actually did, or not. For this reason, we never use a digital signature to prove Alice's authorship of any message. The only data we ever sign are Alice's values of g^{x_A} in the Diffie-Hellman protocol. Everyone, including Bob and Eve, can then be assured that Alice was really the one who chose the value of x_A that produced g^{x_A}, but that is all they know.

Bob, on the other hand, has extra information: x_B, and with it the shared secret $g^{x_A x_B}$. We will use this shared secret next to prove Alice's authorship of the message to Bob, and only to Bob.

3.3 MACs and repudiability

Although we want repudiability for our private, off-the-record communication, we still need authentication in order to get security; Bob needs to be assured that Alice is in fact the one sending him the messages, even if we insist that he be unable to prove that fact to anyone else.

For this purpose, we turn to message authentication codes, or MACs. A MAC is a function computed on a message using a secret "MAC key", which is shared by Alice and Bob. (A MAC can be thought of as a keyed hash function.) Alice uses her copy of the MAC key to compute a MAC of her message, and sends this MAC along with her message in a secure transmission; Bob verifies the integrity and authenticity of the message by computing the MAC on the received message using his copy of the shared MAC key, and comparing it to the MAC that was transmitted. A popular MAC construction is HMAC [17], based on a one-way hash function.

Since it is necessary to know the secret key to generate a proper MAC, if the results match, Bob knows that someone with knowledge of the shared MAC key must have sent this message. Since he presumably knows that he didn't send it himself, and only he and Alice know the MAC key, it must have been Alice who sent the message. Also, Bob knows that the message has not been modified since Alice generated it, since otherwise the MACs would not match.

However, a MAC can't provide non-repudiation: Eve can't look at the MAC'd message and determine that Alice sent it, because Eve doesn't know the MAC key. Further, Bob can't even prove to a third party that Alice sent the message; all he can prove is that someone with the MAC key generated it, but for all anyone knows, Bob could have made up the message himself!

These properties of a MAC make it perfect for off-the-record communication. Only Bob can be assured that Alice sent the message, and that the message has not been modified, yet no one (not even Bob) can prove this fact to any third party.

3.4 Malleable encryption and forgeability

In off-the-record messaging, we would like to have an even

[2] We are of course assuming that the cipher itself is strong enough so as to resist being broken without the key.

[3] For clarity, we describe only the simplest form of Diffie-Hellman key agreement here; for more detailed versions of the protocol, see [7].

stronger property than repudiability: **forgeability**. Not only do we want Bob and Eve to be unable to prove that Alice sent any given message, we want it to be very obvious that anyone at all could have modified, or even sent it. To do so, we use a **malleable encryption** scheme, which makes it easy to alter the ciphertext in such a way as to make meaningful changes in the plaintext, even when you don't know the key.

In some encryption schemes, such as certain modes of a block cipher, it is difficult to produce ciphertexts that decrypt to meaningful plaintexts without knowing the key. Even if Eve intercepts Alice's ciphertext, any changes she might make will likely result in the plaintext becoming random bits, rather than, say, English text. In general, it is poor practice to rely on this difficulty to authenticate a message, as there are truncation and other attacks which Eve might be able to use. However, such attacks may be difficult to apply in some cases, and we want to make it absolutely clear that anyone could have changed a message.

We therefore use a stream cipher. A stream cipher encrypts the plaintext by masking it with a keystream using the exclusive-OR operation; to decrypt, the same exclusive-OR is used to remove the keystream and reveal the plaintext. This encryption is malleable, as a change to any bit in the ciphertext will correspond to a change in the corresponding bit in the plaintext. In particular, if Eve can guess the plaintext of a message, she can then change the ciphertext to decrypt to *any* other message of the same length, without knowing the key. Therefore, a message encrypted with a stream cipher does not prove integrity or authenticity in any way. Of course, Alice can still use a MAC to prove to Bob that her messages are indeed hers; in the next section we will describe some extra safeguards our protocol takes to ensure that no one else can use the MAC to verify authenticity.

4. THE OFF-THE-RECORD MESSAGING PROTOCOL

In this section we shall proceed to build up a messaging protocol that achieves the desirable properties that we described in the previous sections through the use of the cryptographic primitives outlined above. We designed the protocol for low-latency communication protocols, such as instant messaging. Section 6 discusses how it might be changed to accommodate higher-latency communication, such as email.

4.1 Encryption

First, we want to ensure that a message is kept private; therefore, we must encrypt it. As discussed in the previous section, we want to use malleable encryption to provide plausible deniability. A stream cipher is best suited for this purpose. In keeping with current standards, we use AES [23] in counter mode [11]. The encryption key is chosen using a Diffie-Hellman key agreement to establish a shared secret.

To ensure that the keys are short-lived, Alice and Bob can choose to perform a new Diffie-Hellman key agreement, discarding the old key and x_A, x_B values. At this point, it will be impossible for Alice or Bob to decrypt old messages, even with help from an attacker who might remember the transmitted values of g^{x_A} and g^{x_B}, without violating the Diffie-Hellman security assumption. Thus perfect forward secrecy is achieved, as all messages encrypted with the previous key are now unreadable.

To reduce the window of vulnerability, when it is possible to decrypt old messages, Alice and Bob should re-key as frequently as possible. Fortunately, a Diffie-Hellman computation is fairly cheap — it involves only two modular exponentiations. Therefore, most computers will be able to re-key with each message; even devices with limited computational power, such as PDAs, should be able to re-key at least once a minute. To avoid extra messages during such re-keying, we combine Diffie-Hellman exchanges with normal message transmission. Each message includes a Diffie-Hellman public key (g^x) that will be used to derive the key for subsequent messages. So, a message exchange might look as follows:

$$\begin{aligned} A \rightarrow B: &\quad g^{x_1} \\ B \rightarrow A: &\quad g^{y_1} \\ A \rightarrow B: &\quad g^{x_2}, E(M_1, k_{11}) \\ B \rightarrow A: &\quad g^{y_2}, E(M_2, k_{21}) \\ A \rightarrow B: &\quad g^{x_3}, E(M_3, k_{22}) \end{aligned}$$

where $k_{ij} = H(g^{x_i y_j})$, the result of a 128-bit hash function H, such as truncated SHA-1 [22], on an element of Z_p^*, and $E(M, k)$ denotes encryption in AES counter mode using the key k.[4] Each message is encrypted using the shared secret derived from the last key received from the other party and the last key that has been previously sent to the other party. We do not use the key disclosed in one message until the following message, for reasons of authentication, discussed below. For example, in the last message above, Alice has received g^{y_2} from Bob, and the last key she has sent previously is g^{x_2}, so the key used to encrypt a message is $H(g^{x_2 y_2})$. In practice, a key ID should also be used in the message to ensure that both the sender and the receiver know which k_{ij} is being used, since the protocol does not require that Alice and Bob take turns sending messages to each other.

4.2 Forgetting Keys

To achieve perfect forward secrecy, Alice and Bob must *forget* old keys once a new key exchange is complete.[5] Ideally, after Alice sends Bob the key g^{x_n}, she would like to be able to forget x_{n-1}. However, since messaging protocols are typically asynchronous, it is possible that there is still a message in transit from Bob that was encrypted using the previous $g^{x_{n-1}}$ key; if Alice had thrown away the key, she would no longer be able to read the message. Therefore, Alice must remember the old $g^{x_{n-1}}$ key until she receives a message from Bob that uses the new g^{x_n} key. Assuming that messages are delivered in order, all subsequent messages from Bob will be encrypted using the new key.[6]

If Alice sends several messages to Bob in a row without receiving a response, announcing keys $g^{x_n} \ldots g^{x_m}$, she will need to remember the entire sequence of keys $x_{n-1} \ldots x_m$ until she receives a message from Bob, since she cannot be sure which key the next message from Bob will be encrypted under. Since using different keys does not help reduce the

[4] The bit representation of $E(M, k)$ will of course also include the initial counter value, which will be chosen to be unique for each message sent.

[5] For a secure method of forgetting keys, see [8].

[6] If out-of-order delivery is a concern, Alice can remember the $g^{x_{n-1}}$ for a short time window after receiving Bob's message to allow other possibly delayed packets to arrive.

window of vulnerability, we only generate a new key upon receiving a reply from Bob. This way, Alice need only remember at most two of her own keys at a time. Upon receiving a response that uses g^{x_n}, she can forget x_{n-1} and generate a new $g^{x_{n+1}}$ to be announced in the next message she sends.

Of course, if Bob does not reply for a long time, Alice will be able to decrypt a number of her old messages, leaving a large window of vulnerability. To address this problem, Bob should periodically send an empty message acknowledging receipt of a new key from Alice. Alice can also forget the old keys after sufficient time has elapsed that it is highly unlikely that a message from Bob using the old key is still in transit.

4.3 Authentication

As discussed in the previous section, we use a MAC for authenticating each message. To generate a MAC key, we apply a one-way hash function to the decryption key. This ensures that anyone who is able to read a message can also modify it and update the MAC. For example, even if Eve can somehow recover the encryption key (for example, due to a poor random number generator) and decrypt the messages, she will not be able to convince anyone else that it was Alice or Bob and not her who wrote the message.

The encryption key is itself the result of a hash of the Diffie-Hellman shared secret, which also needs to be authenticated in some way. We accomplish this by digitally signing the initial Diffie-Hellman exchange:

$$\begin{aligned} A \to B: &\quad Sign(g^{x_1}, k_A), K_A \\ B \to A: &\quad Sign(g^{y_1}, k_B), K_B \end{aligned}$$

Where k_a, K_A are Alice's private and public long-lived signature keys, and k_b, K_B are Bob's. If Bob already knows Alice's public key, he will be assured that g^{x_1} indeed came from Alice, and therefore the secret $g^{x_1 y_1}$ will only be known to the two of them. He can then treat messages authenticated with the key $H(g^{x_1 y_1})$ as truly coming from Alice.

Note that this is a hybrid approach to authentication, using both digital signatures and MACs. Digital signatures allow us to avoid the requirement of maintaining $O(n^2)$ pre-established shared secrets — a shared secret is established on the fly whenever communication is needed. However, the use of MACs to authenticate the actual messages allows repudiation.

We only need to use a digital signature on the initial key exchange. In further key exchanges, we use MACs to authenticate a new key using an old, known-authentic shared secret. That is, a protocol message looks like:

$$g^{x_{i+1}}, E(M_r, k_{ij}),$$
$$MAC(\{g^{x_{i+1}}, E(M_k, k_{ij})\}, H(k_{ij}))$$

So, if the initial authentication key is known to be secure, then further ones will be secure as well. Note that we cannot use $k_{i+1,j}$ to encrypt and authenticate this message, since the recipient will not be able to verify its authenticity.

4.4 Revealing MAC keys

To add an extra measure of privacy, we do something that at first seems surprising: after Alice knows all of the messages she's sent to Bob which were MAC'd with a given MAC key have been received, Alice *publishes* that MAC key as part of her next message.

Note what this has accomplished: Bob doesn't need to rely on this key any more, since he's already checked all of the messages authenticated by that key. However, now *anyone* can create arbitrary messages that have this MAC key, and no one can rule out any particular person as a potential author of the message. This can be seen as the analogue of perfect forward secrecy for authentication: anyone who recovers the MAC key in the future is unable to use it to verify the authenticity of past messages.

When can Alice be sure that Bob has successfully received all the messages signed with a certain MAC key? If she receives a message from Bob encrypted with $k_{i+1,j}$, she can be sure he received a message encrypted with $k_{ij'}$. However, Alice might have sent several messages encrypted with $k_{ij'}$, some of which may not have been yet received. Nonetheless, she can be sure that all messages authenticated with $k_{i-1,j''}$ have been received, so she may reveal all keys of the form $H(k_{i-1,j''})$ that have been used. With more careful bookkeeping, Alice may be able to reveal MAC keys sooner.

5. AN IMPLEMENTATION OF OFF-THE-RECORD MESSAGING

A natural application of the off-the-record messaging protocol is instant messaging (IM). IM is a popular way to have light-weight, informal conversations; several protocols [3, 16, 20] boast millions of users. However, these protocols do not incorporate end-to-end security, which limits their use. People are reluctant to use IM to discuss confidential business issues or sensitive personal information.

It is important that a secure instant messaging protocol achieve the "off-the-record" properties that we have described in this paper. Much of the popularity of IM is driven by the ability to have informal, social conversations [21]; a security protocol must reflect this pattern of usage and avoid properties such as non-repudiation that would destroy the informal atmosphere.

5.1 Design

We have chosen to build our off-the-record messaging protocol on top of an existing IM protocol, using it as an underlying transport. A message is first encrypted and authenticated using our protocol, and then the result is encoded as a text message and sent as a regular instant message. In this way, our solution is easy to integrate with existing protocols and clients, in the manner of a plugin, and we can avoid duplicating features of existing protocols, such as buddy lists.

Another advantage of running over an existing protocol is the potential for incremental deployment: a user can use their IM client to communicate with both people who have the secure messaging plugin and those who don't. To support these two modes, the plugin must keep a list of which buddies support secure communication and which don't. This list is populated automatically: the first time Alice sends a message to another user, Bob, it is sent unencrypted. However, we append an identifier to the end of the message to indicate that Alice supports the secure plugin. Upon receiving a message with such an identifier, the plugin initiates the Diffie-Hellman exchange and uses secure communication from then on. If, however, we receive an unencrypted message without such an identifier, we assume that the sender can only handle unencrypted messages and make a note of that in the list.

During the initial Diffie-Hellman key exchange, we notify the user that we are about to start secure communication and display the fingerprint of the other party's public key. A more cautious user will verify the fingerprint out-of-band; for others, a man-in-the-middle attack is possible at this point. We record the public key value and make sure the same key is used in future sessions. Thus, a successful impostor must be able to carry out an active attack during the first and every subsequent session; failure to do so will result in detection. This model of handling public keys is analogous to that used in SSH [32, 25], and has proven an effective way to distribute public keys in the absence of a widely-deployed public key infrastructure. Of course, if a public key can be securely obtained by other means, it can be imported into the key store and used in the future.

A potential problem is that, while the protocol we describe is session-oriented, most of the instant messaging protocols are connectionless. The off-the-record messaging protocol maintains a virtual session that lasts until the IM client is terminated, or until some period of inactivity. (The latter condition is necessary since IM clients are often left running for many days, on unattended computers.) However, it may occur that Alice terminates her end of a session while Bob's is still active (e.g. Alice logged out and then logged back in). If Bob now sends Alice a message, she will not be able to read it, since she has forgotten the encryption key.

We address this by maintaining a cache of the last outgoing message, and creating a NAK (negative acknowledgment) message. When Alice receives the unreadable message, she sends a NAK, along with the initial message of a new session. Once the session is established, Bob re-sends the cached message, which will now be readable by Alice. The message need only be cached for a short time (several seconds), to account for the expected latency of the underlying IM protocol in delivering the NAK. In pathological cases, Bob's message will be lost, but we hope that these will occur rarely enough that dropping the message will not impose a great burden on the participants; typically, Alice would simply ask Bob to send the message again.

5.2 Implementation

We implemented the off-the-record messaging protocol as a plugin for the popular Linux IM client GAIM. The implementation consists of two parts: a generic library that implements the messaging protocol, and a GAIM-specific portion that implements the plugin interface and uses the library. The library will simplify the task of creating plugins for other IM clients, and maintaining compatibility. (GAIM implements multiple protocols, so it can be used with AIM, ICQ, and many others.)

The plugin communnicates with the library using a simple API, shown in Figure 1. The function `send_message` and `receive_message` are used to process outgoing and incoming messages. Depending on the state saved in the context, the messages are either encrypted or sent in the clear. The functions return the new (encrypted/decrypted) message, its length, and a result code to indicate whether the message was encrypted, sent in the clear, should be ignored (a protocol message that does not carry user data), or if there was a protocol error. The API also includes a method (not shown) to set a UI-callback that is invoked when the library needs to communicate with the user; for example, when an unknown user's key is seen for the first time. The contexts are used to manage several simultaneous conversations with a number of different users.

We use `libgcrypt` [13] library for the cryptographic functions, using AES [23] for encryption, RSA [29] for digital signatures, and SHA1-HMAC [22, 17] for MAC authentication. Our tool uses `/dev/random` for generating random keys. More details on the current status of our implementation are available at `http://www.cypherpunks.ca/otr/`.

5.3 Measurements

We have performed a simple micro-benchmark of the protocol library to determine how much overhead it imposes on a user. Our test consisted of simulating two participants who take turns sending each other messages. On our test computer — a 450MHz Pentium III running Linux — we observed the benchmark running at about 9 round-trips per second, with varying message sizes not having significant impact. Each round-trip includes two message encryptions, two decryptions, and two key generations. Therefore, one participant could send and receive up to 18 messages per second (36 messages total). This is significantly faster than most people can type, so we believe that the off-the-record protocol will not have a noticeable performance impact. Our subjective observations while using the off-the-record plugin agree; we noticed no performance difference between secure and insecure communication.

6. EMAIL

Although a large portion of personal communication has shifted to instant messaging, much of it is still done over email; the asynchronous nature of the medium makes it suitable in contexts where instant messages are not. Privacy expectations of email conversations are lower, as it is common for the other party to keep a record of all incoming messages. However, the tools commonly used to protect email — PGP and S/MIME — still provide the wrong kind of privacy properties. When Alice sends a personal email to Bob, it is usually not her intent that Bob be able to prove to anyone else what she said; and yet her only option to prove her identity with the current tools is to use a digital signature.

The high latency of email communication makes using our "off-the-record" protocol impractical in the setting of email. Before Alice can send her first message, our protocol requires a key exchange to be completed. This means waiting for Bob to send his share of the Diffie-Hellman key, which requires him to be online. But the possibility that Bob is offline could very well be the reason Alice is using email in the first place!

However, there is a solution Alice can use in this case, called *ring signatures* [30]. A digital signature can proves that Alice sent a message. A ring signature extends this concept and can be used to prove that, given a set of people, *some* member of the set sent the message, but it is impossible to determine which one. So Alice can send her message signed with a ring signature for the set $\{\text{Alice}, \text{Bob}\}$ (and encrypted to Bob.) In this case, Bob will be able to verify that it indeed came from Alice (since he knows he did not send it himself), but will not be able to prove this to anyone else, since he can just as easily generate the ring

```
ENC_CTX new_context(unsigned char * message, int len);
unsigned char * send_message(ENC_CTX context,
  unsigned char * message, int len, int *rlen, int *result);
unsigned char * receive_message(ENC_CTX context, unsigned char *
  message, int len, int *rlen, int *result);
```

Figure 1: The generic off-the-record protocol API.

signature himself. Ring signatures have been implemented as an extension to PGP [19].

Ring signatures are similar to MACs in that they confine the authorship to a set of participants who have a certain secret key (or keys). Unlike MACs, however, they can be verified by people outside the set. If Eve were able to obtain a copy of the message, she could prove to anyone that one of Alice or Bob, and not her, had sent it. It is not clear what the implications are of such a proof, yet Alice certainly has less privacy than in the off-the-record case. With a non-interactive protocol and a risk of compromise of Bob's keys, there is no way to avoid the possibility of such a proof. At best, it can be mitigated with short-lived signature keys: Alice can publish signature keys after she is sure all messages signed with them have been received and verified by Bob. This would invalidate all signatures with those keys.

If Alice and Bob maintain a regular correspondence, they can use the off-the-record protocol for every message after the first one. Alice can include g^x with her original message. When Bob responds, he should send g^y signed with his private key, and his response encrypted and authenticated by the keys derived from g^{xy}. Alice and Bob can continue communicating this way without using digital signatures for as long as they are able to store copies of the current Diffie-Hellman keys. Due to the high latency of email, the window of vulnerability for compromising a message will certainly be longer; however, their communications will still be more private than if they had used PGP or S/MIME.

7. RELATED WORK

Perfect forward secrecy has been long recognized as a desirable feature, and several protocols use it for secure communications [5, 25, 26, 32]; some modes of TLS [9] also provide PFS. Interestingly enough, the idea of providing repudiability as a feature seems less explored. Certainly, many protocols use MACs for authentication; however, they are used for performance reasons and not to guarantee repudiation. The SKEME [18] protocol shares many design goals with our protocol; it also provides repudiable authentication and perfect forward secrecy. It avoids using digital signatures entirely, since it is concerned not only with protecting the privacy of the contents of the conversation, but also with concealing the fact that Alice and Bob ever talked. Abadi's proposed a similar protocol [1], with the extension that it is possible to hide even Alice's willingness to talk to Bob. Our protocol has focused on privacy rather than anonymity since the instant messaging context makes it difficult to conceal the identity of the participants. If anonymity is desired, one can use either SKEME or Abadi's private authentication instead of the signed Diffie-Hellman key exchange in our protocol.

The TESLA broadcast authentication protocol [27] is similar to ours in that it reveals the MAC key after a time; however, it does this for the purpose of efficient key distribution rather than to allow after-the-fact forgeries.

8. CONCLUSIONS

While the strong proofs provided by digital signatures in cryptographic packages like PGP and S/MIME are useful for signing contracts, most casual conversations online do not require, and in fact, should not have, that level of permanence associated with them.

In this paper, we have developed the "off-the-record messaging" protocol, which allows users to communicate online in a repudiable, and perfect forward secret manner, while at the same time, maintaining confidentiality and authenticity assurances.

We have implemented the protocol as a plugin for a popular Linux IM client, and we plan to extend support to other IM systems, including Windows-based ones, and possibly email systems as well. Our hope is to create many opportunities for people to have private, off-the-record conversations on the Internet.

9. ACKNOWLEDGMENTS

We would like to thank Russell O'Connor for discussions about social implications of digital signatures that motivated our work, Adam Back and David Wagner for comments on earlier versions of this paper, Len Sassaman for continued encouragement to publish our results and release the source code, and the anonymous reviewers for their many insightful suggestions.

10. REFERENCES

[1] Martín Abadi. Private authentication. In *Privacy Enhancing Technologies Workshop*, 2002.

[2] Inc. America Online. Aim personal certificates. http://enterprise.aim.com/products/aim/personalcerts.

[3] America Online, Inc. AOL Instant Messenger. http://www.aim.com/.

[4] Editor B. Ramsdell. S/MIME version 3 message specification. RFC2633, June 1999.

[5] I. Brown, A. Back, and B. Laurie. Forward secrecy extensions for OpenPGP. Internet Draft, October 2001.

[6] J. Callas, L. Donnerhacke, H. Finney, and R. Thayer. OpenPGP message format. RFC2440, November 1998.

[7] Ran Canetti and Hugo Krawczyk. Analysis of Key-Exchange Protocols and Their Use for Building Secure Channels. In *Theory and Application of Cryptographic Techniques*, pages 453–474, 2001.

[8] Giovanni Di Crescenzo, Niels Ferguson, Russell Impagliazzo, and Markus Jakobsson. How to Forget a Secret. In *STACS 99, Lecture Notes in Computer Science 1563*, pages 500–509. Springer-Verlag, 1999.

[9] T. Dierks and C. Allen. The TLS protocol version 1.0. RFC2246, January 1999.

[10] W. Diffie and M. Hellman. New Directions in Cryptography. In *IEEE Transactions on Information Theory*, pages 74–84, June 1977.

[11] M. Dworkin. Recommendation for block cipher modes of operation: Methods and techniques. NIST Special Publication 800-38A, December 2001.

[12] Electronic Privacy Information Center. United States v. Scarfo (Key-Logger Case). http://www.epic.org/crypto/scarfo.html.

[13] Free Software Foundation. `libgcrypt`. http://directory.fsf.org/security/libgcrypt.html.

[14] gaim-e project. gaim-e encryption plugin. http://gaim-e.sourceforge.net/.

[15] C.C. Günther. An identity-based key-exchange protocol. In *Advances in Cryptology — EUROCRYPT*, pages 29–37, 1989.

[16] ICQ, Inc. ICQ.com. http://www.icq.com/.

[17] H. Krawczyk, M. Bellare, and R. Canetti. HMAC: Keyed-hashing for message authentication. RFC2104, February 1997.

[18] Hugo Krawczyk. SKEME: A versatile secure key exchange mechanism for internet. In *Symposium on Network and Distributed Systems Security (NDSS)*, 1996.

[19] Lance Cottrell, Pr0duct Cypher, Hal Finney, Ian Goldberg, Ben Laurie, Colin Plumb, *or* Eric Young[7] Signing as one member of a set of keys. http://www.abditum.com/ringsig/.

[20] Microsoft Corporation. .NET Messenger Service. http://messenger.msn.com/.

[21] B. A. Nardi, S. Whittaker, and E. Bradner. Interaction and outeraction: Instant messaging in action. In *ACM 2000 Conference on Computer Supported Cooperative Work*, pages 79–88, Philadelphia, PA, 2000.

[22] National Institute of Standards and Technology. Secure hash standard (SHS). Federal Information Processing Standards Publication 180-1, April 1995.

[23] National Institute of Standards and Technology. Announcing the advanced encryption standard (AES). Federal Information Processing Standards Publication 197, November 2001.

[24] National Institute of Standards and Technology. Digital signature standard (DSS). Federal Information Processing Standards Publication 186-2, October 2001.

[25] OpenBSD Project. OpenSSH. http://openssh.com/.

[26] H. Orman. The OAKLEY key determination protocol. RFC2412, November 1998.

[27] A. Perrig, Ran Canetti, J.D.Tygar, and D. Song. Efficient authentication and signing of multicast streams over lossy channels. In *IEEE Security and Privacy Symposium*, 2000 May.

[28] Reuters. FBI confirms "Magic Lantern" exists. http://news.com.com/2102-1001-276976.html, 12 December 2001.

[29] Ronald L. Rivest, Adi Shamir, and Lenoard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of teh ACM*, 21(2):120–126, 1978.

[30] Ronald L. Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *ASIACRYPT*, pages 552–565, 2001.

[31] Cerulean Studios. Trillian. http://www.trillian.cc/products/.

[32] T. Ylonen. SSH – secure login connections over the Internet. In *6th USENIX Security Symposium*, pages 37–42, San Jose, CA, July 1996.

[33] P. Zimmermann. *The Official PGP User's Guide*. MIT Press, 1995.

[7] This implementation was published anonymously and signed by a ring signature with the keys of the authors listed.