

# Really Secure Chat, Really!

Kenny C.K. Fong<sup>1</sup>, Jonathan M. Hunt<sup>2</sup>, Soyini D. Liburd<sup>3</sup>, and Eduardo I. McLean<sup>4</sup>

December 10, 2002

## Abstract

*This paper provides the architectural design of a secure authenticated instant messaging protocol. Such a protocol should provide the user with the ability to effectively sign and encrypt messages, easily establish communication session keys, provide adequate levels of security, and be fast enough to still provide real-time "instant" communication. Our design achieves these goals by automating the signing and authentication aspects of our scheme, using a strong but simple group key exchange protocol, and by using OAEP/AES message encryption to provide more than adequate message security in real-time. These features all serve in allowing users to easily establish and engage in secure communication sessions.*

## 1 Introduction

Instant messaging has over the last few years become an intensely popular means of communication. Various businesses depend on instant messages as a means of improving productivity or making employees aware of company sensitive data, and friends and families use them as an inexpensive means of staying in touch with one another. Due to increased dependence on instant messaging for communication, the sensitivity and amount of information that frequents instant messaging channels have risen over the years, however the level of security with which messages are handled and delivered has not. None of the five most popular instant messaging protocols (AIM, MSN Messenger, Yahoo Messenger, ICQ, IRC) provide the option for secure communication sessions and none of the existing secure messaging programs provide a simple means of engaging in secure communications that does not rely on a trusted third party [8,16,18].

Client	TTP	Key Exchange	Forward Security	Authentication
Trillian	YES	DH	YES	TTP
CypherGuard	Option	Proprietary	?	TTP
SIRC	YES	NO	?	TTP

Table 1: Security of Secure Messaging Programs in the Market

Goals in designing a secure chat system should adhere to the following:

1. The system should provide authentication and confidentiality without relying on a "trusted" third party. It should be secure against adaptive chosen ciphertext attacks and message replay.
2. The system can be layered on top of an existing chat and key distribution infrastructures, for example, AIM and Yahoo Messenger.
3. The system should be easy to use, at least as easy as PGP to setup keys. It should provide group key secrecy.
4. The security provided by the system should not add a significant delay to sending and receiving messages. The system should be fast and efficient enough to be applicable to large groups or clients with access to only a small amount of computing power.

<sup>1</sup> Division of Engineering and Applied Sciences, Harvard University, USA, kcfong@fas.harvard.edu

<sup>2</sup> Information Services and Technology, MIT, USA, jmhunt@mit.edu

<sup>3</sup> Department of Electrical Engineering and Computer Science, MIT, USA, soyini@mit.edu

<sup>4</sup> Department of Electrical Engineering and Computer Science, MIT, USA, eddie@mit.edu

5. The encryption scheme and strength should be user selectable, e.g., RSA or AES key size can be chosen differently to meet different needs.
6. Open design should be adopted, so that the system can be publicly reviewed for security strength similar to the competition that resulted in AES.

In this paper we provide a design for an easy-to-use secure authenticated chat system that adheres to the aforementioned goals. In Section 2, we provide designs for the rotational key exchange protocol used in our system and compare our protocol with other similar key exchange protocols. In Section 3, we describe the encryption and signature schemes used in our system and analyze its overall performance. In Section 4, we give an analysis of our system's security. We conclude this paper in Section 5.

## 2 Session Key Establishment

### 2.1 Session Key Management

We use a symmetric session key in our OAEP/AES messaging protocol because it allows faster message encryption and decryption, and thus faster chat, than a public key system. We propose that clients be given a choice between the higher security, less efficient existing key agreement protocols [2,12,13,14], and the dynamic key distribution protocol, discussed here, that sacrifices forward secrecy but is much faster and more efficient than existing protocols.

Current tree-based group Diffie-Hellman (TGDH) schemes provide key independence within each chat so that persons cannot decrypt chat messages sent before they enter the chat or after they leave [12,13]. Many of these schemes also allow each member of the chat to contribute some secret towards the shared session key without ever sending the session key over the network, encrypted or otherwise [12,13]. Our key distribution protocol sacrifices these potentially advantageous attributes of tree-based schemes for considerable gains in performance. TGDH protocols have achieved performance levels of  $O(\log n)$  exponentiations and  $O(1)$  messages sent for entry into and exit from dynamic chat groups containing  $n$  persons. This is efficient compared to older schemes like Burmester-Desmedt [7] and basic group Diffie-Hellman, but still restricts the number of users that can participate in the chat to less than 100 [2] and is still unsuitable for chat groups with a high frequency of entrance and exit. Our key exchange protocol yields performance levels of  $O(1)$  exponentiations and messages sent for entry into and exit from dynamic chat groups. This allows for very large groups, of more than 10,000 persons, and makes the protocol much faster than current tree-based schemes. Our protocol is thus better than current tree-based schemes for large groups who have moderate security needs (where forward secrecy is not required) but require very fast communication or have low computational capabilities. We use a form of decentralized key distribution. Only the chat initializer (user 1) contributes a secret to the initial session key,  $k_1$ , but once that initial key has been decided any chat member may request that the key be updated. This could include choosing a new session key and distributing it if that chat member suspects, for example, that the previous key has been compromised and all the members of the chat agree. Persons are unable to decrypt messages that have been encrypted with an initial key  $k_1$  that is unknown to them. We use a decentralized scheme to prevent a single-point-of-failure that may be targeted by attackers and we use a key distribution scheme to avoid the numerous rounds needed to incorporate each member's secret contribution to the group key in traditional key agreement schemes. We use key rotation [10] to ensure backward security by allowing each chat member to independently generate the new shared secret key  $k_{new}$  every time a new person enters the chat. The new key  $k_{new}$  is equal to  $h(k_{old})$ , where  $h$  is a one-way, collision-resistant hash function that is known to the chat members. Though this scheme does not provide forward secrecy, we provide a list of identities (public keys or certificates) of all the persons who have ever been in the chat so that persons wishing to join the chat are aware of all the people who are able to decrypt their messages. This allows a potential chat member to refuse to join the chat, or start a new chat of his own, if he does not trust someone who previously entered the chat.

### 2.2 Key Establishment Protocol

Our key establishment scheme uses the Diffie-Hellman key exchange protocol to distribute the shared session key to new members, and uses a one-way hash function,  $h$ , to update session keys on entry.

**Entry:** The person who initializes the chat, user 1, picks a key  $k_1$  that is suitable for AES.

1. When a user  $n+1$  wants to join the chat, he sends a request to any user  $i$  currently in the chat. This request contains his public key  $PK_{n+1}$  and  $g^{u_{n+1}} \pmod p$ .
2. If user  $i$  wants to include user  $n+1$  in the chat, user  $i$  sends him  $g^{u_i} \pmod p$  – where  $u_i$  is user  $i$ 's secret, an *Identity List* containing the identities (public keys or certificates) of all persons who have ever been admitted to the chat, the *session id* of the chat and a random number  $r$  as a challenge. User  $i$  is termed user  $n+1$ 's sponsor.
3. If user  $n+1$  is satisfied with the identities in the chat, he responds to the challenge by encrypting and signing  $g^{u_{n+1}}$ ,  $g^{u_i}$ , the session id and  $r$ .
4. If user  $i$  is satisfied with user  $n+1$ 's proof of identity, user  $i$  broadcasts to the current chat room members that user  $n+1$ , with public key  $PK_{n+1}$ , is about to join the chat. The other users add this public key to their (identical) copy of the Identity List and compute the new session key  $k_{new} = h(k_{old})$ . At this point  $k_{old}$  is no longer used as the session key. User  $i$  then sends user  $n+1$  the new session key  $k_{new}$ , encrypted with  $g^{u_{n+1}} \pmod p$ .

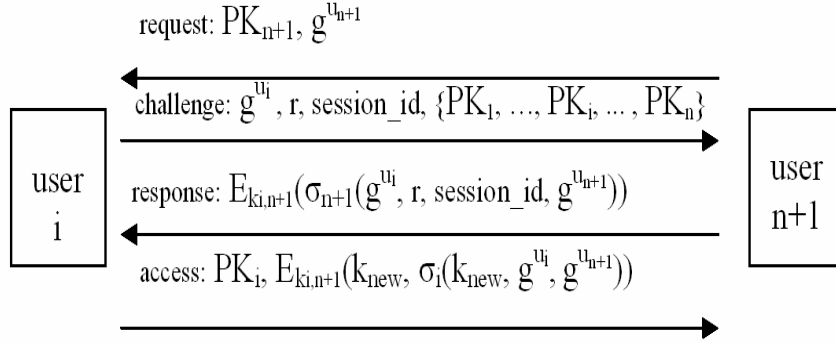


Figure 1:  $k_{i,n+1} = g^{u_{n+1}} \pmod p$ .  $k_{new}$  is the session key.  $g^{exp}$  values are all mod  $p$ .  $\sigma_i$  represents a signature created with the private key corresponding to  $PK_i$ .

**Exit:** No exit protocol is needed. Users continue to send messages with the awareness that anyone who has the shared session key may read them, whether or not that person appears to be present in the chat.

**Notes:**

1. *Trust and integrity.* We assume that user  $i$  and user  $n+1$  trust each other since their exchange is authenticated, and as such we can believe that user  $n+1$  trusts that the integrity of the Identity List has not been deliberately compromised by user  $i$ .
2. *Group authentication.* Before issuing the challenge, users could discuss with the chat room whether or not they should allow the person with private key corresponding to  $PK_{n+1}$  to enter the chat.
3. *Key confirmation.* The shared session key is sent over the network, but it is safely encrypted using an authenticated Diffie-Hellman scheme. The fact that the sponsor signs a message containing the key makes it easy for the new chat member to recognize quickly if he receives a corrupted key and he can use their pair wise secret to request that the key be resent.

### 2.3 Analysis

Our session key establishment protocol is very efficient in that it incurs very low computation and communication costs. Each chat member is able to compute new session keys independently and consequently our protocol requires no key distribution messages to existing chat members, except the announcement that a chat member is about to be added. Thus entry can be achieved with  $O(1)$  communications – one broadcast and four messages between the potential chat member and his sponsor. Further, existing chat members, with the exception of the sponsor, have to do no exponentiations at all when a new member enters. The potential chat member and his sponsor have to do  $O(1)$  exponentiations, two each, to set up a pair wise shared secret between them that can be used to encrypt the session key. As mentioned before, exit or disconnection incurs no cost at all to remaining chat members. Because of these extremely low communication and computation costs, very large group sizes and very high frequencies of entry and exit can be tolerated.

Our session key establishment protocol is very robust. The protocol does not rely on long-term pair wise secrets that must be stored and protected and could be compromised. Further, chat can continue seamlessly among the remaining chat members, regardless of which chat members exit or are disconnected due to network difficulties. Because there is no exit protocol, a disconnected chat member can seamlessly join the chat again without announcement and without having to request the current session key. If new members have entered since his disconnection, the reconnected member can repeatedly apply the hash function to his old key until he has the current session key of the chat.

### 3 Performance

#### 3.1 Key Exchange

One of the big drawbacks of shared computation keys is the performance, particularly over wide area networks (WAN). [2] shows the comparison of 5 group key agreement protocols and the experimental performance results in both a local area network (LAN) and WAN. We are comparing our key exchange protocol against two of the protocols tested by Amir et al.: group Diffie-Hellman (GDH) and tree group Diffie-Hellman (TGDH). In the LAN setting, TGDH was the best performer and among the top two in the WAN setting. Our comparisons show that our scheme is better than TGDH in both a LAN and WAN environment, especially with large memberships.

		Communication				Computation		
		Rounds	Messages	Unicast	Multicast	Exponentiation	Signatures	Verifications
GDH	Join	4	$n+3$	$n+1$	2	$n+3$	4	$n+3$
	Leave	1	1	0	1	$n-1$	1	1
TGDH	Join	2	3	0	3	$3/2 \log(n)$	2	3
	Leave	1	1	0	1	$3/2 \log(n)$	1	1
Ours	Join	3	5	4	1	2	2	2
	Leave	0	0	0	0	0	0	0

Table 2: Communications and Computation Costs<sup>5</sup>

Because our scheme does not rely on shared computation of the key, we do not need to do any additional computation or send any messages when someone leaves a session. The key remains the same after the departure. This does leave open the possibility that the member who left can still decrypt messages sent to the group if he still has access to the network traffic of the session. That member can also generate all the future keys for the session by computing the hash of the current key whenever a new person logs in. To mitigate this risk, each participant maintains a list of all previous participants of that session, which is provided to new members of the chat as they join. When GDH and TGDH got to 50 users, the delay for an exit were respectively about 110 msec and 60 msec for the LAN and 800 msec and 800 msec for the WAN with a 512-bit key.

Our performance improvements are expected to be even better for the join operation. Since our scheme does not depend on the size of the chat, we should see only a modest increase in joining times due to the increase in costs to do a multicast message to a larger group of people. As we are sending half the multicasts for a join for GDH and a third of the ones used by TGDH, our growth due to this factor will be less. We expect that the graph of our join time on the LAN will be a flat line well below 40 msec and between 600-800 msec if we used the same WAN as Amir et al. Our performance should continue in this way well into the thousands of members.

As the group grows into the thousands or if a particular session has been around for a long time with lots of join and leave operations, the list of all chat members will become so large as to dominate the key exchange performance just in encrypting and transferring that list. To remedy the situation, a new key would need to be generated and distributed to all the current chat members as if the session was just starting again. Basically, when the performance become bad enough, one of the users could start a new chat session next door and mention it on the old session, since message passing will still perform as usual.

<sup>5</sup> Table format, including data for GDH and TGDH from [2]

## 3.2 Encryption and Signature of Messages

We also have sought to provide sufficiently fast message encryption while adding security against chosen ciphertext attacks by using the Optimal Asymmetric Encryption Padding (OAEP) with Advanced Encryption Standard (AES). In [4] Bellare and Rogaway show how to use OAEP with RSA to provide a proof of plaintext awareness and a non-deterministic output as we desire. Once we have the key exchanged, we must complete the several steps to encrypt and similar steps to decrypt each message we send.

Encryption involves 1 PRNG (or hash function), 2 hash functions, 1 signature, and the AES encryption. Decryption involves basically the same computations for AES decryption, 1 verification, and 2 hash functions to undo the OAEP.

		Cost	# needed for encrypt	cost for encrypt	# needed for decrypt	cost for decrypt
hash functions <sup>6</sup>	Md4	1.5 $\mu$ s	3	4.5 $\mu$ s	2	3.0 $\mu$ s
	Md5	1.8 $\mu$ s	3	5.4 $\mu$ s	2	3.6 $\mu$ s
	Sha1	3.1 $\mu$ s	3	9.3 $\mu$ s	2	6.2 $\mu$ s
Signatures	RSA 1024-bit	6.5ms	1	6.5ms	0	0.0ms
	RSA 2048-bit	40ms	1	40ms	0	0.0ms
	DSA 1024-bit	3.5ms	1	3.5ms	0	0.0ms
	DSA 2048-bit	12ms	1	12ms	0	0.0ms
Verifications	RSA 1024-bit	0.3ms	0	0.0ms	1	0.3ms
	RSA 2048-bit	1.2ms	0	0.0ms	1	1.2ms
	DSA 1024-bit	4.3ms	0	0.0ms	1	4.3ms
	DSA 2048-bit	14ms	0	0.0ms	1	14ms
AES <sup>7</sup>	128-bit	1.6ms	1	1.6ms	1	1.6ms
	192-bit	1.7ms	1	1.7ms	1	1.7ms
	256-bit	1.8ms	1	1.8ms	1	1.8ms

Table 3: Encryption and Decryption Costs

If we take the worst case in all cases, then our total time to encrypt and decrypt a 1,000 byte message will be  $9.3\mu\text{s} + 6.2\mu\text{s} + 40\text{ms} + 14\text{ms} + 1.8\text{ms} + 1.8\text{ms} = 57.6\text{ms}$ . In a typical WAN setting, the time to encrypt and decrypt the message is going to be short compared to the delay in the network. Even if we experience a 300msec delay for the WAN link, messages can still be sent and received in under a second, which should be acceptable performance for any human user.

## 4 Security

### 4.1 Session Key Establishment

Before comparing the security of the tree-based group Diffie-Hellman key exchange protocol (TGDH) with that of the more efficient group key distribution scheme we propose, let us list the common security goals of dynamic group communication as follows:

- Group key secrecy: it is computationally infeasible for a passive adversary to discover any group session key.
- Backward secrecy: a group session key cannot be used to discover any previous group keys. Consequently, a new chat member cannot read any past messages.
- Forward secrecy: a group session key cannot be used to discover any subsequent group keys. Consequently, a chat member exiting the chat session is unable to read future messages in the chat room.

<sup>6</sup> Hash function values are using the 64 byte results found in the Appendix using a 1,000 byte message.

<sup>7</sup> The AES cost is computed as the time it takes to encrypt/decrypt a 1,000 byte block based on the Java implementations shown in [3].

- Key independence: a chat member cannot read any messages sent before he entered the chat room, and cannot read any messages that will be sent after he leaves the chat room. This is a combination of forward secrecy and backward secrecy.
- Authenticated key exchange (AKE)
  - Implicit authentication: only the intended partners can compute the session key.
  - Semantic security: a fresh session key is indistinguishable from a random string.
- Mutual authentication (MA): each chat member is convinced of the identity of his partners.

Notice that if any chat member colludes with the adversary, then both TGDH and our scheme are totally broken, because the chat member may reveal the session key to the adversary. In this situation, we may as well regard the chat member as an adversary himself. Therefore, theoretically speaking, we should prove that either all the chat members in the chatroom are honest, or all are adversaries. Practically speaking, this means when an entity A requests to enter the chat room, his identity needs to be authenticated by those chat members already in the chatroom, to make sure that he is not an adversary. Simultaneously, A needs to authenticate the identities of those chat members already in the chatroom before entering, to make sure that there is no adversary in the chatroom. This is what it means by mutual authentication, which is a stronger kind of authentication than implicit authentication. Note that key independence and mutual authentication are two different concepts.

Steiner et al. [17] proved in 1996 that the basic GDH protocol guarantees group key secrecy and key independence. The proof is based on the intractability of the Group Decision Diffie-Hellman problem. In 1998, McGrew [14] proposed TGDH and proved that it is at least as secure as GDH. However, no formal proofs that GDH and TGDH achieve AKE or MA have ever been presented. Bresson et al. [6] proposed the Authenticated GDH (AGDH) protocol in 2001 that is shown to achieve AKE.

The session key distribution scheme we propose clearly satisfies group key secrecy as a consequence of the intractability of the discrete logarithm problem. Any passive adversary cannot obtain the user secret  $u_i$  from  $g^{u_i}$  and similarly  $u_{n+1}$  from  $g^{u_{n+1}}$ . More importantly, when someone joins the chat room, the old session key is never transported, and the new session key is encrypted before it is sent to the new chat member. The one-wayness of the hash function  $h$  used guarantees backward secrecy in our scheme, but unfortunately it does not guarantee forward secrecy. Any adversary who obtains the current session key  $k$  can easily compute the next session key  $h(k)$ . Hence, our scheme does not achieve key independence.

Mutual authentication can be used to minimize any risk incurred due to lack of forward secrecy. When user  $n+1$  makes a request to user  $i$  of joining the chat room, user  $i$  provides a list of identities (public keys or certificates) of all the persons who have already been in the chat so that user  $n+1$  is aware of all the people who are able to decrypt his messages. This allows a potential chat member to refuse to join the chat, or start a new chat of his own, if he does not trust someone who previously entered the chat. Similarly, user  $n+1$  provides user  $i$  with his public key, which can be used to authenticate the identity of user  $n+1$  by user  $i$ . Later in the response message in the interactive protocol, user  $n+1$ 's signature on  $(g^{u_i}, r, session\_id, g^{u_{n+1}})$  using his private key can ensure that he does own the public key. However, we need to point out that although this proposal attempts to model mutual authentication, it in fact relies on trust over the public keys (i.e. PKI) and no formal proof of MA or implicit authentication is given here. More importantly, our scheme assumes that user  $n+1$  trusts user  $i$  and consequently the set of public keys sent to it by user  $i$ .

Finally, assuming that the hash function used is random, our scheme does satisfy semantic security, which means any group session key looks like a random bit string. Furthermore, our scheme does not use any TTP for key distribution.

Our scheme provides impressive performance while sacrificing a tolerable amount of security.

## 4.2 Encryption of Messages

A chat message block  $M = (\text{timestamp}, \text{recipient}, \text{sender}, \text{message text})$  is encrypted as follows. First, OAEP is applied on  $M$  to yield  $M'$ . Next,  $M'$  is encrypted with AES in CBC mode. Confidentiality of the message is achieved by the use of AES in CBC mode, one of the more secure modes of operation for block ciphers. 128-bit

AES has approximately  $3.4 \times 10^{38}$  possible keys, which means there are on the order of  $10^{21}$  times more AES 128-bit keys than DES 56-bit keys. As mentioned in [1], assuming that one could build a machine that could recover a DES key in a second (i.e., try  $2^{55}$  keys per second), then it would take that machine approximately 149 thousand-billion (149 trillion) years to crack a 128-bit AES key. To put that into perspective, the universe is believed to be less than 20 billion years old. It is true that AES is a young cipher and so there might exist attacks against AES that are faster than key exhaustion but have not been discovered yet. However, AES has already been approved by NIST, which continues to follow developments in the cryptanalysis of AES. The provision of other ciphers besides AES in our scheme allows users to choose other ciphers if they do not have confidence in AES. Finally, barring any attacks against AES that are faster than key exhaustion, then even with future advances in technology, it is speculated that AES has the potential to remain secure well beyond twenty years.

The application of OAEP on the chat message introduces non-determinism by padding the message with a random number. OAEP was introduced by Bellare and Rogaway [4] in 1995. In the paper, they proved that  $f$ -OAEP is semantically secure and plaintext-aware ( $f$ -OAEP refers to the scheme of applying OAEP on the message first before applying the encryption function  $f$ ); moreover, they proved that plaintext-awareness of  $f$ -OAEP implies security against adaptive chosen-ciphertext attacks (ACCA). Note that security against ACCA is equivalent to non-malleability due to Dwork et al. [9]. However, it turned out that the definition of plaintext-awareness presented in this paper is too weak to imply security against ACCA. In 1998, Bellare et al. [5] refined the definition of plaintext-awareness and showed that plaintext awareness plus security against chosen plaintext attack implies security against ACCA. However, this paper does not address the question of whether  $f$ -OAEP is plaintext-aware under the new definition of plaintext-awareness. Later, Shoup [15] even proved that a proof that  $f$ -OAEP is plaintext-aware (under the new definition of plaintext-awareness) is unlikely to be possible for all  $f$ . However, in 2000, Fujisaki et al. [11] proved that RSA-OAEP is plaintext-aware (and consequently secure against ACCA), assuming that RSA is a one-way function. It is now widely believed that OAEP does guarantee plaintext-awareness and so guard against ACCA for many trapdoor permutation  $f$ , one candidate being AES. Finally, we need to point out that one must be very careful in implementing OAEP, since poor implementations can give rise to attacks [11].

Non-repudiation is achieved by including the timestamp in the chat message block, which also protects the recipient and sender against replay attacks. Authentication is achieved by signing the concatenation of the header of the encrypted message block with RSA signature scheme. Note that no TTP is involved in the encryption and signature scheme here. However, we do need to point out that the signature scheme relies on trust over the public verification key (i.e. PKI).

### 4.3 Non-Cryptographic Vulnerabilities

The following lists the most common non-cryptographic vulnerabilities against the security of our chat system:

- Software flaws, such as buffer overflows or insecure configurations, may be present in the client chat software and may provide a means for remote users to initiate attacks that execute code on internal systems. Our chat system proposal requires that the source code be publicly reviewed, increasing the chance of any software flaws being discovered.
- Social engineering attacks may entice users into taking insecure actions, such as communicating sensitive information with outsiders. A well-written security policy should accompany the chat software, warning users of the potential for social engineering attacks and of taking caution in releasing information.
- Untrusted networks (both domestic and international) allow interception from adversaries. Strong encryption of messages with OAEP/AES in our proposal achieves confidentiality of messages sent through untrusted channels.
- Attacks involving Trojan horse and virus programs have been known to leverage chat networks to enable intruders to coordinate the actions of compromised computers in attacks against other Internet sites. This is a security issue of the underlying messaging infrastructure (e.g. AOL Instant Messenger, Yahoo Messenger), not of our messaging protocol.
- Denial-of-service attacks could be mounted by issuing an exploding number of requests of joining the chatroom to the same entity already in the chatroom.

## 5 Conclusion

The scheme described herein provides users with a secure means of instant messaging communication in real time. Using our group Diffie-Hellman key exchange protocol and OAEP/AES block cipher encryption as well as an automated means of encrypting, authenticating, and signing messages users can now engage in communications with a strong sense of backward security and protection against ACCA. Though forward security is not accounted for the scheme described herein ensures that communication sessions only occur amongst trusted parties. Our scheme solves the problem of providing users with strong security without terribly complicating the method by which communication occurs.

## References

- [1] Advanced Encryption Standard (AES) fact sheet. Available at <http://csrc.nist.gov/encryption/aes/aesfact.html>.
- [2] Y. Amir, Y. Kim, C. Nita-Rotaru, and G. Tsudik. On the performance of group key agreement protocols. In *Proceedings of the 22<sup>nd</sup> IEEE International Conference on Distributed Computing Systems*, June 2002.
- [3] L. Bassham III. NIST efficiency testing of round 2 AES candidate algorithms. Presentation, available at <http://csrc.nist.gov/encryption/aes/round2/conf3/presentations/bassham.pdf>.
- [4] M. Bellare and P. Rogaway. Optimal asymmetric encryption – how to encrypt with RSA. In *Advances in Cryptology – Eurocrypt 94 Proceedings*, Lecture Notes in Computer Science Vol. 950, A. De Santis ed., Springer-Verlag, 1995.
- [5] M. Bellare, A. Desai, D. Pointcheval and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - Crypto 98 Proceedings*, Lecture Notes in Computer Science Vol. 1462, H. Krawczyk ed, Springer-Verlag, 1998.
- [6] E. Bresson, O. Chevassut and D. Pointcheval. Provably authenticated group Diffie-Hellman key exchange – the dynamic case. *Lecture Notes in Computer Science*, 2001.
- [7] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology – Eurocrypt 94*, May 1994.
- [8] Cypherguard home page. <http://www.cypherguard.com/default.html>.
- [9] D. Dolev, C. Dwork and M. Naor. Nonmalleable cryptography. *SIAM J. Comput.* 30(2): 391-437 (2000).
- [10] K. Fu, M. Kallahalla, R. Rajagopalan, and R. Swaminathan. Secure rotation on key sequences. To be published.
- [11] E. Fujisaki, T. Okamoto, D. Pointcheval and J. Stern. RSA-OAEP is secure under the RSA assumption. In *Proceedings of Crypto' 2001*, LNCS vol. 2139, Springer-Verlag, 2001, pp. 260-274.
- [12] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM Conference on Computer and Communications Security 2000*: 235-244, November 2000.
- [13] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. Cryptology ePrint Archive, Report 2002/009, 2002.
- [14] D. McGrew and A. Sherman. Key establishment in large dynamic groups using one-way function trees. Technical Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, MD, May 1998.
- [15] V. Shoup. OAEP reconsidered. In *Crypto' 2001*, LNCS, Springer-Verlag, Berlin, 2001.

[16] SIRC web page. <http://www.sirc.hu/english.html>.

[17] M. Steiner, G. Tsudik, and M. Waidner. Diffie-Hellman key distribution extended to group communication. In *Third ACM Conference on Computer and Communication Security*: 31-37, March 1996.

[18] Trillian review web page. [http://www.dooyoo.co.uk/computers/applications/trillian/\\_review/386679/](http://www.dooyoo.co.uk/computers/applications/trillian/_review/386679/).

## Appendix

Output from “openssl speed” on a Pentium III 1.2 GHz machine with 512MB RAM:

```
OpenSSL 0.9.6 24 Sep 2000
built on: Thu Sep 12 22:09:51 EDT 2002
options:bn(64,32) md2(int) rc4(idx,int) des(ptr,risc1,16,long) idea(int) blowfish(idx)
compiler: gcc -fPIC -DTHREADS -D_REENTRANT -DDSO_DLFCN -DHAVE_DLFCN_H -DL_ENDIAN -
DTERMIO -O3 -fomit-frame-pointer -m486 -Wall -DSHA1_ASM -DMD5_ASM -DRMD160_ASM
The 'numbers' are in 1000s of bytes per second processed.
type                8 bytes      64 bytes      256 bytes      1024 bytes      8192 bytes
md2                 855.57k       2393.92k       3245.74k       3561.47k       3667.29k
mdc2                2320.20k       2409.02k       2424.83k       2426.20k       2430.29k
md4                 12120.64k      68029.10k     138900.22k     188772.86k     211359.06k
md5                 10531.64k      56477.53k     112707.58k     148943.87k     161125.72k
hmac(md5)           3943.18k       26643.31k     72657.58k     126332.93k     155484.16k
sha1                 6718.99k       32489.00k     58335.23k     72806.40k     78482.09k
rmd160              5633.75k       26121.39k     45411.33k     55707.31k     59902.97k
rc4                  76486.26k     103899.11k    110335.57k    111440.21k    111856.30k
des cbc             20585.01k      22704.51k     23165.78k     23229.44k     23221.59k
des ede3             7860.64k       8197.82k       8254.04k       8265.39k       8243.88k
idea cbc            13122.57k      15375.74k     15662.59k     15710.89k     15660.37k
rc2 cbc             7883.02k       8603.56k       8691.63k       8712.53k       8716.29k
rc5-32/12 cbc       51024.67k      64960.00k     67293.01k     67919.19k     68258.47k
blowfish cbc        30008.69k      34394.76k     35177.30k     34989.74k     35160.06k
cast cbc            29933.07k      33997.99k     34759.08k     34965.16k     35039.91k

                sign      verify      sign/s  verify/s
rsa 512 bits    0.0012s    0.0001s    848.3   8661.0
rsa 1024 bits   0.0065s    0.0003s    154.5   2864.0
rsa 2048 bits   0.0400s    0.0012s    25.0    840.1
rsa 4096 bits   0.2735s    0.0042s    3.7     239.2

                sign      verify      sign/s  verify/s
dsa 512 bits    0.0012s    0.0015s    828.0   683.6
dsa 1024 bits   0.0035s    0.0043s    286.7   234.1
dsa 2048 bits   0.0116s    0.0140s    86.1    71.4
```