A Security Pattern for Data Integrity in P2P Systems

Benjamin Schleinzer
RWTH Aachen University
Chair of Communication
and Distributed Systems
Ahornstr. 55
52074 Aachen, Germany
Email: schleinzer (at) comsys.rwth-aachen.de *

Nobukazu Yoshioka
GRACE Center, National Institute of Informatics
2-1-2 Hitotsubashi
Chiyoda-ku, Tokyo
Email: nobukazu (at) nii.ac.jp

ABSTRACT

Peer-To-Peer-systems (P2P) introduced new methods to distribute large amounts of data to end users. To increase the distribution speed resources from all participating network nodes, the peers, are used, and therefore the workload on own resources decreases. To utilize all peers large data is split into small pieces, so called chunks, and these chunks are distributed among peers therefore making each chunk available on different peers. To identify and find chunks in P2P-systems hash algorithms are used, and each peer is responsible for a specific range of the hash's keyspace and all chunks that fall within this keyspace.

With data stored on multiple peers new security risks in terms of confidentiality, integrity, and availability arise. Our security pattern targeted specifically for P2P-systems helps system designers to identify possible threats and show appropriate countermeasures. We show how secure hash algorithms can guarantee the integrity of the distributed data even though chunks are sent to, received from, and stored by multiple, possible untrustworthy, peers.

1. INTRODUCTION

Peer-To-Peer-systems (P2P) handle large quantities of data without relying on centralized client server based scenarios [10], but instead use resources from all participating network nodes, the peers. All peers act as servers for other peers which can be used to distribute data and in turn increase the speed of the distribution process. Also the availability of the data can be increased as peers keep copies of the chunks. Common to all P2P-systems is that the resources of the all peers can be used in addition to own resources.

To use the resources of all participating peers in the network and to distribute the workload among them the data is split into multiple even-sized data blocks called chunks [8]. The chunks are then distributed among the peers, where each peer is responsible of only a fraction of the chunks. Downloading a chunk in a P2P-system is a two-step process, as first a peer storing the chunk has to be located before the download process can be initiated. As different peers are contacted for each chunk the workload is shared by all peers.

To spread the workload evenly among the peers most P2P-systems use a hash algorithm to produce unique identifiers for each chunk, and each peer is responsible for a range of hash values. To find a chunk with a give identifier the hash value is used to provide peers with routing information. Each peer forwards requests for a chunk to the peer he knows either has the chunk or is closest to the chunk ([15]). Once the final peer is found he can contact the originator of the request who's address was forwarded with the request. Information is then shared directly between those two peers.

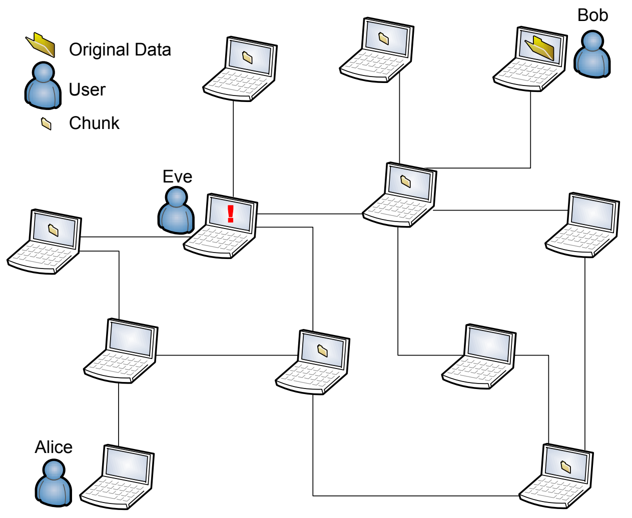

Figure 1: Basic P2P-system

A negative effect of the resource sharing in P2P-systems is that the trustworthiness of the peers can not be assured. Thus chunks are vulnerable to attacks [3] such as information disclosure (confidentiality) or tampering (integrity) attacks and routing requests can be misdirected. Therefore we either allow only trustworthy peers to join the system or

*This work was partially funded by the DFG research group 733

use other mechanisms to assure the availability, integrity, and confidentiality ([1]) of the data. Availability can be achieved by redundantly storing chunks on different peers. Encryption can ensure the confidentiality of the data, but implementation details depend largely on the application. We will focus on how secure hash algorithms protect the integrity of chunks.

To demonstrate how communication among individual peers work and how hashes are employed consider the P2P-system in figure 1 where Bob wants to share some data and Alice wants to download that data. Some of the peers already downloaded individual chunks. For Alice to acquire the data she asks Bob for a list of chunks. She then tries to find the peer responsible for the chunk using the P2P-system's algorithm. When the routing request is forwarded over Eve or the chunk downloaded from Eve all information shared is subject to manipulation unless some security mechanisms are employed to circumvent this.

Section 2 will give more details on the pattern. We will motivate the need for secure hash algorithms by giving a short example followed by the context in which to apply the pattern and forces to consider. Afterwards in section 2.6 we will show different levels of details of the pattern. First, a class diagram gives an overview over involved parts of the system, second a sequence diagrams shows the flow of information, and third implementation details are given. The reader should to have a basic understanding of P2P-systems addressing and routing mechanisms as found e.g. in [10] and of secure hash algorithms (see e.g. [13]).

2. DATA INTEGRITY IN P2P-SYSTEMS

2.1 Intent

Secure hashes can be used to ensure data integrity. In a P2P-system data is split into smaller chunks, hashed, and spread over multiple peers. Depending on the strength of the hash's algorithm tampering attacks are prevented even though the chunks are stored and handled by untrusted entities.

2.2 Example

Alice wants to retrieve a large document from Bob that he stored using a P2P-enabled software as Bob has only very limited resources. To store the document he split it into several small chunks and distributed them over the peers of the network. Alice does not know which peers store these chunks. To find them Bob sends her a message containing a list of the chunks for his document and how to address the peers storing the chunks. Alice now asks peers in the network to send the chunks that belong to Bob's document. Eve wants to spread her worm and instead of sending Alice the requested chunk she will send her worm. Alice has no way of detecting this misuse and to remove or quarantine the worm.

2.3 Context

This pattern can be applied by service providers with limited resources, who want to distribute large quantities of data to multiple users. The distribution process must be resilient against tampering attacks. All participating network nodes can communicate directly.

2.4 Problem

In a P2P-network all peers can share information with other peers. This means that, among others, the data is vulnerable to tampering attacks and can not be trusted. However, service providers need to utilize peers to decrease the workload of their own resources, but still want to ensure the integrity of the data. An approach is needed that provides integrity of the shared data without relying solely on the initial service provider resources.

2.5 Forces

For the proposed solution we have to consider the following forces:

Minimize own resource usage Large quantities of data need to be distributed with limited own resources. Instead of relying only on own resources, downloading peers help in disseminating the data by acting as servers to others.

Fast resource identification Peers store different chunks of data. If downloads happen from multiple peers simultaneously, identifying a peer that stores a specific chunk has to be done efficiently.

Data integrity Data must be secured against tampering, as chunks stored on foreign peers can be altered for different misuse scenarios.

Data Confidentiality As with tampering, the confidentiality of the data needs to be ensured.

Minimal communication with data provider The original owner of the data has only limited resources, thus the validity of a downloaded chunk has to be identified with minimal communication with the owner of the data.

2.6 Solution

It has been shown that strong cryptographic hashes ensure data integrity, and at the same time identify chunks in a P2P-network [15]. These characteristics can be used to prevent tampering attacks in P2P-networks and efficiently distribute data. To do so the initial data provider splits the data into even-sized chunks of smaller data. For each chunk the hash is computed and stored, and users requesting the data get a list with these hashes over a secure connection. Each entry in the list identifies a different chunk that needs to be retrieved. The following sections present the structure of the solution, it's dynamics with involved parties and common pitfalls during implementation.

2.6.1 Structure

The structure of the secure pattern is given by the class diagram in figure 2. The **ServiceProvider** wants to distribute some data to a large user base. The ChunkGenerator divides the data into chunks and generates a list of hashes. The list, together with the chunks, is sent to the peers participating in the system. Depending on the implementation different peers receive different chunks during this initial dissemination. Other peers use the list to request needed chunks and verify the validity of each chunk using the ChunkVerifier after a successful download. A valid chunk is then stored to either assemble the original data when all chunks are downloaded or send it to requesting peers.

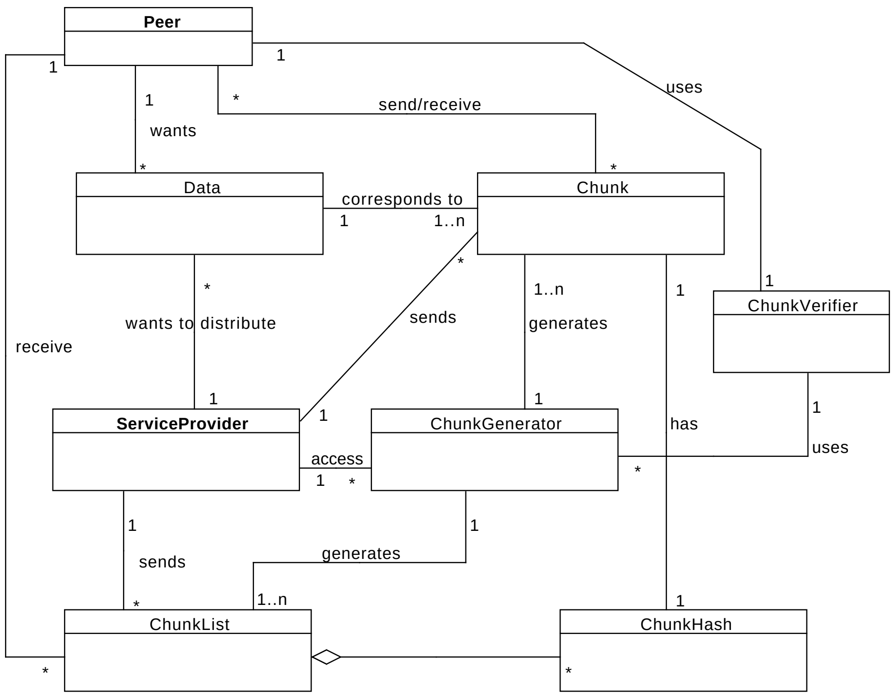

Figure 2: Class diagram for Secure Hash Addressing Pattern

2.6.2 *Dynamics*

The following two sequence diagrams, given in figure 3 and figure 4, show the dynamics for using hashes to download chunks and achieve integrity. The first diagram shows how the list of hashes is generated:

Download sequence diagram.

Summary: A service provider wants to distribute a large amount of data. The data is divided into equal-sized chunks that are disseminated to peers of the P2P-system.

Actors: The initial service provider

Precondition: None

Description:

1. The data is passed to the ChunkGenerator to be divided into equal-sized chunks.
2. The ChunkGenerator divides the data and computes the hash for each chunk.
3. The ChunkGenerator returns a list of hashes and the generated chunks.
4. The ChunkList can be accessed using a secure method that allows peers to validate the integrity of the list and the identity of the service provider.

Postcondition: The chunks are ready for distribution over the network. Clients can request the list of needed chunks.

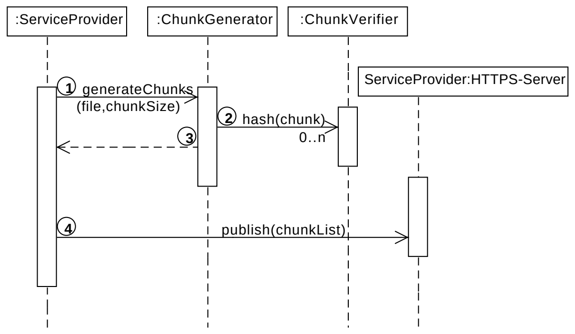

Figure 3: Sequence diagram for computing the list of hashes

Integrity check sequence diagram.

Summary: A peer wants to download the data offered by the service provider.

Actors: The downloading peer and other peers of the P2P-system.

Precondition: The peers knows how to retrieve the list of valid hashes (ChunkList) and uses a secure connection that validates the integrity of the list and the identity of the service provider.

Description:

1. The peers asks the initial service provider for a ChunkList.

2. The service provider sends the ChunkList via a secure connection.
3. For each chunk the peer nearest to the chunks' hash is requested to send it. The nearest peer has an ID that is smaller then the hash of the chunk and there is no other peer known who's ID is smaller then the chunk's hash and bigger then the ID of the contacted peer.
4. If the requested chunk is not stored by the peer contacted in step three the send request is forwarded to the peer storing it. This process is repeated until the correct peer is found or if no such peer exists the chunk is requested from the initial service provider. For details cf. section 2.7.
5. The chunk is downloaded from the peer storing it.
6. The retrieved chunk is verified.
   (a) If the computed hash and the hash stored in the list are equal the chunk is verified and gets accepted.
   (b) If the computed hash and the hash stored in the list are not equal the chunk is discarded and downloaded again from a different peer.

Postcondition The data is downloaded and verified.

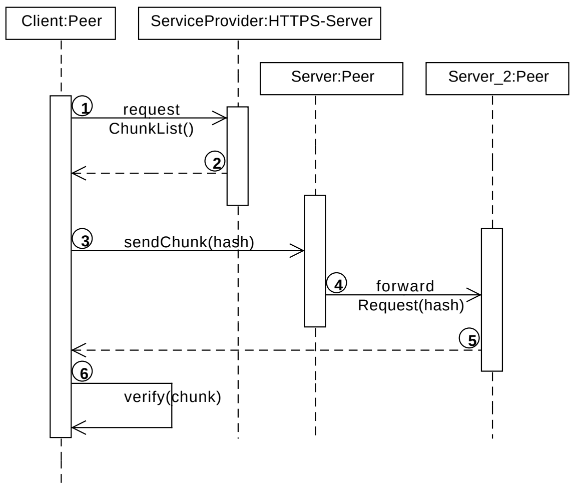

Figure 4: Sequence diagram to download a single chunk of data

2.6.3 *Implementation*

The following problems need to be considered when implementing this pattern. A common mistake is to distribute the list of hashes without securing the transmission. This leaves the list open to attacks which could be used to add malicious chunks or change a chunk's hash. To prevent this attack the download of the list has to be secured against tampering attacks, and the initial data provider's identity has to be verified using the secure pipe pattern, e.g. using SSL encrypted connections [11].

Another problem arises when the algorithm used to compute the hashes is not resilient against collision and preimage attacks [6, 16]. Attackers could perform these attacks to generate valid looking chunks that contain malicious data.

Apart from these problems the peers and the initial service provider have to use the same algorithm to generate the hashes. Only then can the peer verify the downloaded chunk.

2.7 Known Uses

Some known implementations for this pattern are, e.g. Freenet [2], CFS [4], Kadmila [5] or Chord [15]. Freenet and CFS are P2P-filesystems that use hashes to identify individual files. Kademila implementations of a Distributed Hash Table (DHT) that was adapted for and used by BitTorrent [7] to find and verify downloadable chunks. Chord also implements a DHT, and provides fast and efficient routing in the P2P-system. More variants of DHTs exist with Can [9], Pastry [12] or Tapestry [17] that have different properties when it comes to route length, number of know peers or exchanged message when a new peer joins.

All DHTs use a ring like communication structure and distribute the hash's keyspace over available peers. Peers are responsible for storing chunks that belong to their part of the keyspace and forward request to all other chunks to other peers. Therefore they always know their direct predecessor and successor and, depending on the implementation, various other peers.

2.8 Consequences

Using secure hashes in P2P-systems to verify the integrity of chunks and to address chunks has the following advantages:

- The initial service provider is only contacted when the list of valid hashes is requested or when a chunk is not stored by any peer.
- The integrity of the chunk is validated when the hash of a retrieved chunk is equal to the one stored in the list. No communication to the initial service provider is needed.
- To increase the download speed data is downloaded by requesting multiple chunks from different peers.
- The hash of a chunk is used to locate the peer storing it and to forward routing requests.
- Integrity checks and data assembly can be done by the peers without communicating with other peers.

The pattern presents some liabilities, which in part can be negated employing other patterns (see Section 2.10):

- The list of valid hashes is susceptible to different misuses, e.g. tampering, man in the middle attacks. An appropriate pattern has to be applied to ensure the list's security.
- Some algorithms for computing hashes are vulnerable to collision attacks. Using these algorithms threatens the security of the whole pattern.
- The confidentiality of the data is not protected using this pattern. If confidentiality is a concern this needs to be addressed by employing other patterns.

- If only a limited number of peers exist, the overhead of finding available peers to download the chunk from negates the advantage that arises from using multiple peers.

2.9 Example resolved

Alice wants to retrieve a large document from Bob that he stored using a P2P-enabled software as Bob has only very limited resources. To store the document he split it into several small chunks and distributed them over the peers of the network. Alice does not know which peers store these chunks. To find them Bob uses a secure connection to send her a message containing a list of the chunks for his document and how to address the peers storing the chunks. Alice now asks peers in the network to send the chunks that belong to Bob's document. Eve wants to spread her worm and instead of sending Alice the requested chunk she will send her worm. After downloading a chunk Alice computes the chunk's hash, detects Eve tampering attack and discards Eve's chunk. Alice repeats the downloading and verification steps until all chunks on the list have been retrieved and verified. She then reassembles the chunks and has the original document that Bob wanted to send her without using his resources, but ensuring the document's integrity.

2.10 Related patterns

Other patterns should be used together with this pattern. To transfer the list of chunks the secure pipe pattern [14] is needed, otherwise tampering attacks are still possible. Hashing and appropriate algorithms are described in [13]. However, recent developments in attacks against these algorithms needs to be taken into account.

2.11 Conclusion

We showed how even with limited own resources large amounts of data can be distributed to a wide user base while insuring the integrity of the disseminated data. The data is split into chunks and for each chunk a hash is computed. The initial service provider only communicates the computed hashes and individual chunks are downloaded directly from other peers. Through the hash the integrity of each downloaded chunk and thus the integrity of the original data can be verified. Additionally hashes are used for routing requests to find peers that store specific chunks, which is important when high-performing P2P-systems are designed.

3. REFERENCES

[1] M. Bishop. *Introduction to Computer Security.* Addison-Wesley Professional, 2004.

[2] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies, volume 2009 of LNCS*, pages 46–66. Springer-Verlag, 2001.

[3] F. Cornelli, E. Damiani, S. D. Capitani, S. Paraboschi, and P. Samarati. Choosing reputable servents in a p2p network. In *In Proceedings of the 11th World Wide Web Conference*, pages 376–386, 2002.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *SOSP '01: Proceedings of the eighteenth ACM symposium on Operating systems principles*, pages 202–215, New York, NY, USA, 2001. ACM.

[5] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pages 53–65, London, UK, 2002. Springer-Verlag.

[6] B. S. P. Hoffman. Attacks on cryptographic hashes in internet protocols. RFC 4270, November 2005.

[7] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips. The bittorrent p2p file-sharing system: Measurements and analysis. In *In Proceedings of Peer-to-Peer Systems IV, volume 3640 of LNCS*, pages 205–216. Springer-Verlag, 2005.

[8] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM*, 36:335–348, 1989.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 161–172, New York, NY, USA, October 2001.

[10] S. Ratnasamy, P. Francis, S. Shenker, R. Karp, and M. Handley. A scalable content-addressable network. In *In Proceedings of ACM SIGCOMM*, pages 161–172, 2001.

[11] E. Rescorla. *SSL and TLS: designing and building secure systems.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[12] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, pages 329–350, Nov. 2001.

[13] B. Schneier. *Applied cryptography (2nd ed.): protocols, algorithms, and source code in C.* John Wiley & Sons, Inc., New York, USA, 1995.

[14] C. Steel. *Applied J2ee Security Patterns: Architectural Patterns & Best Practices.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[15] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. *Computer Communication Review*, 31(4):149–160, Oct. 2001.

[16] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions md4 and ripemd. In *In Proceedings of Eurocrypt '05, volume 3494 of LNCS*, pages 1–18. Springer-Verlag, 2005.

[17] B. Y. Zhao, J. Kubiatowicz, A. D. Joseph, B. Y. Zhao, J. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical report, University of California at Berkeley, 2001.