mSSL: Securely Sharing Data from a Server among Clients (*Preliminary version*)

Jun Li and Xun Kang

Department of Computer and Information Science
University of Oregon
{lijun, kangxun}@cs.uoregon.edu

While conventionally a client needs to directly request data from a server, a new trend of data service over the Internet is to allow multiple clients, such as thousands of clients of a web server, to share data among themselves in a peer-to-peer fashion [1–7]. This creates a hybrid communication model that involves both client-server and peer-to-peer communications. If a client has downloaded some blocks of data from a server, other clients can obtain those blocks from this client, rather than the original server (Figure 1). This mechanism can potentially prevent a server from being overwhelmed when serving large audiences, and enable even an under-provisioned site to provide scalable data service.

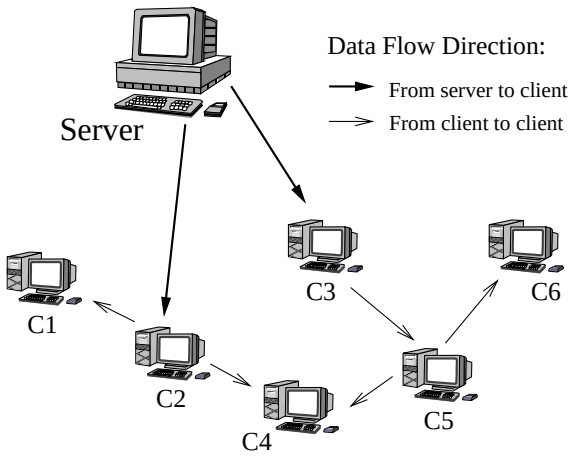

Fig. 1. Peer-to-peer data sharing among clients of a common server

Accompanying this trend, however, are new security challenges that conventional client-server approaches such as SSL [8, 9] cannot address. How can a server allow peer-to-peer data sharing without weakening the access control? When a client retrieves data, whether from a server or its peer clients, can the integrity of the data always be guaranteed? How can the data confidentiality be

supported? If a client provides data to another client, can the former provide a proof about the service that the latter cannot deny? What if a small web server that all of sudden becomes popular cannot certify its own identification?

Corresponding to the conventional SSL service that protects client-server connections, we propose to build a *mSSL* service that protects both client-server and client-client connections in such a hybrid environment. In particular, mSSL will provide a set of security functionalities to enable secure sharing of server data among clients and to support applications that sit on top of mSSL. Both conventional security issues such as access control, data integrity, and data confidentiality and new security issues such as proof of service will be addressed in this new context.

Very importantly, while allowing clients to share data that they originally directly download from a server, the security should not be weakened compared to traditional client-server model, or only weakened to a minimal degree if any. Note that this hybrid environment actually causes the security mechanisms different from those of either a pure client-server or a peer-to-peer environment. Unlike a client-server environment, a server does not directly control all the data flow. A provider client, for example, has to decide whether to provide data to a recipient client. Or, a recipient client must verify the data from a provider client. Also unlike a pure peer-to-peer environment, security solutions in a hybrid system can take advantage of the existence of a server, enabling a potential integrated "centralized plus distributed" approach.

We are in the middle of evaluating the performance of mSSL. The security functionalities provided by mSSL may be used in different combinations for different applications, and the success of mSSL will depend on its efficacy under all those combinations. Of the most particular concerns are the overhead introduced by mSSL and how much an application could be slowed down by mSSL.

In this short paper, we briefly describe our solutions as following:

- ***Access control***: mSSL implements a ticket-based solution to ensure that only authenticated clients may access a data object, no matter where the data object is located. Figure 2 illustrates basic steps for access control. A client must first contact the server and authenticate itself. Here the client and the server can establish a SSL channel for their secure communication. Once the server decides that the client is allowed to obtain the data, for example, after a client paid for purchasing an audio file, the server will either directly transfer the data to the client, or provide a ticket for the client to contact other peer clients. The ticket is signed by the server, and proves that the server authorizes this particular client to download data from other clients, who can verify the ticket before providing data to the client in question.
- ***Data integrity***: To guarantee that an (authenticated) client can verify the integrity of data, no matter where the data is downloaded from, mSSL adopts a solution based on the Merkle Hash Tree [10]. Integrity verification occurs at the block level, so a client does not have to wait until it receives all the blocks of a file before verifying its integrity. At concept level, the (unsigned) hash value of every block of a file is a leaf node on the file's merkle hash tree,

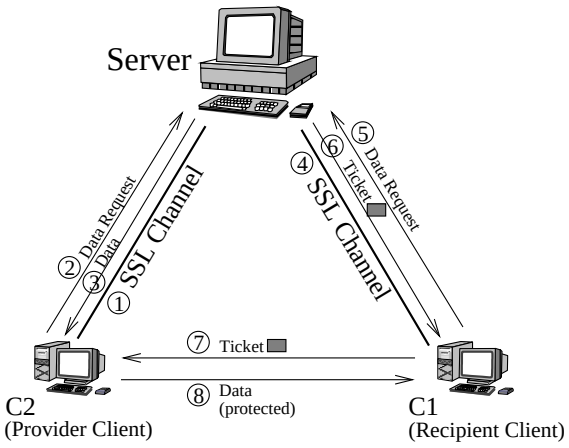

Fig. 2. Ticket-based data access control

and the hash value of the whole file is signed. To verify the integrity of every block, only the hash values on the authentication path of that block need to be obtained [10]. At implementation level, we have developed a super-block-based solution and a per-block-based solution and are comparing their performance. The former provides a special block to contain the hash value of every file block, the latter allows each client to retrieve necessary hash values on demand.

- ***Proof of service***: This mechanism ensures that a recipient client that has obtained data from a provider client or a server cannot deny the data service that it received. Especially, a provider client can prove what service it has provided to others. A simple economy model underpins this service: if a provider client can prove to the server that it has helped its peer clients, it will receive credits for doing so, and can probably have higher priority when downloading data from a server. Our basic solution is to enforce an interlocking block-by-block transfer between a provider client and a recipient client. After a provider client verifies the ticket from a recipient client, it will not provide all the data to the recipient; instead, it will only send the first encrypted block. The recipient must acknowledge the receipt of this block before it can receive a key from the provider to decrypt this block and receive the next block. This process repeats for the following blocks. Every acknowledgement must be signed using the private key of the recipient (whose public key can be issued and certified by the server). With all the acknowledgments received, the provider can then compose proof about the data blocks it has provided. Some tricky issues are also handled, such as the acknowledgement of the last data block.
- ***Confidentiality***: In order to ensure that only authenticated clients can access data, mSSL adopts an object-key-based approach for data encryption

and confidentiality. Every data object or a file can have an object key associated with itself and use this key for encryption. Essentially an object-oriented approach, this scheme has the advantage for doing fine-grained file-level access control. Also, every file can only be encrypted once for all potential clients, instead of once per client. Depending on different application needs, data confidentiality can be provided with or without proof of service. When proof of service is needed, confidentiality is already natually implied (see above). Otherwise, when a client requests data from a server, the server can issue the object key to that client after authenticating its data request; as a result, after the client receives the data from either the server or its peer clients, it can use the object key to decrypt those data.

– ***Same stranger guarantee***: When a small web server without a certified public key wants to serve a large number of clients, all its clients can still be guaranteed that they all interact with the same web server, even though they cannot verify its identity. We allow such a server to issue a self-certified certificate of its own public key, and every authenticated client can use this certificate to verify that it is interacting with the *same* stranger as its peer clients.

In summary, while SSL provides protection for conventional client-server communication, mSSL allows multiple clients to securely share data from a server in this new context and protects both client-server and client-client communication. We believe mSSL's design, once evaluated to be successful, will provide a strong and flexible support for addressing both conventional security issues such as access control, data integrity and data confidentiality and new security issues such as proof service, thus strengthening a wide range of Internet applications.

References

1. Bittorrent. http://bitconjurer.org/BitTorrent/.
2. Daniel Stutzbach, Daniel Zappala, and Reza Rejaie. Swarming: Scalable Content Delivery for the Masses. Technical report, 2004.
3. University of Oregon. Swarming: Building Adaptable and Scalable Content Delivery for the Masses. 2003.
4. Angelos Stavrou, Dan Rubenstein, and Sambit Sahu. A Lightweight, Robust P2P System to Handle Flash Crowds. In *Proceedings of the 10th IEEE International Conference on Network Protocols*, pages 226–235, 2002.
5. Keith Kong and Dipak Ghosal. Mitigating Server-Side Congestion in the Internet through Pseudoserving. *IEEE/ACM Trans. Netw.*, 7(4):530–544, 1999.
6. Venkata N. Padmanabhan and Kunwadee Sripanidkulchai. The Case for Cooperative Networking, 2002.
7. Rob Sherwood, Ryan Braud, and Bobby Bhattacharjee. Slurpie: A Cooperative Bulk Data Transfer Protocol, 2004.
8. SSL. http://wp.netscape.com/eng/ssl3/ssl-toc.html.
9. Eric Rescorla. *SSL and TLS: Designing and Building Secure Systems*. 2000.
10. Matt Bishop. *Computer Security Art and Science*. 2003.