32. Securing Data Transfers: An Integrity Algorithm for Error Recovery Triangulation

Andrew Colarik
AndrewColarik.com
ac@AndrewColarik.com

Jairo Gutiérrez
University of Auckland
j.gutierrez@auckland.ac.nz

Lech Janczewski
University of Auckland
lech@auckland.ac.nz

Abstract

Transferring data is one of the key operations performed by millions of users every day. Users do this by issuing direct commands, such as file transfer commands, or indirectly as a feature invoked by numerous end-user applications. The most important security characteristic of a successful data exchange is the integrity of that data. The receiver user desires to acquire data that has not been modified through malicious acts, or simple human or machine error. Applications that rely on the Transfer Control Protocol (TCP) as the main mechanism to provide end-to-end reliability, including error and sequence control, do not check the integrity of the file being transmitted prior to the transfer. In this paper, we present an overview of current data transfer mechanisms and their security provisions and propose an internal integrity mechanism that provides a triangulation means of error control through the use of one-way hash functions based on the original file being transferred; and a discussion of the implications and limitations that such a mechanism imparts on data transfer mechanisms.

Keywords: Secure Data Transfers, Integrity, Error Recovery, Triangulation, Hash Triplet**.**

Introduction

Data transfers constitute one of the most commonly performed tasks in the Internet today. Users publishing files to a website, moving files that are too large for an email application, transferring files securely between different computers, uploading photos to services such as Flickr (www.flickr.com) or backing up a to a remote server are all engaged in data transfer operations. File Transfer Protocol (FTP), for example, is still used to perform bulk data transfers across networks (Grzywa et al. 2001). It is important to note that this simple-sounding task (data transfer) is essential for the interoperation of networked computers, especially when the different computing environments of modern heterogeneous networks are taken into account. Potential user devices in the network range from Personal Digital Assistants (PDAs) and mobile phones to specialized servers and large-scale corporate systems. All of these devices could operate using different word formats; they might store the data in different forms and forward their packets in non-compatible ways (big-endian vs. little-endian computers) (Tanenbaum 2003).

Ask yourself a few simple questions. Do you know how secure data transfers really are? Can you really trust the file you just received? Was the file you just spent an hour downloading corrupted before you began the download? It is these fundamental questions that drive the continued improvement efforts to provide a data transfer protocol that does its job well and does not waste the user's time, money, and energy.

Let us examine the case of FTP, one of the traditional techniques to perform data transfer. There are many different data transfer protocols but FTP is perhaps the best example for the presentation of our ideas. When a communication is initiated, two connections are established for the session. The first is a control channel that is used to coordinate the connections and bulk transfers, and the second is the data channel (Postel & Reynolds 1985). The authentication of the identity of the users and the application of data and control channel confidentiality outlined by Brown & Jaatun (1992) has been enacted through the use of FTP security extensions. These extensions utilize established security mechanisms, such as Public-Key Infrastructure and Kerberos, through the use of the Generic Security Services Application Program Interface (GSS-API) in order to provide authentication of the user, data transfer integrity, and confidentiality on both the control and data channels (Horowitz & Lunt 1997). It is in the area of file integrity that we propose an alternative solution.

In FTP, the integrity of the file is perhaps the most important security aspect. The user desires to acquire a file that has not been modified through malicious acts, or simple human or machine error. FTP has "no provision for detecting bits lost or scrambled in data transfer" (Postel & Reynolds 1985). It does utilize Transfer Control Protocol (TCP) for this process, coupled with a marker code inserted into the data stream for restarting the transfer when the data is corrupted or the transfer is interrupted. More notably, FTP does not check the integrity of the file being transmitted prior to the transfer. Other applications (such as TFTP and Remote Network File Sharing) have the same weakness. In this paper, we present an integrity mechanism that operates during the data transfer session that provides a triangulation means of error control through the use of one-way hash functions based on the original file being transferred; and a discussion of the implications and limitations that such a mechanism imparts on the data exchange. By "triangulation" we mean introducing a mechanism that points at the location where possible transmission error occurred.

Overview of data transfer mechanisms

There are two main approaches to perform data transfers in today's IP-based networks. One is based on FTP and several variations to that protocol and the second, and much newer, approach is based on peer-to-peer networking configurations. Let's review both of them.

FTP Overview

Essentially, FTP is a client-server protocol that facilitates the transfer of files between two computers connected via a network, such as the Internet (Brown & Jaatun 1992). The protocol has several fundamental components that perform the required functions for establishing the sessions between systems, and coordinates the transfer of the files. These components are as follows:

- Server-FTP that consists of the Server Protocol Interpreter and the Server Data Transfer Process;
- Server-side File System;
- User-FTP that consists of the User Interface, User Protocol Interpreter, and the User Data Transfer Process;
- User-side File System;
- A User; and
- A communication channel for the control and data connections.

The FTP architecture is displayed below (see Figure 1):

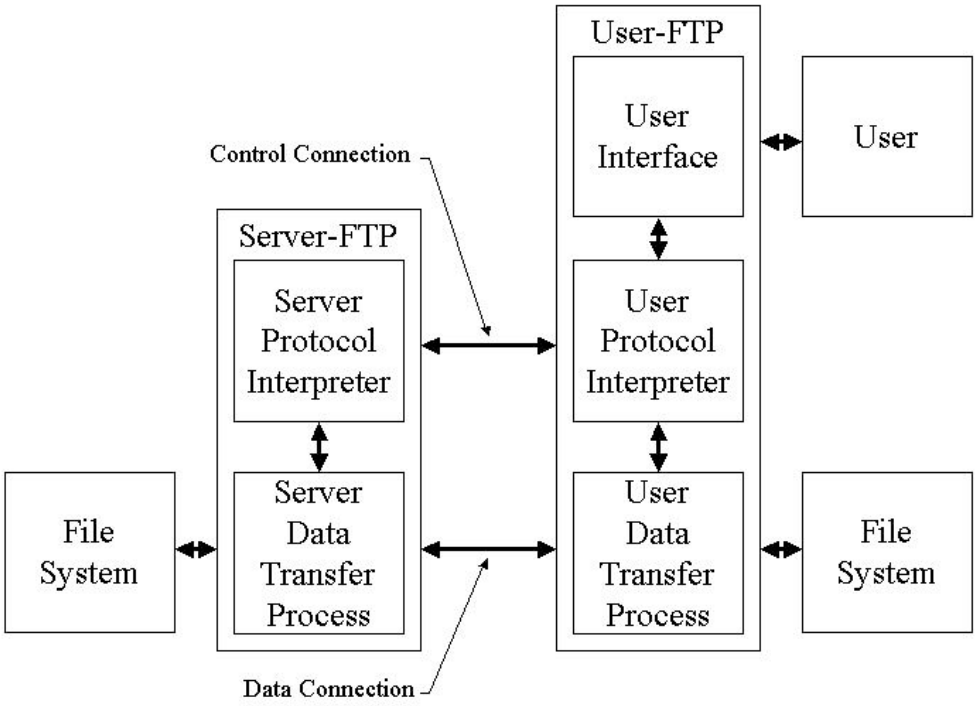

Figure 1: FTP Architecture (Postel & Reynolds 1985)

An alternative configuration of FTP provides for the transfer between two servers through a third system that provides the control function between the servers. In both of the configurations, FTP requires that the control channel be open during the data transfer process (Postel & Reynolds 1985).

Telnet authentication and encryption

As previously discussed, FTP utilizes the control connection to coordinate the data connection and execute commands on the File Systems of both the user and the server. FTP utilizes the Telnet protocol to execute the commands (Postel & Reynolds 1985). This fundamental design lends itself to security breaches that may permit eavesdropping of user ID's, passwords, file names, and other information passed through the control channel. It also may allow an active attacker to change settings and execute commands on the file system (Brown & Jaatun 1992). This fundamental security flaw was initially addressed when Borman (1993) proposed the passing of authentication information, and a mechanism to enable encryption of the data after successful authentication for the Telnet protocol. The result was that user passwords would not be sent in clear text, and the data stream would be encrypted utilizing any general authentication and encryption system. However, it should noted that the "Telnet authentication and encryption option does not provide for integrity protection only (without confidentiality), and does not address the protection of the data channel" (Horowitz & Lunt 1997).

Security Extensions

In 1992, Brown & Jaatun detailed a set of security extensions for the FTP utility in the TCP/IP suite. The extensions provide authentication of the user, and encryption of the data at

varying levels of security for both the control and data channels. In summary, FTP Security Extensions impart the following:

- User authentication that surpasses the original password mechanism;
- Server authentication;
- Session parameter negotiation that includes encryption keys and attributes;
- Command connection protection including transmission integrity, confidentiality of the data and/or control commands, or both; and
- Data transfer protection including transmission integrity, confidentiality of the data and/or control commands, or both. (Housley et al. 2000)

When using security extensions, the systems involved in the transfer process negotiate an agreed security mechanism. Once agreed to, the authentication process begins. The server may require additional security information and invoke a security data exchange (Housley et al. 2000). It should be noted that the systems might not agree on a security mechanism and continue the normal processes due to mechanism incompatibilities or the non-existence of such mechanisms on one or both systems. Provided that the security mechanisms exist and are agreed on, and all needed security information is transferred and confirmed, the two systems are then considered to form a security association. This association will contain the needed information to perform authentication, and keying information for integrity and confidentiality providing they are a part of the previous mechanism negotiated (Horowitz & Lunt 1997). The association is maintained throughout the rest of the session. There are four levels of protection that can be provided in a security association. The first is Clear, which indicates no security is in effect. The second is Safe, which indicates integrity confirmation will be applied to the data. The third is Confidential, which indicates the data will be encrypted. The forth is Private, which indicates integrity confirmation and encryption (Horowitz & Lunt 1997). One major weakness to this approach is that when data transfers are to be authenticated, "the authentication checks are performed on individual blocks of the file, rather than on the file as a whole" (Horowitz & Lunt 1997) allowing insertion attacks in the data stream resulting in corrupted files. FTP Security Extensions rely on the external security mechanisms to correct this flaw (Horowitz & Lunt 1997).

Encryption using KEA and SKIPJACK in FTP

In order to further formalize and develop the security mechanisms available to FTP, Housley et al. (2000) proposed the Key Exchange Algorithm (KEA) in conjunction with the SKIPJACK symmetric encryption algorithm to be incorporated into the security extensions. They further go on to specify the incorporation of the Fortezza Crypto Card to perform these functions.

Peer-to-peer networks

Peer-to-peer networks operate in a distributed fashion and are self-organizing. Many of the disadvantages of their lack of control and management are compensated by the resilience, added processing capabilities and added storage facilities that these networks bring to the wide-area networking space. Schemes such as Pastry (Rowstron & Druschel 2001), Kademlia (Maymounkov et al. 2002) and Viceroy (Malkhi & Mazieres 2002) use overlay configurations to provide self-organization, robustness, routing and reliability capabilities. However the schemes lack security features and this is common to many of the peer-to-peer mechanisms that rely on Distributed Hash Tables (Lua et al. 2005).

BitTorrent, as a centralized P2P system, offers some level of security that is missing from distributed schemes. A centralized server provides hashing information that is used by the peers to download information and verify its integrity (Lua et al. 2005). However, none of the known P2P schemes allow for checking the request for a file (combined with a hash of the file in the request) nor do they check the state of the file *prior* to transmission. Also, because of this lack of security, additional authentication and transfer encryption schemes do not prevent substitution attacks of any given file fragment.

What's missing?

With the exception of a relatively recent proposal of a Resumable FTP (Grzywa et al. 2001), the trend towards the development of FTP's security functions appears to be an expansion of its core interoperability with other protocols and security mechanisms. Efforts to provide security for Web Services rely on Secure RPC at best (W3C 2001) (Safford, Hess & Schales 1993), and this in practice only addresses authentication of issued commands. De-facto standards such as SSH FTP assume that its implementation runs over a secure channel (Galbraith, Ylonen & Lehtinen 2002) and the fingerprinting process used only applies to the authentication process and not the file transfer itself (Friedl 2003). Finally, looking at P2P systems for help is not the answer either: P2P are overlays and they make use of lower level protocols such as FTP. Therefore, what is missing is further development of the core processes contained within FTP itself to provide, what we feel is FTP's principal function, the delivery of the intended file with its integrity maintained while providing a core error recovery process in the event of intended or unintentional file corruption both prior and during the transfer process. Given that "most software is not designed for fail-safe or fail-secure operation" (Grzywa et al. 2001), development efforts need to be directed at re-examining core processes in order to produce self-recovering operations.

Our proposal offers a lightweight approach that is integrated at the protocol level. In the following section, the authors introduce an integrity algorithm known as a Hash Triplet, an overview of the basic process steps performed with integrating the algorithm into FTP, and a proposal for using the algorithm in error detection and recovery. Again, we would like to repeat that the Hash Triplet concept to be presented in the next section may be used for many data transfer protocols and we have chosen FTP protocol as it is well known around the world.

Hash Triplets

The underlying function of a Hash Triplet is to facilitate the enforcement and enhancement of the handshake protocol at a file transfer level. It involves the use of a one-way hash product generator, which resides on both sides of the file transfer. The hash product generator is used to create a unique bit stream (commonly 128-bits) that is based on the original file it was created from. Any changes to the original file will result in a different hash product being generated. There are occasions when two different files can generate the same hash product. This is known as a collision and is undesirable for system continuity. Thus, the choice of the hash product generator must be collision-resistant (low probability of a duplicate being generated), such as MD5, SHA-1, and RIPEMD-160. The basic security handshake process is outlined in Figure 2 (Adapted from Colarik 2003). Hash Triplets may use any hash product generator and recent developments in breaking hash generators (Schneier 2004; CAN 2007) indicates that tools more powerful than MD5 or SHA-1 should be used in the future.

Prior to any FTP session, the file to be transferred is stored with a corresponding, previously generated hash product file (Storage Hash) in the File System. It should also be noted that a unique file name is used in conjunction with the hash product during storage and transfers, and as such, further diminishes the hash product duplication issue (collisions) discussed previously. In step 1, a message containing the file name and its corresponding hash function is sent. In step 2, the message is received, the hash file (Message Hash) is stored, and a request for the file is sent with the previously stored Message Hash. In step 3, the system will generate a new hash (Generated Hash) and compare it with the Message Hash, and the Storage Hash. The results of the comparison will be checked against the Send Request Response Table (see Table 1) for the appropriate action.

If the hash products match, the system will initiate the file transfer and include a hash (Package Hash) in the transfer. If they do not match, the corresponding corrective actions may be displayed to the user or may initiate some error recovery mechanism appropriate to the condition. In step 4, the file and Package Hash is received, and the Hash Triplet function is initiated, and the results compared to the Received File Response Table (see Table 2) for appropriate action.

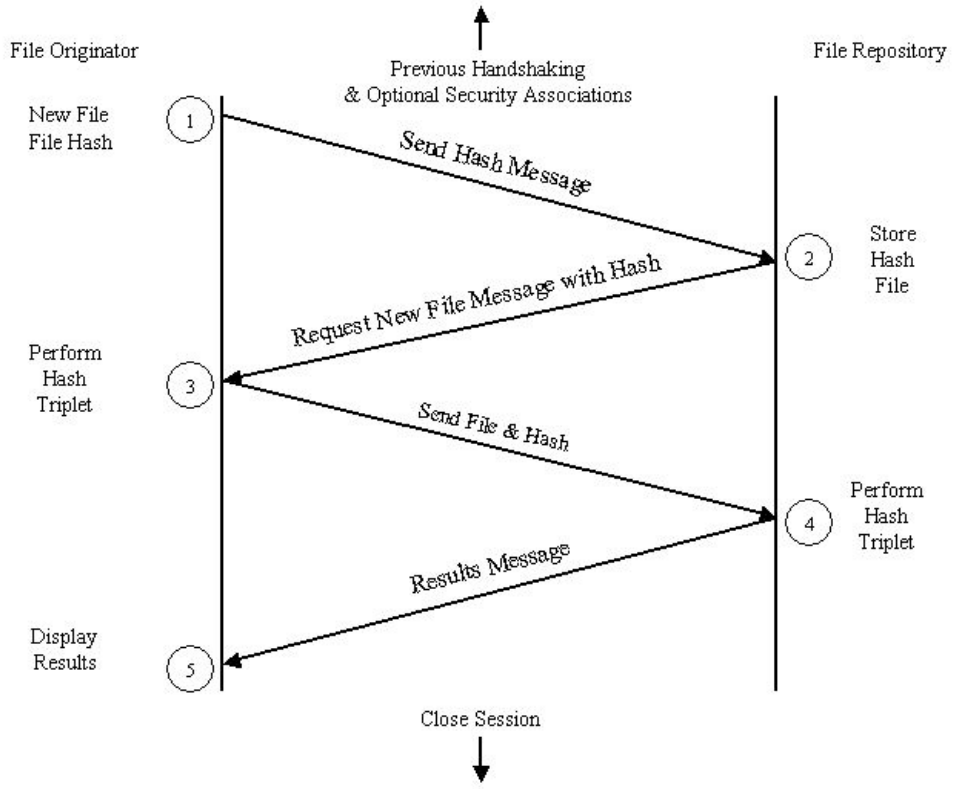

Figure 2: Data Transfer Sequence Diagram

The file is saved if the hash functions match. In any case, the corresponding message for the Hash Triplet is forwarded to the File Originator. In step 5, the message is received. It is in step 5 that an additional mechanism needs to be developed to take corrective actions in the event that an error occurred. Throughout the above process, we believe the needed replies can be integrated into the existing three-digit FTP command structure.

Proof of concept generation discussion

An application was written in Java and utilized a CORBA compliant object request broker provided in the Java 2 Platform Standard Edition development kit 1.4 to demonstrate the concepts presented above. CORBA was selected to provide for the widest range of portability between systems (Object Management Group 1995). The hash function generator contained within the source code was based on the 128-bit message digest MD5 model developed by Rivest (1992). Only the areas shaded in gray in Tables 1 and 2 were applied to the logic of the Hash Triplets. Simple text files were created and their corresponding hash functions were generated and saved in separate folders on two different computers connected via a network. The file transfer process outlined in Figure 3 (steps 1-5) was initiated and tested to determine that each of the Hash Triplet comparisons were either acted upon, in the case of all three hashes matching, or an error message was displayed and the process halted. An example of this would be that prior to the transfer, a text file was modified without the corresponding storage hash being updated. The result was that the process never progressed past step 3 and the error message "File has changed since original message, original hash may not have been updated, confirm file integrity before continuing, regenerate hash and reconfirm" was displayed.

Table 1: Send Request Response

Scenarios				
Send Request Response	Request Hash	Storage Hash	Generated Hash	Action Required / Notification Returned
	h_1	h_1	h_1	Validated file, integrity confirmed, send file
	h_1	h_1	h_2	File has changed since original message, original hash may not have been updated, confirm file integrity before continuing, regenerate hash and reconfirm
	h_1	h_2	h_2	Message to Recipient is invalid or out of date, may be invalid request, request retransmission of message
	h_1	h_2	h_1	Original hash may be corrupted, update hash file, confirm before transmitting file
	h_2	h_1	h_1	Message to Recipient is invalid or out of date, issue new notification, request retransmission of message
	h_2	h_2	h_1	File has changed since original message, original hash may not have been updated, confirm file integrity before continuing, regenerate hash and reconfirm
	h_2	h_1	h_2	Original hash may be corrupted, update hash, confirm before transmitting file
	h_2	h_1	h_3	Re-evaluate all controls for file, discontinue distribution of file, and everyone is going to hell in a hand-basket ☺

Limitations, Considerations, and Implications

This line of research was a direct result of a lateral application of a proof-of-concept developed, as a module, for a secure patch management system. The development and testing of the proof-of-concept was restricted to a small, homogeneous environment in which a limited number of possible combinations were conducted (differing operating systems, file types, file sizes, number of errors introduced, etc.). As a result, this benign environment does not simulate the conditions of a network and its systems that are under attack, operating at or in excess of maximum capacity, or degraded capabilities due to legacy or matured system infrastructure. However, the basic concept has been proved and future developments and experiments are now easy to design provided the resources of time and money are available. This research was a component of PhD thesis work (Colarik 2003) and the financial/time

constrains allowed only the verification of the concept, leaving room for more extensive tests in the future.

Table 2: Received File Response

Scenarios				
Received File Response	Package Hash	Message Hash	Generated Hash	Action Required / Notification Returned
	h_1	h_1	h_1	Validated file, integrity confirmed, store / install / execute / process file
	h_1	h_1	h_2	File has changed at or after transport phase, re-request file
	h_1	h_2	h_2	Transported package maybe corrupted, originator's controls may need to be examined, request retransmission of package
	h_1	h_2	h_1	Original hash may be corrupted, package may be invalid, request notification message again & re-request file
	h_2	h_1	h_1	Package to Recipient may be corrupted, request retransmission of package
	h_2	h_2	h_1	File has changed since original message, original hash may not have been updated, confirm file integrity before continuing, regenerate hash and reconfirm
	h_2	h_1	h_2	Original hash may be corrupted, message may be invalid, request message again and re-request file
	h_2	h_1	h_3	Re-evaluate all controls for file, discontinue file request and discard, and everyone's going to hell in a hand-basket ☺

It is intended that our proposal does not exclude additional security mechanisms that are available for establishing communication sessions, authenticating the users, and validating the activities performed during these transactions (Abadi & Needham 1996). Mechanisms such as Public-Key Infrastructure (PKI), Secure Socket Layer (SSL), and Kerberos should be considered in addition to our proposal in the safe transfer of files between systems. In fact, the integration of keys into the generation of the hash products would assist in authenticating the sender of the file, and as such, will be considered in future development projects.

With this proposal to secure FTP transactions, we have utilized the existing FTP Architecture (see Figure 1). We have done so by including the governance of the File System (at least in part) by simply managing the files' change status through the use of hash functions. In addition, we feel that the incorporation of error triangulation becomes significantly more important when a user conducts file transfers in a server-to-server configuration (see Figure 2). The governance of multiple systems remotely through a third system creates additional opportunities for error that lower level protocols may not recover from efficiently. Recovering from these errors without jeopardizing system integrity requires careful and future consideration.

Conclusions

A basic requirement by users is that files be transferred efficiently and without modification. File integrity, therefore, becomes a critical element in the sharing of files. During the transfer of a file, a self-correcting system would permit the user to acquire a file while the error controls would operate transparently in the background. The key contribution of the Hash Triplet mechanism presented here is that it identifies the specific points of origin for the root error causes and allows system designers to promote a wider range of remedies. No similar system, as far as we know, provides such capability.

We would like to point out that the essence of the Hash Triplet concept is an extension of the pre-transfer file indicator code properties to the file transfer operation. An analysis of received

information allows a receiver of a pre-transfer file indicator coded message to quickly determine not only if the message were transferred correctly but also to quickly locate the possible error. Hash Triplets provide this type of information.

In this paper, we have discussed the Hash Triplet concept using the FTP protocol since, by definition, FTP is a protocol designated for the transfer of bulk files. However, we think that the outlined concept may be applied to many other protocols used for transferring files. Further research could be designed at this goal. As further research developments occur in securing the file transfer process, file integrity and error recovery governance may well need to extend beyond the protocol level and be integrated into any given FTP application program and/or the operating system itself.

References

Abadi, M., and Needham, R. "Prudent Engineering Practice for Cryptographic Protocols," *IEEE Transactions on Software Engineering*, (22:1), 1996.

Borman, D. "Telnet Authentication and Encryption Option," *IETF Internet-Draft*, Telnet Working Group, Cray Research, Inc., April 1993.

Brown, Lawrie, and Jaatun, Martin Gilje II "Secure File Transfer Over TCP/IP," *Proceedings of IEEE Tencon-92*, 1992.

Central News Agency (Taipei) "Chinese Professor Cracks Fifth Data Security Algorithm," *The Epoch Times*, 11 Jan 2007. Retrieved from the www on the 22nd of January 2007. URL:http://en.epochtimes.com/news/7-1-11/50336.html

Colarik, A. "An integrity mechanism for file transfer in communications networks," *New Zealand Provisional Patent Application 525794, U.S. Patent Application No. 10/843,142,* and *Canadian Patent Application No. 2,467,140*, May 2003.

Colarik, A. PhD Thesis: *A Secure Patch Management Authority*, University of Auckland, Auckland, New Zealand, 2003.

Friedl, M. "SSH Fingerprint Format," *IETF Internet-Draft*, March 2003.

Galbraith, J., Ylonen, T. and Lehtinen, S. "SSH File Transfer Protocol," *IETF Internet-Draft*, December 2002.

Grzywa et al. "Application-Level Survivable Software: rFTP Proof-of-Concept," *Proceedings of the 26th Annual IEEE Conference on Local Computer Networks*, 2001.

Hastings et al. "A Case Study of Authenticated and Secure File Transfer: The Iowa Campaign Finance Reporting Systems (ICFRS)," *Performance, Computing, and Communications Conference, 1997. IPCCC 1997, IEEE International*, 1997.

Horowitz, M., and Lunt, S. "FTP Security Extensions," *IETF RFC2228*, Cygnus Solutions, Bellcore, October 1997.

Housley et al. "Encryption using KEA and SKIPJACK," *IETF RFC2773*, February 2000.

Object Management Group "CORBAservices: Common Object Services Specification," March 31, 1995.

Lua, E. K., Crowcroft, J. and Pias, M. "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications Surveys*, Second quarter (7:2), 2005.

Malkhi, D., Naor, M. and Ratajczak, D. "Viceroy: A Scalable and Dynamic Emulation of the Butterfly," *Proceedings of ACM PODC 2002*, 2002.

Maymounkov, P. and Mazieres, D. "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," *Proceeding of IPTPS*, 2002.

Posner, E. and Reichstein, Z. "Configurations for File Transfer Protocol Error Protection," *IEEE Transactions on Communications*, (34:3), 1986.

Postel, J., and Reynolds, J. "File Transfer Protocol (FTP)," *IETF STD9, RFC959*, October 1985.

Rivest, R. L. "The MD5 Message-Digest Algorithm," *IETF RFC 1321*, April 1992.

Rowstron, A. and Druschel,P. "Pastry: Scalable, Distributed Object Location and Routing for Large-scale Peer-to-peer Systems," *Proceeding of Middleware*, 2001.

Safford, D., Hess, D. and Schales, D. "Secure RPC Authentication (SRA) for TELNET and FTP," *Proceeding of the Fourth USENIX Security Symposium*, 1993.

Schneier, B. "Cryptanalysis of MD5 and SHA: Time for a new standard, Crypto researchers report weaknesses in common hash function," ComputerWorld, 19 August 2004**.**

Srinvasan, R. "RPC: Remote Procedure Call Protocol Specification Version 2," *IETF RFC1831*, Sun Microsystems, August 1995.

Tanenbaum, A. *Computer Networks*, 3rd Ed., Prentice-Hall, Upper Saddle River, NJ, 2003.

W3C "SOAP Version 1.2: W3C Working Draft 9," July 2001.

Ylonen et al. "SSH Transport Layer Protocol," *IETF Internet-Draft*, July 2003.