Journal of Modern Science and Technology
Vol. 2. No. 2. September 2014 Issue. Pp.99-112

Secure Messaging in Friend to Friend Computing using Multi-party OTR

Mohammad Jabed Morshed Chowdhury[1], Narayan Ranjan Chakraborty[2] and A.K.M Fazlul Hoque[3]

Popularity of social networking and Peer to Peer (P2P) computing opens a new door for many computing solutions such as file sharing, multimedia broadcasting etc. Friend to Friend (F2F) is a desktop grid computing framework which uses instant messenger (IM) and P2P computing solutions to provide lightweight grid computing. But the success of grid computing in open environments like the Internet is highly dependent on security mechanism in the system. Users want their data and processing must be secured in grid environment. Without these mechanisms, users with data-critical applications will never rely on desktop grids, and will rather prefer to support higher costs to run their computations in closed and secure computing systems. This paper demonstrates how multi party Off-The-Record (OTR) protocol can be used to secure the communication in collaborative framework (e.g., F2F framework).

Keywords: Instant messaging, Off the Record, Perfect Forward Secrecy, Friend-to-friend computing.

1. Introduction

Grid computing is distributed computing framework that enables integrated and collaborative use of computers, networks, databases and other scientific entities owned by different organizations. Communities dealing with computational problem which requires large amount of computing resources, secure data sharing across organizational boundaries or both are typical grid computing user (Schwiegelshohn et al, 2010).

One of the ideas of grid computing is using software/middleware to divide a job (computational problem) among several computers which belong to a virtual cyber community. Many software toolkit and systems have been developed, most of which are academic research projects, all over the world.

Most of the grid middleware suffer from one major drawback: they are complex which requires huge administrative e ort to keep the infrastructure running. This might be useful for large organization but not to the small communities especially individual researchers. This leads to the development of lightweight grid solution such as desktop grid. The idea behind desktop grid is to use the desktop computers or notebooks

[1]Mohammad Jabed Morshed Chowdhury, Department of Computer Science and Engineering, Daffodil International University, Bangladesh, E-mail: jabedmorshed@gmail.com
[2]Narayan Ranjan Chakraborty, Department of Computer Science and Engineering, Daffodil International University, Bangladesh, E-mail: narayan@daffodilvarsity.edu.bd
[3]A.K.M Fazlul Hoque, Daffodil International University, Bangladesh,
E-mail: rhoque@daffodilvarsity.edu.bd

connected to the internet as a grid computing node. However, most of the desktop grid solutions still suffer from complex administrative effort.

Interestingly, popularity of social networking and Peer to Peer (P2P) computing opens a new door for many computing solutions such as file sharing, multimedia broadcasting etc. Great advantages of these solutions are that they are ad hoc in nature which makes it lightweight in terms of administrative over-head.

Friend to Friend (F2F) is a desktop grid computing framework which uses instant messenger (IM) and P2P computing solutions to provide lightweight grid computing (Norbisrath et al, 2008). Instant messenger is one of the most popular web technologies. F2F uses social contacts from the IM to create a grid to solve computational problems. Usage of IM contacts brings easier grid creation and certain of level of trust in the system as there is an implicit trust among the friends in IM.

Although trust level in F2F is higher than many other desktop grid solution but it cannot be declared secure unless some cryptographic solution is incorporated to the framework. Right now, security issues is yet to be dealt in F2F computing. This paper discusses the issues related to security implementation in F2F computing mostly in terms of a popular IM security protocol, Off The Record (OTR) (Goldberg et al, 2009 and Borisov et al, 2004) Section 2 gives outline of the requirement for instant messenger (IM). Section 3 gives outline of the family of OTR messaging protocols. Section 4 describes the F2F framework. Section 5 presents the security requirements for the F2F messaging and probable design for secure F2F messaging. Section 6 concludes the review with a brief discussion of the proposed solutions.

2. Security for Instant Messenger (IM)

Electronic Mail (or email) allows people around the world to communicate in a fast and very efficient way. Due to its success, nowadays the email system is entrusted with all forms of information, including very sensitive data. As a consequence, the need to secure the email infrastructure has received plenty of attention, and solutions such as PGP (Zimmerman, 1995) are widely available. Such solutions are designed to provide with the three pillars of secure communications, namely: confidentiality, integrity and authentication.

As it happened with electronic mail, this IM is also going through a second phase of maturation. In particular, the use of IM technology both in private personal communications as well as in sensitive business settings requires that security protection be added to these tools. While many of the popular IM solutions, such as MSN Messenger and Yahoo, do not provide any means of protection, others have already started adding security functionality to their IM programs. Such is the case of AOL that provides protection similar to s/MIME and Trillian's SecureIM which others data encryption but not authentication. These initial examples indicate a growing awareness for the need to secure IM applications. Before continuing it is worth pointing out that existing security tools such as SSL and VPNs can provide protection of IM traffic between servers but are insufficient in most implementations to secure the traffic between the end points to the communication.

Such an end-to-end security is required to protect personal communications as well as needed in many business scenarios. An important step towards the establishment of widespread mechanisms for protecting end-to-end IM communications has been taken by (Borisov et al, 2004).They developed a security design for IM based on the premise that not only IM requires end-to-end security but the specification of the medium and its applications require special security considerations. In particular, they observe that well established end-to-end tools such as PGP (designed to protect email traffic) fall short of providing the security functionality required in the IM setting. Interestingly, while more demanding, the interactive character of the IM setting also presents some advantages over off-line settings such as email.

Indeed, in the case of email, when Alice wants to send an email message to Bob, she cannot engage in a security exchange with Bob (who may not be connected to the network at all), but rather Alice will use Bob's public key to encrypt the message before sending it. An adversary (Eve) can now sit on the network and collect all these encrypted emails, which at this point are useless to her. But what if at some future point Bob's private key gets compromised and exposed to Eve? This is a realistic possibility given that many of these keys are not well protected (insecure storage, easy to guess PINs or passwords, etc.). With the knowledge of Bob's private key, not only can Eve read Bob's future communications, but she can also read any of the past messages addressed to Bob that were protected under this key! (All is needed is that Eve recorded the encrypted communication.)

In interactive settings as IM, this drawback can be avoided by using systems that provide the so called "perfect forward secrecy" (PFS) (Guenther, 1989 and Diffe et al, 1992). In this type of systems, the exposure of a secret key (even a long-term private key) doesn't compromise the security of past transactions. The basic idea is that long term secret keys are used for the purpose of authentication only, while encryption itself is performed using short-term session keys that are erased at the end of a session. As long as these session keys are not recoverable from past transcripts and the long-term secret keys then PFS is ensured. The interactive nature of IM systems allows designing a security solution that achieves PFS.

Another issue to consider is privacy, intended not just as the desire to keep the contents of communications secret (we refer to the provision of such secrecy as confidentiality), but also the desire of keeping private the very fact that a communication between two given individuals ever took place. In particular, cryptographic means applied for securing the Communications should not leave a proof verifiable by a third party (say a judge) that a particular conversation took place. Ideally, any party to a conversation should be able to deny its contents even if the peer to the communication purposely misbehaves to obtain such a proof. Obviously, this requirement conflicts strongly with the desire for authenticity: if Alice requests a proof that a certain message came from Bob, how can Bob later deny that he ever sent that message?

Again if the communication is interactive, there are solutions to this quandary. The idea is that Bob provides a proof that is convincing only for Alice and nobody else. As said, even Alice should not be able to later prove that the exchange ever took place. This kind of authentication, with a "deniability" feature as above, is called

deniable authentication. The conflict between authentication and deniability features is best illustrated by digital signatures whose main feature (and the main reason to use them in some applications) is the provision of non-repudiation (namely, the ability to prove to a third party that a signer committed to a document or other piece of information). However, nonrepudiation is the opposite of what deniability asks for. Techniques for supporting deniability are provided by encryption functions and message authentication codes (MAC). An early example of a protocol providing support for this feature is SKEME (Krawczyk el al, 1996) which was designed in order to provide a deniability option to IPsec's IKE protocol. In recent time, deniable authentication has received some more formal attention in the cryptographic literature (Krawczyk el al, 1996, Boudot et al, 2001, Raimondo et al, 2005 and Goffart, 2013).

3. Off the Record (OTR) Protocol

The original version of OTR was developed in 2004 by Nikita Borisov, Ian Gold-berg and Eric Brewer to provide secure IM communication between two parties. The target of the protocol is to provide encrypted, authenticated and forgeable messaging with perfect forward secrecy. The protocol works as follows:

The two parties (i.e Alice & Bob) who wish to have a secure private con-versation, start with symmetric key generation which will be used for message encryption. This key has to be short lived so that compromising the current key doesn't compromise the security of the future sessions. This is achieved through Diffe-Hellman (DH) key exchange. As DH key generation suffers from man in the middle (MITM) attack, the parties sign their initial key exchange messages (g^x & g^y) using long lived public keys. After the key exchange part is done, both the parties posses a shared key (g^{xy}). To alleviate the problem of jeopardizing secure messaging by compromising shared key, the parties must re-key frequently (use new x and y). Since the keys change quickly, key IDs are used so that both parties can identify which pair of exponents is used in any given message. In any given time, a new key is sent to the other party when message using previous key is received from him. At the same time, previous key is erased from the system. This achieves perfect forward secrecy.

Messages are encrypted with the hash of the current key using AES in counter mode algorithm. For integrity checking, each message also contains a message authentication code (MAC) calculated by SHA1-HMAC algorithm. MAC key is generated from the hash of the encryption key. Signed shared secret in the first phase and MAC in the later phase provides authentication of the parties for every messaging. After deleting a key from the system, the parties publishes MAC key in public. This accomplishes deniable authentication as anyone can forge a previously generated message (Schwiegelshohn et al, 2010).

Raimondo found an attack on OTR (Raimondo et al, 2005). They pointed out that identity misbinding attack on DH will work on the OTR during the initial key exchange. This attack allows an adversary, eve to impersonate as Bob to Alice. This weakness is overcome by introducing a variant of SIGMA protocol during the first key exchange phase (Alexander et al, 2007). OTR version 2.0 is published in 2005, which comprises the improvement against the misbinding attack. Interestingly, the protocol still suffers authentication problem if eve the adversary controls the IM server

(Goffart, 2013). This problem is solved by including socialist millionaire authentication mechanism (Boudot et al, 2001) in the protocol. The detail of version 2.0 is given in (Goffart, 2013).

3.1 Multiparty OTR

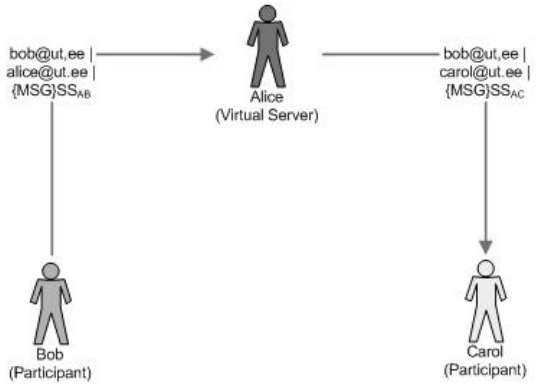
Multi-party chat is an integral part of instant messaging. But original design of OTR doesn't work on multiparty secure communication. Recently, two multi-party OTR solutions have been published: Group OTR (GOTR), Multi Party OTR (MPOTR). These schemes are important in light of secure F2F communication design. Therefore they are discussed in the following sections.

Group OTR is basically extension of OTR 2.0 with multiparty support. The encryption and MAC key exchange and management is same as OTR 2.0. The design concentrates on a virtual server who is actually the initiator of the chat room. The basic idea is that the server creates encryption and MAC key for every other member. Therefore, every member has a key shared with the server and server has multiple keys corresponding to each member excluding himself. Other members doesn't have shared key between themselves. Therefore they have to communicate with each through the server. The protocol works as follows (Seker et al, 2007)

When a member sends a message to the server he simply uses his shared key. When a member sends message to other non server members, he encrypts the message using his shared key and sends it to the server. The server then decrypts using the corresponding shared key and re-encrypts with the intended recipients key. After that he sends the encrypted message to the intended recipients. So the server actually acts as a bridge between the non server parties. For correct routing, the destination email address is added in front of the encrypted message. In case, the message is aimed for non server member, adding recipients email address creates confusion. Because, the recipients sees his email address and tries to decrypt the message and gets garbage and server sees that message is not for him and he drops it. As said earlier, as the message has to be processed by server at first, therefore server's email address is used as a destination email address in the message. When server re-encrypts the message he concatenates the email address of the actual recipient with message.

Message originator identification is also important, as a receiver has no way of knowing whether the received message is sent by the server or other member. To remit this problem, the sender also adds its email address with message. Finally, there is no way of identifying the initiator of chat in IM design. So, every member keeps a configuration which contains the information of virtual server with other security parameters. To eliminate rekeying overhead with multiple parties, rekeying is not done in a single chat session. Communication between non server parties is illustrated in figure 1.

Figure 1: Communication between Non Server Parties in GOTR



The protocol is basically multiple replications of two-party OTR with one virtual server acting as a central mediator. The protocol has several major flaws which are described in the following list:

1. The protocol implementation doesn't support new member inclusion in the middle of a chat session.

2. The protocol deeply relies on virtual server which brings classical central point of failure problems. If the server is down for some reason, the chat session dies and other members cannot communicate with each other further. Also, if the server is a malicious party or attacked by malicious party, the security of the whole system fails.

3. Multi party chat is a broadcast communication scheme where every party learns every message. But this doesn't hold true for GOTR implementation.

MPOTR: Cryptographic design of OTR actually makes it impossible to implement it in multi party chats effectively. One of the reasons is usage of MAC, which actually fails to prove authentication in multi party session. Therefore MPOTR dropped the cryptographic design of the OTR, keeping only the theme of OTR, four properties of OTR: encryption, authentication, deniability and perfect forward secrecy. MPOTR also provide anonymity as well. Unlike GOTR, MPOTR considers all parties (initiator, other participant) equal. The protocol uses a shared secret, calculated from each member contribution for encryption and ephemeral public/private key pair for signing for authentication. The protocol has three phases:

1. Setup
2. Communication
3. Shutdown

The first phase is used to generate key for encryption and authentication. The second phase is used for encrypted and authenticated message exchange and the last phase is used for session termination. Unlike the original OTR, rekeying is not frequent, rather lives as long as the session is active. However, the term session is not same as chat session. In this case, a new session starts when someone decides

to leave or someone decides to join the chat room. A brief description of each phase is given below:

In the first phase, all the parties start by broadcasting a newly generated random number (X). After collecting the random number from all the other parties, a session id is calculated using the random number and identification of the parties. This identification is an alias. Throughout the session, the parties don't reveal their actual identification which gives the protocol the support of anonymity. The parties then generate their own ephemeral public/private key pair for signing messages and share the public exponent with each other. The verification of signed message provide authentication. Then every party calculates a shared secret using the session id and participant identification. At the end of this phase, every party goes through attestation phase where each party calculates hash of all unauthenticated parameters shared with each and broadcasts the hash value. Each party compares all the hash value with its own calculated value and terminates the session if any of the comparisons fail.

The protocol then moves to communication phase. At this stage, a message sender encrypts the message with the shared key. The party also signs the session id and the encrypted message and broadcasts his session id, encrypted message and the signature. When other parties receive this message verify the session id using his own session id and the signature using the session id and the encrypted message. If both of the verifications are successful, they decrypt the message.

When, anyone of the party wants to leave the session or a new party wants to join the session, the protocol goes to shutdown phase. In this phase, the parties collect consensus from all the other parties to check whether every party has received all the messages sent so far. This done by computing hash of all the messages received and generated so far. If the consensus is same for all the parties they send end message to terminate the session. Upon receiving end message, all the party publishes the private exponent of the signing key pair and other parameters in public. This achieves deniable authentication. If a new session needs to be started, the protocol starts from the setup phase once again to create new encryption and authentication key. This achieves perfect forward secrecy.

Compared to GOTR, MPOTR is sound in design and follows the properties or regular chat based communication. Implementation of MPOTR is yet to be published and effectiveness of MPOTR cannot be verified until then.

3.2 OTR Implementations

The first version of two-party OTR was implemented on GAIM. Currently OTR comes with Pidgin IM. But it is supported in many other messengers by following three mechanisms.

1. Native
2. Plug-in
3. Proxy

OTR is directly included in Adium, CenterIM, Sip Communicator, MCabber and some other instant messengers. It must be noted that Sip Communicator uses OTR4J library which is developed by other group based on the original design. Pidgin, Miranda, Trillian and some other instant messenger supports OTR through plug-in. AOL, iChat, Proteus and some other messengers supports OTR via proxy application (Borisov et al, 2004). In the case of multi-party OTR, only GOTR has been implemented on the GAIM messenger although the implementation is still in its beta phase. MPOTR implementation has not been published yet.

4. F2F Computing

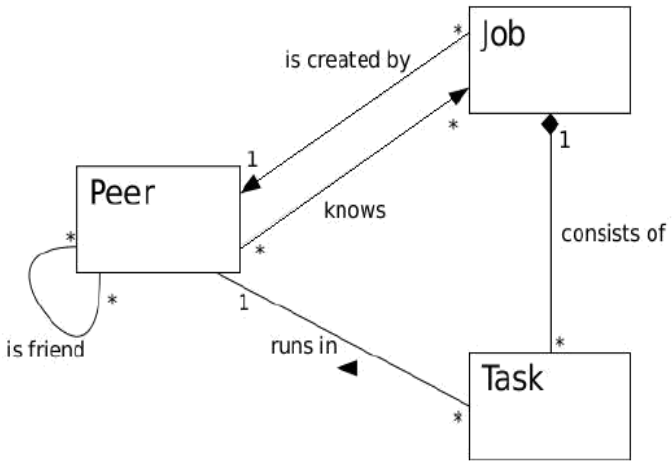
Friend to Friend Computing (F2F) is a lightweight desktop grid computing framework designed by Ulrich Norbischrath. In grid computing a computational problem is divided into small tasks and each task is then distributed to different computers where they are being processed separately. Final solution of the job is obtained by collecting the solution of all the tasks from the different computers. In desktop grid computing this is done by distributing the task in the different PCs or similar machines spread around a network or internet. Well known desktop grid computing solutions are Einstein@home (Einstein, 2013) SETI@home (Korpela et al, 2001) etc. These solutions suffer from overly complicated administrative overheard of grid creation and management. F2F framework offered grid computing solutions with comparably less overhead of grid management by incorporating technologies from P2P computing and instant messaging. A grid in F2F is formed using the contacts in instant messengers of a person. Motivation for this design is driven by the popularity of social networks. As there is a pre-established trust among the friends in a messenger based contacts, trouble of group management is smaller if the group can be established from those contacts. Following section briefly describes the design of the F2F computing.

4.1 F2F Computing Framework

The main idea of F2F is to use instant messenger as a grid creation application using the contacts in the friend list so that administrative efforts are minimized. Key concepts of the framework are Peer, Job and tasks which is illustrated in figure 2.

Peers are the friends from the F2F enabled messenger contacts. These peers solve sub-problems known as tasks belonging to a certain job. One of the peer controls the group management and task distribution. F2F computing framework is constructed based on these relations. The framework has two versions: F2F 1.0 and 2.0. Each version is briefly discussed as follows. Figure 3 shows the layout of F2F 1.0 (Norbisrath et al, 2008). As depicted in figure 3, F2F 1.0 architecture has three layers:

Figure 2: Relation between Peer, Job and Task



- Connection
- Computation
- Application

The first layer provides the connection that is used to exchange messages between the peers of a grid. Second layer is the key layer as it houses the core functionality that implements the grid management and job processing through the help of application layer APIs. Application layer contains the APIs that are used to create grid among friends, terminate grids, create jobs, and distribute tasks, message communication between the tasks and some other actions. F2F 1.0 implementation supports IM, TCP and UDP (with NAT traversal) communication protocol. Grid and job management is done by modified SIP Communicator API.

F2F 1.0 has several major design limitations. To make the framework more flexible F2F 2.0 is devised. The architecture of the improved framework is shown in figure 4. F2F 2.0 includes an additional layer called adapter layer. This layer provides flexible communication between application layer and computation layer. Computation layer uses a couple of buffers for improved message communication. F2F 2.0 uses F2F API which is built upon pidgin IM and a headless python API which can be used in console. The application development is still on process and current implementation contains only IM communication support. TCP and UDP (with NAT traversal) will be implemented very soon.

4.2 Grid management in F2F

Grid setup in F2F varies from version to version. In this case we will discuss only the F2F v2.0 grid setup and management mechanism. Grid setup and management of F2F 2.0 is illustrated in the sequence diagram showed in figure 5. Simplified explanation of the system is as follows:

Figure 3: F2F 1.0 Architecture

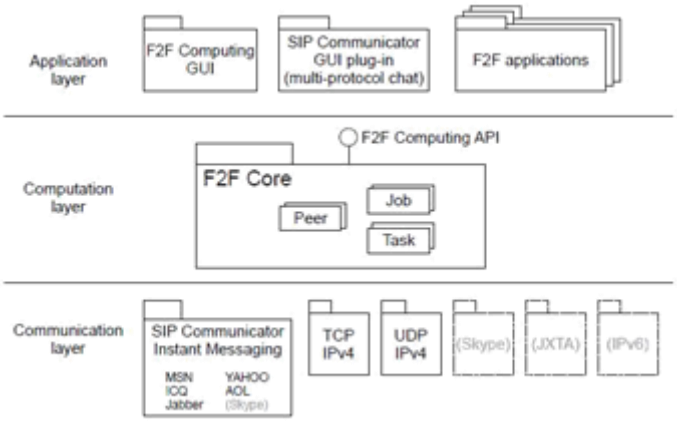


Figure 4: F2F 2.0 Architecture

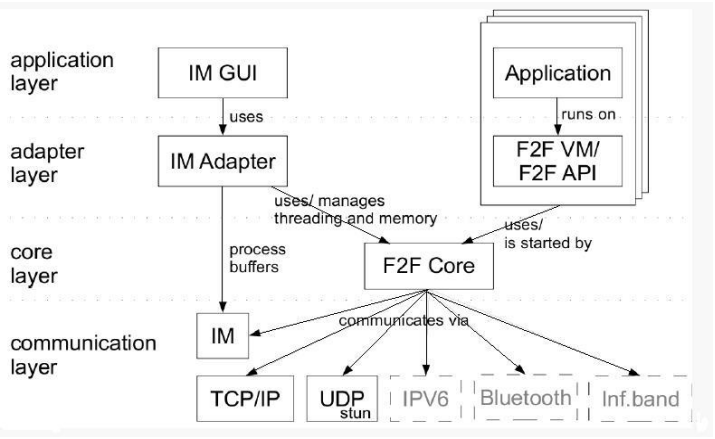
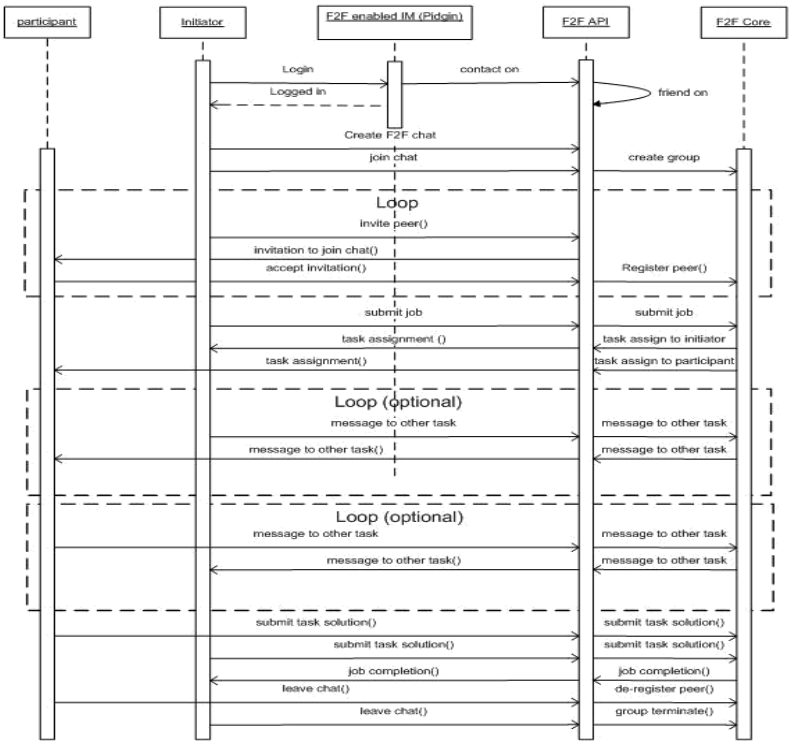


Figure 5: Sequence Diagram of Grid Management in F2F 2.0



A user who wants to solve a computational problem in a grid logs in to his F2F enabled application (pidgin IM). After logging he has to start a chat group for grid creation. A group will be registered in the core when the initiator joins the chat himself. Then he will see a list of meta-contacts/peers who also have the F2F enabled application and logged in to the application at the moment. He will invite his peers to the chat and every peer will be registered to the F2F core. After the group has all the intended participants, initiator will submit the job to the core. Core will divide the job into tasks and distribute among all the participants and the initiator. During task processing, tasks can communicate among them through the core. When a task is completed, each corresponding party submits the result to the core. Upon receiving all the task solutions, the core accumulates the solution and submits the result to the initiator. When a peer leaves, he is de-registered from the group. When initiator leaves, the core terminates the group and declares end of the grid.

4.3 Message communication in F2F

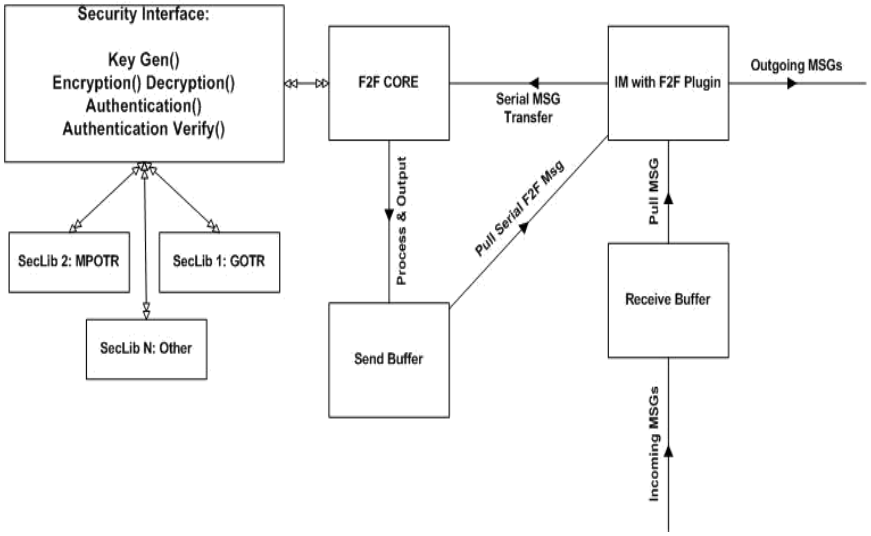
Message communication system is also changed in F2F 2.0 compared to F2F 1.0. The major changes in this case are the use of send and receive buffers between the layers. Currently the system supports only IM communication protocol. Block diagram in figure 6 illustrates the message communication mechanism. The mechanism is explained as follows: All the messages are processed in F2F core. Instant messenger enabled with F2F plug-in acts as an intermediary between the local core and remote peers. All the messages from the remote applications are stored in the receive buffer. IM plug-in pulls messages serially from this buffer to the sent buffer. F2F core pulls messages from the sent buffer serially and process them according to their type. After processing each message, F2F core pushes the messages into sent buffer once again. F2F plug-in pulls this message from the buffer and submit it to the receive buffer to forward to the intended recipient.

5. Secure Messaging in F2F Computing

F2F computing framework has a pre-established trust in the system because of using known contacts from the IM friends. But the security of the system is still questionable as the design of IM system promotes man in the middle architecture (IM server can see every messages between the parties). More importantly, friends in the messengers are not authenticated. Therefore, anyone can impersonate as a friend. In light of those weaknesses, F2F computing needs cryptographic security which is not present in the current design. We suggest following security requirement for F2F computing:

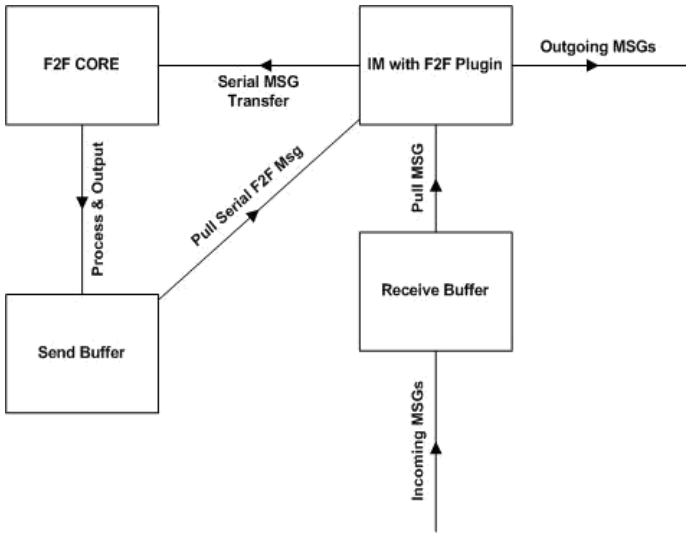
1. Encrypted and authenticated communication between the peers for trusted and private communication.
2. Perfect Forward secrecy so that an adversary cannot obtain security keys at any given instance to compromise the security of system.
3. Deniable authentication is optional.
4. It is practical to use standard o the shelf security API rather than designing one from the scratch.
5. The system should support multiple security protocols so that any desirable o the shelf package can be used at any given instance.

Figure 6: Block Diagram of Message Communication Mechanism in F2F 2.0



As F2F computing framework deeply relies on IM systems (Pidgin in F2F 2.0 and SIP Communicator in F2F 1.0), it seems logical to implement security module in light of IM system security. We have discussed in section 2, about the effectiveness of OTR and OTR variants in the IM environment. However, F2F uses group chat in the IM system. So, it is not possible to use two-party OTR for F2F messaging security. Multi-party solutions based on OTR such as GOTR and MPOTR can be used in this case. Both MPOTR and GOTR protocols are open source products which conform to the ideology of F2F computing. Interestingly, other communication system such as TCP or UDP support will spawn in the F2F design very soon. So, considering only IM security system will be a poor decision. The security module should support other appropriate security protocols as well. In light of this discussion, design depicted in figure 7 is proposed for F2F computing framework.

Figure 7: Proposed F2F Computing Framework with Security Module



The good thing about this design is that it supports expandability. Rather than using a certain cryptographic protocol directly, encryption, authentication and other security requirements are fulfilled by Security interface. Security interface can use any desirable protocol stored in the security library repository. This modular design supports easier upgrading and modification.

6. Conclusion

Incorporating cryptographic solution is not an easy decision. To make a popular application not only they have to be secure but also to be flexible and easy to use. Inclusion wrong type of cryptographic solution makes an application complex and thereby unpopular.

F2F is going through major redesigning phase which makes it difficult to choose the right security protocol. At the moment, it seems that OTR family security protocols are more suited to F2F computing framework. The lesson learned from this project can be used in any collaborative framework.

References

Alexander, C., and Goldberg, I. (2007), "Improved user authentication in off-the-record messaging," *Paper presented at the 2007 ACM workshop on Privacy in electronic society (Pp. 41-47), New York, USA.*

Borisov, N., Goldberg, I., and Brewer, E. (2004), "Off-the-record communication, or, why not to use PGP," *In Proceedings of the 2004 ACM workshop on Privacy in the electronic society (Pp. 77-84), New York, USA.*

Bian, J., Seker, R., and Topaloglu, U. (2007), "Off-the-record instant messaging for group conversation," *In Information Reuse and Integration, 2007. IRI 2007. IEEE International Conference on (Pp. 79-84), Las Vegas, USA.*

Boudot, F., Schoenmakers, B., and Traore, J. (2001), "A fair and efficient solution to the socialist millionaires' problem," *Discrete Applied Mathematics, 111(1), Pp. 23-36.*

Di Raimondo, M., Gennaro, R., and Krawczyk, H. (2006, October), "Deniable authentication and key exchange," *In Proceedings of the 13th ACM conference on Computer and communications security (Pp. 400-409), New York, USA.*

Diffie, W., Van Oorschot, P. C., and Wiener, M. J. (1992), "Authentication and authenticated key exchanges," *Designs, codes and cryptography*, 2(2), Pp. 107-125.

Einstein, Einstein@home. http://einstein.phys.uwm.edu/. Access on 20th august, 2013

Günther, C. G. (1990, January), "An identity-based key-exchange protocol," *In Advances in Cryptology—Eurocrypt'89 (pp. 29-37). Springer Berlin* Heidelberg, Germany.

Goldberg, I., Ustaoğlu, B., Van Gundy, M. D., and Chen, H. (2009), "Multi-party off-the-record messaging," *In Proceedings of the 16th ACM conference on Computer and communications security (pp. 358-368), New York, USA*

Korpela, E., Werthimer, D., Anderson, D., Cobb, J., and Lebofsky, M. (2001), "SETI@ HOME—massively distributed computing for SETI," *Computing in science & engineering, 3(1), Pp. 78-83.*

Krawczyk, H. (1996), “SKEME: A versatile secure key exchange mechanism for internet,” *In Network and Distributed System Security, 1996., Proceedings of the Symposium on (Pp. 114-127), San Diego,USA*.

Norbisrath, U., Kraaner, K., Vainikko, E., and Batrasev, O. (2008), “Friend-to-friend computing-instant messaging based spontaneous desktop grid*,” In Internet and Web Applications and Services, 2008. ICIW'08. Third International Conference on (Pp. 245-256), Athens, Grece.*

Schwiegelshohn, U., Badia, R. M., Bubak, M., Danelutto, M., Dustdar, S., Gagliardi, F., and Von Voigt, G. (2010), “Perspectives on grid computing,” *Future Generation Computer Systems, 26(8), Pp. 1104-1115.*

Zimmermann, P. R. (1995), “The official PGP user's guide,” *MIT press, Cambridge, MA, USA.*