On-The-Record: A Non-Repudiable, Authenticated, and Confidential Chat Client

Joseph C. Montminy, III and Brandon Wilson
Department of Computer Science and Electrical Engineering
University of Maryland Baltimore County
1000 Hilltop Circle, Baltimore, MD 21250
Email: [montmin1, bwilson1] @umbc.edu

***Abstract*— We propose an instant messaging protocol that provides a secure communication channel, authenticates all participants, and produces a secure and non-repudiable log of the conversation. Our inspiration comes from Borisov, Goldberg, and Brewer, who created a secure but repudiable protocol with their Off-The-Record Instant Messaging protocol. In addition, we seek to mask the distribution of chat traffic so an adversary cannot gain any knowledge from the size of chat messages. We also minimize the use of long-standing keys in our system. We analyze the system's overall time performance and its ability to mask chat traffic.**

I. Introduction

Our system provides a comprehensive, non-repudiable, authenticated instant messaging system. AIM Pro ©, the current business-grade instant messaging system from AOL ©, claims to provide encryption for communications [1]. However, we have developed a system that provides security at all points in the chat process and scales to large group conversations. Our system achieves several other goals as well:

1) Encrypts and masks the chat initiation sequence.
2) Creates ad-hoc session keys, based upon client-generated contributions, to prevent intruders from compromising long-standing keys before legitimate clients can use them.
3) Creates encrypted logs of all conversations that also contain the accompanying digital signatures.

In settings such as government offices, banks, and military bases, physical security and secrecy are major concerns. Compact personal electronic devices such as cellular phones allow malicious individuals to leak information [2]. In such locations, computer-based chat can be an acceptable way to communicate outside the secure area. Our system provides a means for people in such situations to communicate securely and to keep an encrypted, authenticated copy of their chat. Thus, only the participants in the conversation have any record of what was said, and only they can choose to reveal its contents if they deem it necessary.

We do not introduce any new cryptographic fundamentals in this system. We combine several existing cryptographic algorithms, including RSA [3], AES [4], and SHA-1 hashing [5], to produce a system that uses no long-standing keys to encrypt any conversation data traversing the network. We also hide nearly all encryption from the user, to maintain the interface of existing Instant Messaging clients.

The rest of this paper is organized as follows. In Section II, we outline related work to secure instant messaging conversations, as well as their weaknesses. In Section III, we describe the On-The-Record architecture and implementation in detail. In Section IV, we discuss vulnerabilities and limitations of our system. Finally, Section V details future work and provides some concluding remarks.

II. Background and Related Work

Instant Messaging (IM) has become an increasingly popular alternative to email as a means of fast and convenient communication over the Internet. This rise in popularity has sparked research into securing the communication between the conversation participants in a manner suitable to users' needs. Researchers and software developers differ on the characteristics of a messaging system that make it secure, depending on the setting in which the system is used.

One of the first companies to attempt to secure IM was Trillian [6]. They used symmetric encryption, but they did not provide authentication or integrity. America On-line (AOL) also added security to their IM client [7] by integrating Digital Certificates with the S/MIME protocol [8] which uses public key encryption to encrypt and digitally sign MIME objects, providing integrity, confidentiality, authentication, and non-repudiation. One weakness in AOL's protocol, however, is that the protocol uses long-lived keys. If an adversary compromises a user's private key without that user's knowledge, then the adversary can read and authenticate any message from the past or future encrypted with the key, posing a significant security concern.

One of the most recent attempts to secure IM was presented by Borisov et al. [9] in their Off-the-Record communication protocol, intended to mimic personal communication between participants in a private conversation. Borisov et al. note five characteristics of personal communication:

1) Confidentiality - The communication between individuals should remain secret.
2) Integrity - The receiver of a message should be able to verify that the message has not been altered.
3) Authentication - The receiver of a message should be able to verify that a specific author sent a specific message.

4) Repudiation - The sender can completely deny having sent any message because there is no plausible proof that the receiver can use to justify a claim.
5) Perfect Forward Secrecy - Messages sent in the past cannot be recovered in the future.

The Off-The-Record protocol posed by Borisov et al. satisfies these properties, but we believe their approach may not always be appropriate, especially in a business setting where the non-repudiability of communication is necessary. The Off-The-Record protocol is also limited because it uses the Diffie-Hellman key exchange to continuously generate secret keys during the conversation. This protocol, however, cannot produce a key shared between more than two users, rendering group conversation cumbersome. Our approach only uses the Diffie-Hellman negotiation for user authentication and to generate conversation session keys. The protocol we propose is secure, applicable to businesses, and resolves the persistent key issue present in those protocols.

III. System and Security Architecture

A. Client-Server Architecture

The physical architecture consists of multiple users or clients who connect to a central server. We make no assumptions about the physical network; the client and server only need to be able to connect to one another. The server and client also play distinct roles in supporting the chat process. The server authenticates clients and grants access to the system, in addition to notifying clients of the initial request for a conversation. Beyond these tasks, however, the server functions as a relay, receiving messages from clients and routing them to their intended recipients. Suppose Eve monitors network traffic and can see what messages enter and leave the server. By designing the server to route messages to their ultimate destination, it is harder for Eve to gain information from network traffic, since packets do not travel directly between clients. To further mask the nature of the network traffic, we add random padding to acknowledgments, as explained further in Section III-C.

The client performs most of the cryptographic processing in our architecture. When a client Alice wants to send a message M, she encrypts M with the conversation session key before sending it to the server. The server can determine which clients should receive M, but cannot read M since it lacks the session key necessary for decryption. The client who wishes to initiate a chat, hereafter referred to as the chat "owner," also handles all responsibilities related to generating the conversation session key. Finally, each client logs all messages and their corresponding digital signatures and encrypts the logs. By minimizing the the amount of information traversing the network, Eve has little opportunity to glean information from network traffic.

B. Security Implementation

When a client initially contacts the server, they share no symmetric key, requiring the use of asymmetric key cryptography. Thus, the client and server negotiate a Diffie-Hellman secret and use this as an AES key for further communication. While this does not provide the initial authenticity and integrity of RSA, our implementation is reasonable once the server authenticates the client. Whoever authenticates the client application knows the client's password; this is the same guarantee we gain from RSA, since passwords typically guard RSA private keys. To sign messages, we generate a SHA-1 digest of the message M, generating $H(M)$, and encrypt the digest with the client's private RSA key K_{priv}, generating a final signature $E_{K-priv}[H(M)]$. This provides integrity and authenticity, since only someone with access to K_{priv} can produce the signature.

We encrypt conversation messages in a manner analogous to client-server messages. In a group setting, however, establishing a symmetric key raises serious issues. As mentioned in Section 2, the Diffie-Hellman protocol is not immediately suitable for more than two participants [10]. The client who initiates the chat can arbitrarily create a key to be used by all participants, but this approach is insufficient. What assurance do clients have that the key has not existed for a long time, vulnerable to compromise?

Our solution to this problem draws from the concept of shared secrets [10]. Each client generates a random string of bytes B, hashes this string with SHA-1 to obtain $H(B)$, and sends this digest to the chat owner. The client encrypts $H(B)$ for transit with a Diffie-Hellman secret negotiated between himself and the chat owner. The chat owner then hashes these strings using SHA-1 producing $H(H(B))$, concatenates the hashes in an arbitrary order, and hashes this concatenated string, using the most significant 128 bits as the session key [11]. This can be seen in Figure 1. Alice contributes string $M1$, Bob contributes $M2$, Charlie $M3$, and Diane $M4$; H is the hashing algorithm.

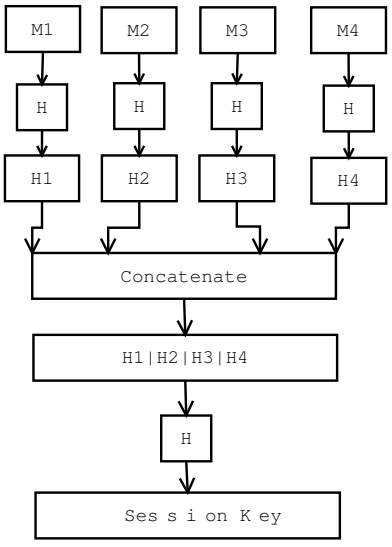


Fig. 1. Session Key Creation

The conversation owner then sends each participant a list of the individual hashes, their concatenation, and the session

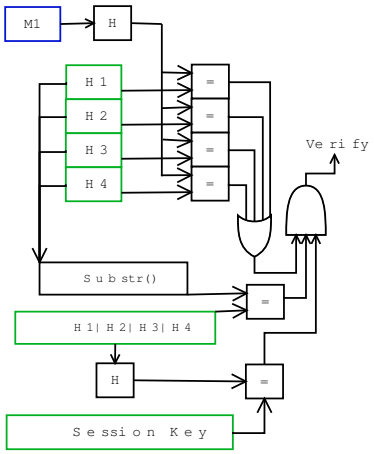


Fig. 2. Session Key Verification

key, as seen in Figure 2. To verify the key, the client then does three things:

1) Verifies that $H(H(B))$ is in the list.
2) Ensures that the hashes are all in the concatenation.
3) Verifies that the concatenation was hashed properly.

This setup allows each client to verify their own contribution to the key but not the contribution of anyone else. Since the session key contains the contribution M_i, it is almost definite that the session key could not have existed prior to the chat owner receiving M_i.

Another security issue is the distribution of message traffic. If one user reports to the others, the reporter's messages may be disproportionately larger than those of their audience. This may alert an adversary to a change of events, such as the conclusion of research or a major decision about to be made, even if the adversary does not know what was said during the conversation. While acknowledgments help balance the number of messages, they do not affect message size [11]. To reduce this risk, we add a random amount of padding to our acknowledgments; by doing this, Eve can learn less from a message based on its size. The amount of padding inserted is randomly set between the minimum and maximum size of chat messages in the conversation, so acknowledgments are not conspicuously large or small.

C. Communications Protocol

In our system, both the client and the server send many different types of messages, each encrypted in a slightly different manner. We use a uniform message protocol so all parties know exactly how to decrypt a message regardless of its type.

Since each conversation is encrypted with a different session key, the user must know exactly which key decrypts a given message. Our architecture uses message headers to identify the message type without revealing message content. Based on the header, the recipient knows what parts of the message to decrypt and what algorithm and key to use.

D. Secure Design

Our system adheres to all of the Eight Design Principles of a Trusted System [12] [13]. For space constraints, we only discuss the principles most relevant to our work here.

1) Least Privilege: Clients know only the screen names and public keys of other users; the central server architecture restricts them from information such as IP addresses. Such data could reveal a client's physical location or authentication information. In situations where the secrecy of a user's physical location is vital (such as an informant), this is especially necessary.
2) Fail-Safe Defaults: Because each client submits a random byte string for the session key, we reduce Eve's ability to guess the contributions. If client generations come from user input, Eve can guess the contribution based on her knowledge of the client. Clients may also send the same contribution repeatedly, further decreasing security. Using randomly generated contributions nearly eliminates this possibility.
3) Economy: By standardizing our protocols, we have made our system as economical as possible. The messages use a simple header/payload architecture, and the header dictates how the client processes the payload. Also, the server acts solely to authenticate clients and relay messages, decreasing its workload.
4) Separation of Privilege: Security in our system depends on several pieces of information, not just one. Neither a conversation session key nor a client's private key gain an adversary access to any future conversations.

We also protect serialized objects. In order to send messages across the network, we serialize them into arrays of bytes. However, this leaves these objects open to tampering. Furthermore, objects are encrypted and digitally signed while in serialized form, making it infeasible for an adversary to alter any piece of the message undetected.

IV. Security Vulnerabilities and Performance

A. Security Vulnerabilities

There are two major security vulnerabilities in our system. Currently, the server is a trusted key authority, and new clients learn of other clients and their public keys from the server. Clients can store this information locally, so every time they log on, they can verify the server's key records against their own. If an adversary alters the server key records before a new client L ever receives it, however, then the adversary can intercept messages from L without their knowledge, since L will be comparing the tampered key to itself. We also assume the security of the server data in our system. In an actual deployment, the server would need further security than what our implementation currently provides. If an adversary compromises the server, they can learn the passwords of other clients and assume their identities.

B. Testing and Performance

We used established cryptographic algorithms implemented in an open-source cryptographic library open to peer-review,

and therefore did not test their correctness, trusting the algorithms to be accurate. Instead, we investigate the overall performance speed, as Borisov et al. did in the Off-the-Record system [9]. We also statistically analyze examine the message sizes over the course of the conversation to determine if the padding sizes change as the message sizes change.

We used several outside resources to perform our tests. In both the speed and network traffic tests, we wanted to control the conversation content. Thus, we consulted the Gutenberg Project [14] and used excerpts from *Macbeth* [15], *As You Like It* [16], and *Hamlet* [17] to simulate a variety of conversational settings. We used an excerpt from a 1961 White House meeting on the Cuban Missile Crisis [18], which features ten users and a wide variety of message lengths.

For the Diffie-Hellman key negotiation, both parties need to agree on a prime modulus P and a generator G. Since the protocol's security lies in the intractability of the discrete logarithm problem, we used constant 100-digit values for these parameters. We obtained these primes from WIMS [19], a French mathematics-related Web server that provides resources to further verify number primality. While WIMS states the prime generator should not be used for real cryptographic systems, we felt it was sufficient for our simulation and would move to a more secure prime generator in a public release.

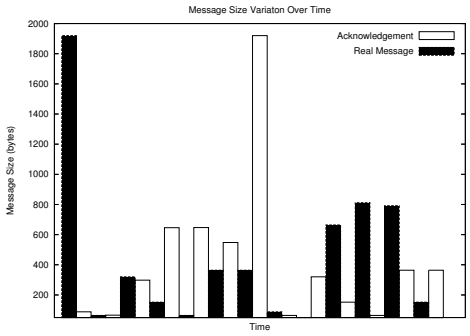
We tested the login time of our system by authenticating a user with "buddy" lists of size 0, 1, 5, and 10, running ten trials for each size. Our results showed no significant difference in login time based on the size of this list; the average login time for an empty buddy list was 572.2 ms, while the login time for a buddy list of ten users was 584.2 ms. Using the logic of Borisov et al. [9], we conclude that these times are acceptable.

TABLE I
MESSAGE SIZE VS. TRANSMISSION TIME.

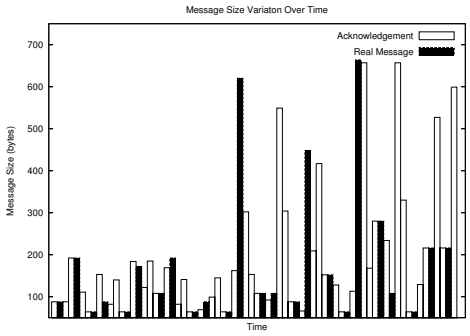
Message Size	Average Transmission Time (ms)
100	58.3
1000	70.5
10000	130.2

We also tested the transmission time of messages of varying length, as seen in Table I, running ten trials of each message size. The times in Table I include encryption and decryption of the messages. While the transmission time does increase significantly between the two largest sample sizes, the increase in message size far exceeds the increase in transmission time. Thus, we find these results to be acceptable, and believe we would find similarly acceptable transmission times for even larger messages.

Figure 3(a) shows the size of conversation messages versus acknowledgment messages for a balanced two-way conversation. In this particular conversation, a large message occurs early in the chat. While this largely does not impact the size of the acknowledgment messages for the rest of the conversation, one conspicuously large acknowledgment does appear about halfway through the conversation. We believe this suggests the need to refine the padding function to dampen the effect of such messages, discussed further in Section V.



(a) Two-Way Conversation



(b) Three-Way Conversation

Fig. 3. Message size vs. Acknowledge size for different conversations.

Figure 3(b) demonstrates a longer conversation involving more clients. Here, the padded acknowledgments blend quite well into the legitimate chat traffic. As the size of the chat messages grow, the size of the acknowledgments increase as well. While there are some outlier acknowledgments toward the end of the chat, they are not as conspicuous here as they are in the two-way chat.

To truly stress-test our system, we used a 1961 transcript of a White House meeting [18]. There are ten participants in this conversation, with some delivering lengthy analysis to the president, with others simply giving their assent or dissent to strategies others propose. For this test, we used a bank of ten computer terminals, with one also functioning as the chat server, a situation we deemed acceptable since we were not interested in time-related data in this experiment. We hoped to see that the size of acknowledgment messages would mirror the sizes of the legitimate ones, growing and shrinking as appropriate.

This trial highlighted the weaknesses of our current acknowledgment-padding function. A graph of message sizes for this trial is too detailed to present here. As in the two-way conversation, there are several very large messages early in the conversation. However, these messages create a large window for the amount of acknowledgment padding, and thus

the acknowledgment messages tend to be conspicuously larger than the legitimate ones, making them easier to detect.

V. Future Work and Conclusion

We currently envision two areas for future work. First, we want to improve our algorithm to for padding the acknowledgment messages. While we initially believed that selecting a random size between the smallest and largest messages seen in the conversation would suffice, the Kennedy transcript showed that this was not the case. When a large message occurs very early, as in the Kennedy transcript, all subsequent acknowledgment packets can be extremely large compared to actual chat messages, since the early, large message creates a large range from which to randomly select an acknowledgment packet size. We believe more sophisticated network traffic models, as well as only considering the previous N messages, may provide guidance in improving the function.

We also see the potential for future work in adapting our protocol to handling larger data objects, such as pictures, movies, documents, and other media. Rarely do meetings occur where no objects need to be disseminated to the group; however, transmitting these objects discreetly may be daunting. We also plan to re-examine the centrality of our architecture; while it has benefits, it also has security risks, such as the ability for an adversary to compromise the entire system by compromising the server.

We have presented a chat system that not only secures messages and authenticates users, but also produces logs that cannot be repudiated by any of the participants in the conversation. In addition, our system dramatically reduces the use of long-standing keys in a chat environment, and tries to mask the nature of the chat traffic so the size of messages does not inadvertently leak information to an adversary.

Acknowledgments

We would like to acknowledge and thank Richard "Rick" Carback for his insights on establishing the session key and masking network traffic. We also thank Adam Anthony, John Kloetzli, and Blaz Bulka for their insights on developing this system, as well as all readers for their comments. Finally, we thank Dr. Krishna Sivalingam, whose course inspired this work, and Dr. Marie desJardins for her advice and insights on this work.

References

[1] S. M. Kerner, *New AIM Pro Drops the Toys, Supports Outlook*. http://www.instantmessagingplanet.com/enterprise/print.php/3621086: internetnews.com, July 2006.

[2] R. Coloma, *Camera phones boost telco industry but raise security, privacy concerns*. http://www.physorg.com/news4601.html: Physorg.com, 2005 June.

[3] R. L. Rivest, A. Shamir, and L. M. Adelman, "A METHOD FOR OBTAINING DIGITAL SIGNATURES AND PUBLIC-KEY CRYPTOSYSTEMS," p. 15, 1977. [Online]. Available: citeseer.ist.psu.edu/rivest78method.html

[4] J. Daemen and V. Rijmen, "Aes proposal: Rijndael." [Online]. Available: citeseer.ist.psu.edu/daemen98aes.html

[5] D. Eastlake and P. Jones, "Rfc 3174 - us secure hash algorithm 1 (sha-1)," Network Working Group, Tech. Rep., September 2001.

[6] C. Studios, "Trillian," http://www.trillian.cc/products.

[7] J. Lin, "Aim encryption certificates," http://www.ocf.berkeley.edu/~jjlin/aim-certs.html.

[8] P. Z. D. Atkins, W. Stallings, "Rfc 1991 - pgp message exchange formats," http://www.faqs.org/rfcs/rfc1991.html, August 1996.

[9] N. Borisov, I. Goldberg, and E. Brewer, "Communication privacy: Off-the-record communicaton, or, why not to use pgp," in *Proceedings of the 2004 ACM Workshop on Privacy in the electronic society*. ACM Press, New York, New York, USA, October 2004, pp. 77–84.

[10] L. C. W. Wade Trappe, *Introduction to Cryptography with Coding Theory*. Upper Saddle River, NJ 07458: Prentice Hall, 2002.

[11] R. Carback, "Personal interview," November 2006.

[12] J. Saltzer, "Protection and the control of information sharing in multics," in *Communications of the ACM*, vol. 17, Num. 7, July 1974, pp. 388–402.

[13] J. Saltzer and M. Schroeder, "The protection of information in computing systems," in *Proceedings of the IEEE*, vol. 63, Num. 9, September 1975, pp. 1278–1308.

[14] M. Hart, *Project Gutenberg*. http://www.gutenberg.org/wiki/Main_Page: Gutenberg Project, December 2006.

[15] W. Shakespeare, *Macbeth*. http://www.gutenberg.org/dirs/etext98/2ws3410.txt: Collins, 1998.

[16] ——, *As You Like It*. http://www.gutenberg.org/dirs/etext99/1ws3011.txt: The Complete Works Of William Shakespeare, 1999.

[17] ——, *Hamlet*. http://www.gutenberg.org/dirs/etext99/1ws3011.txt: The Complete Works Of William Shakespeare, 1999.

[18] T. A. P. of Yale Law School, *The Cuban Missile Crisis Transcript of a Meeting at the White House*. http://www.yale.edu/lawweb/avalon/diplomacy/forrel/cuba/cuba018.htm: Yale Law School, 2007.

[19] G. Xiao, *WIMS - WWW Interactive Mathematics Server*. http://wims.unice.fr/wims/en_home.html: Université de Nice, November 2006.