TALLINN UNIVERSITY OF TECHNOLOGY
Department of Computer Science
TUT Center for Digital Forensics and Cyber Security

ITC70LT
Gvantsa Grigolia 144965

EVALUATION OF DATA OWNERSHIP SOLUTIONS IN REMOTE STORAGE.

Master Thesis

Supervisor: Ahto Buldas
Professor

Tallinn 2016

Autorideklaratsioon

Autorideklaratsioon on iga lõputöö kohustuslik osa, mis järgneb tiitellehele. Autorideklaratsioon esitatakse järgmise tekstina:

Olen koostanud antud töö iseseisvalt. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud. Käsolevat tööd ei ole varem esitatud kaitsmisele kusagil mujal.

Autor: [Ees− ja perenimi]

[July 9, 2016]

Annotatsioon

Annotatsioon on lõputöö kohustuslik osa, mis annab lugejale ülevaate töö eesmärkidest, olulisematest käsitletud probleemidest ning tähtsamatest tulemustest ja järeldustest. Annotatsioon on töö lühitutvustus, mis ei selgita ega põhjenda midagi, küll aga kajastab piisavalt töö sisu. Inglisekeelset annotatsiooni nimetatakse Abstract, venekeelset aga

Sõltuvalt töö põhikeelest, esitatakse töös järgmised annotatsioonid:

- kui töö põhikeel on eesti keel, siis esitatakse annotatsioon eesti keeles mahuga $\frac{1}{2}$ A4 lehekülge ja annotatsioon *Abstract* inglise keeles mahuga vähemalt 1 A4 lehekülg;
- kui töö põhikeel on inglise keel, siis esitatakse annotatsioon (Abstract) inglise keeles mahuga $\frac{1}{2}$ A4 lehekülge ja annotatsioon eesti keeles mahuga vähemalt 1 A4 lehekülg;

Annotatsiooni viimane lõik on kohustuslik ja omab järgmist sõnastust:

Lõputöö on kirjutatud [mis keeles] keeles ning sisaldab teksti [lehekülgede arv] leheküljel, [peatükkide arv] peatükki, [jooniste arv] joonist, [tabelite arv] tabelit.

Abstract

Võõrkeelse annotatsiooni koostamise ja vormistamise tingimused on esitatud eestikeelse annotatsiooni juures.

The thesis is in [language] and contains [pages] pages of text, [chapters] chapters, [figures] figures, [tables] tables.

Contents

List of Figures

List of Tables

1. Introduction

Describes the problem statement, illustrates why this is a problem and describes the contribution the thesis makes in solving this problem. Optionally, it can give a short description (1-3 sentences each) of the remaining chapters. Good introductions are concise, typically no longer than 4 pages.

2. Terms and definition

Defines the fundamental concepts your thesis builds on. Your thesis implements a new type of parser generator and uses the term non-terminal symbol a lot? Here is where you define what you mean by it. The key to this chapter is to keep it very, very short. Whenever you can, don't reinvent a description for an established concept, but reference a text book or paper instead.

3. Background and Related Work

3.1. Data Deduplication

Cloud computing is an on-demand service. Customers are charged based on used storage and bandwidth.[1] Both service providers and customers are interested in cost efficient solutions of cloud storage. Data deduplication offers disk and bandwidth savings. Idea is simple – avoid or remove a duplicated data. This section covers basic concepts of deduplication technology. Lists various methods and processing type sand underlines the approaches used in cloud storage.

3.1.1. Hash Based Deduplication

To remove or avoid duplicated data, it must be detected first. Hash based data deduplication uses the hash values of a file (or data chunk) as a file (or data chunk) identifier. Hashes of files are calculated and then are kept on the server. When the file is uploaded first time, its hash is computed and it is compared with the existing hashes on the server. If there is a match, the file is not stored on the disk (or in case of client-side deduplication, is not transfered at all). Instead, server creates the reference, which points on the already existing file, with the same hash value. If computed hash does not match with any of the hashes, the file together with the hash value is stored on the server.[1]

3.1.2. Types of Deduplication

Data deduplication differs based on processing methods. If it takes place before the client application transfers the file to the server, it is known as client-side deduplication. If it takes place, after the file is uploaded on server, it is known as server side deduplication. In client-side deduplication scenario, the client application computes the hash of the file and sends it to the server. If the hash already exists on the server side, client application does not send the

[1] "With Amazon S3, you pay only for the storage you actually use. There is no minimum fee and no setup cost. Amazon S3 has three pricing components: storage (per GB per month), data transfer in or out (per GB per month), and requests (per n thousand requests per month)." http://aws.amazon.com/s3/pricing/

file. If no match is found, it means,that the file is unique and client application transfers it to the server. On the other hand, if client application directly sends file to the server and server computes the hash after it, it is called server-side deduplication. Both processing methods save storage, but client-side deduplication also reduces bandwidth consumption.[2]

Apart of divers processing methods, data deduplication differs in processing levels. There are file and block level data deduplication. Difference between them is intuitive. In case of file level, hash of file is calculated and as a result server stores unique files. In block level scenario, files are divided into blocks(fixed or variable size). Hashes of these blocks are calculated and duplicated data on block level is avoided. [2]

The last concept is, single and cross client data deduplication. Single client data deduplication removes duplicated data in scope of one user. Duplicated data will be stored on the server, if it belongs to different users. On the other hand, cross client deduplication vanishes the user boundaries and unique data would be shared among the users.[2]

3.1.3. Summary

Cloud storage providers are looking for, the most efficient way to reduce the cost. In cross user client-side deduplication case, file or ”chunks” of file are stored only ones on the disk and users are sharing the data. It reduces the bandwidth cost dramatically, because the deduplication takes palace on client side, and duplicated files are not uploaded at all.[3] Such cost reductions is attractive for cloud storage providers, but this technology has some security drawbacks. Next section covers potential attacks taking place during cross-user client-side data deduplication in cloud storage.

3.2. Confidentiality and Privacy Issues in Remote Storage

Although deduplication is a beneficial technology, there are security drawbacks, leading to potential attacks. Taking into consideration the behavior of the cross-user client-side data deduplication, it is easy to learn some general facts. This section focuses on attacks breaching the confidentiality of data and privacy of remote storage customers, when cross-user client-side data deduplication takes place.

3.2.1. Potential Attacks

Danny Harnik was first who has demonstrated, the potential attacks in remote storage related to data deduplication technology.[4] The paper covers three cases: file detection, file content detection and covert channel. The first case shows, how trivial is to learn whether the remote server already contains the particular file or not. Attacker uploads the file and observes the network traffic or the time required to upload the file. If the file already is stored on the server, there is no need to upload it again. Client application sends only the hash of the file to the server. The observer detects, that amount of data is smaller then file's size itself(Size of the hash depends on hash function and is smaller then file size). If file is "big enough", observing the time required to upload file on server, is sufficient to learn, whether the server already contains the file or not. The law enforcement authorities, can use this behavior. Check if storage provider contains the file (e.g. file's content is against the law) and later, they can force remote storage service providers to revile the identity of the file owner.

Data deduplication technology opens the possibility to guess the content of user's data.The approach is straightforward, attacker just tries all possible variations and waits for occurrence of data deduplication. Once it takes place, attacker learns that such file (file with this content) exist on the server. The trick is that, unlike the dictionary attacks it is not detectable. It is the legitimate way to upload new documents on the server.[4] This type of attack is easy to lunch against the files with small min-entropy. To have batter understanding, refer to the following example. Bob is invited at the event in the cinema. He stores his invitation ticket in the cloud. Alice wants to learn the row and the place of Bob's ticket. She put the Bob's name on the right place and starts to brute force row and place numbers. Alice generates files with different content and uploads on cloud. Once the deduplication takes place, she will get the desired information.

Last case describes the covert channel. Precondition for this scenario is, that attacker already have to own the victims machine. In order to exchange one bit information "0" or "1", attacker generates two random files and uploads one of them. If the first file is uploaded the covert channel transfers "0" else it transfers "1" bit. Covert channel can transfers more information, by altering the number of files or the meaning of file.[4]

All above stated attacks demonstrates the side channel effects of data deduplication. Attackers exploit the vulnerability, that data deduplication is detectable. But later Halevi states that main issue is not the detectability, but using the hash value as a proxy in remote storage.[5]

He claims that, to use a hash as a proxy to retrieve the file is vulnerable. Owning a small static piece of the file(e.g. hash of the file) does not necessarily mean owning the entire file. He referees to the Dropship[2] open source project, as a brief example of misusing the storage provider. Dropship turn the remote storage provider into CDN (Content Distribution Network) service. For that time Dropbox[3] was operating based on the cross-user client-side deduplication. The users of Dropship, where able to download the file in their folder, just sending the file's hash for check to the Server. This open source project was considered as the violation of Terms of Service of the company and is not operating anymore. Halevi introduces the Proof of Ownership Protocol, which dramatically reduces the probability of the attacker to retrieve the file, without owning it. Next section covers the detail description of Proof of Ownership Protocol and other solutions offered to substitute the hash as a proxy approach for data ownership in remote storage.[5]

3.3. Summary

The amount of savings offered by data deduplication, depends on data type and content produced by users of such services.[6] In case of office workers as users in remote storage, the benefit from deduplication is high. Office workers use mostly identical template to generate the data and the portion of duplication is high. Applying data deduplication technology saves bandwidth and disk space. But same time it rises privacy and confidentiality issues. The major weakness is that, client-side deduplication is detectable and using hash as a proof of ownership is not sufficient. Anyone who possesses the hash value of file, is able to retrieve the file from the server. If the attacker obtains the hash of the file, he can retrieve the file from the server and gain unauthorized access to it.

Data privacy issue in cloud computing is one of the aspects that could break the trust of the users towards the service providers. So those who what to stay on the market, should build the systems, which takes into consideration privacy and confidentiality.

[2] https://github.com/driverdan/dropship - "Instantly transfer files between Dropbox accounts using only their hashes"

[3] https://www.dropbox.com/

4. Approach

We have demonstrated importance of data deduplication technology for remote storage services. And have determine the root cause of breaching the privacy and confidentiality. This section covers the solutions, which refuse to use the static piece of information (hash of the file) as a proxy and offers alternative ways to proof the ownership of the data. We numerate the solutions from one to seven base on published date and show how it works and what are there security and efficiency characteristics.

4.1. Solution # 1

This subsections covers Proof of Ownership (PoW) protocol, introduced by Halevi.[5]PoW involves two parties: Prover and Verifier. The goal of prover is to convince the verifier, that he "owns" particular file. While the goal of verifier is to check if the affirmation of the prover is true. To accomplish their tasks, verifier uses summary value of file, while prover relies on the file itself. Paper [5] offers three solutions, and the subsection reviews all of them, but covers security and efficiency characteristics only for the last one. Before we move to the solutions, we have to underline two constraints. First, attacker may have compliances which own the file, but the total number of bits that attacker can receive from them must be less then initial min-entropy[4] of file. And second, attacker can not interact with compliances during the proving phase.(e.g. case misusing the remote storage as CDN)

4.1.1. Setup

The most secure and less efficient from suggested three solutions, uses erasure code.[5] Form each 90% of bits, it is possible to recover the whole file. After the file is encoded using era-

[4] "The min entropy, in information theory, is the smallest of the Rényi family of entropies, corresponding to the most conservative way of measuring the unpredictability of a set of outcomes, as the negative logarithm of the probability of the most likely outcome." "A random variable X has **min-entropy** k, denoted $H_\infty(X) = k$, if $\max_x Pr[X = x] = 2^{-k}$ "[7]

[5] "The basic premise of erasure coding goes as follows: Take a file and split into k pieces and encode into n pieces. Now, any k pieces can be used to get back the file"

sure code, next step is to build the Merkle-tree[8] on the encoded file. The verifier(server) keeps the root value of the computed tree and the number of leaves. During the poof phase, verifier(server) asks the prover(client) for some number of leaves' values and their sibling paths. The verifier checks if all the provided sibling paths gives the valid Merkle-tree root value. Based on the outcome, server grants or does not grant the access to the file.

Computing erasure code requires access to the file and in case of large files (the files stored on the secondary storage) it raises communication complexity. To increase the efficiency of the protocol, erasure encoding is substituted with universal hashing[9]. First the file is hashed and then the Merkle-tree is built on the hash. The hashing servers to reduce the size of file up to some predefined number of bits(max length 64MByte). The second solution is more efficient then first one, but it weakens the security. Security requirement for first solution claims: attacker can not retrieve file from the server, if the min-entropy remained in file after attacker receives the bits form compliances, is bigger then security parameter. Erasure encoding substitution with universal hashing, made changes in security requirement as well. For second solution, security requirement stress that, attacker can convince the verifier to grant access to the file, if it receives some T bits from compliances, which can be less then min-entropy of the file. (e.g.64MByte)

Erasure code and universal hashing solutions, both considers that the input file is taken form an arbitrary distribution. On the other hand, the third solution claims that, in realistic scenarios, the attacker always has some information about file which he desires to extract.Thereof, it is resalable to relax the security requirement and define it for particular class of distribution. Such relaxation of security requirement gives possibility to modify the protocol and make it more space efficient. In particular instead of working with bit vectors, it is possible to divide file into blocks and operate over the blocks. There are three phases to prepare the input for Merkle-Tree: Initializing, reducing and mixing. First the M bit size file is divided into m blocks. In the initializing phase, l blocks of buffer and IV (Initial Vector) are allocated. Next comes reduction phase, which is a liner mapping. It maps, original file's m blocks to the allocated l buffer blocks. Each block of the file is XORed in specific number in some locations. And locations are taken from IV, which is generated as SHA256(IV[i-1],File[i]). Where i is the block number of the file and IV[0] is defined as SHA256-IV.[6]. The same operations take palace at mixing phase. But with one difference, instead of file blocks, buffer blocks are taken as input of XORing.

[6]For SHA-256, the initial hash value, H(0), consists of the eight 32-bit words, in hex. These words were obtained by taking the first thirty-two bits of the fractional parts of the square roots of the first eight prime numbers.https://tools.ietf.org/html/rfc4634#section-6.2

4.1.2. Security And Efficiency

To demonstrate the soundness of the last solution, it is better to view the file from attacker's perspective. Input file in this scenarios is not take form arbitrary distribution, but form some class of distribution. And it reasonable for real life scenarios, as attacker always know some peace of information(e.g. file format) about the file that he tries to retrieve. M-bit file with k bits of min-entropy, can be represented from attackers perspective as $\vec{f} \leftarrow \vec{w} \cdot A + \vec{b}$, where $\vec{w} \in \{0,1\}^k$ and is chosen randomly, while $A \in \{0,1\}^{k \times M}$ and $\vec{b} \in \{0,1\}^M$ are chosen by attacker(based some knowledge that attacker has). Protocol uses hash function to prepare input for Merkle-tree, which is linear mapping, $h(\vec{f}) = \vec{f} \cdot C = \vec{w} \cdot AC + \vec{b}C$.[5] Important part in this linear mapping is that the linear code that is generated by AC matrix to have a large minimum distance. And it is possible to achieve as we are choosing matrix C for mapping. The theorem #3 proved in the paper states that the last solution is the secure proof of ownership with soundness $\left(\frac{L-d+1}{L}\right)^t$ where L is reduce buffer, t is number of challenges on Markle-tree and d is the minimum distance of the linear code generated by AC matrix.("For example, if the code has minimum distance$\geqslant \frac{l}{3}$ then we get soundness of $(\frac{2}{3})^t$.")[5]

Time efficiency is one of the important features, that characterizes the protocol and influences decision weather to implement it or not. Halevi evaluates the performance of PoW protocol, and compares it with non-secure data-deduplication and whole file transfer (without data-deduplication) implementations of remote storage. Overall time protocol requires, is decomposed in three parts: Client, Server and Network time.[7] Client time is calculated as the sum of the subtasks client performs and subtasks are: reading file from the disk, computing the SHA256 hash, going through reduction and mixing phases and computing the Merkle-tree. Server time – the time server needs to check Mekle-tree authentication signature. And Network time – respectively the time necessary for data generated by prover and verifier to travel via network. Server and Network time consumption is negligible. (E.e. checking 20 sibling paths "costs" 0.6ms and data generated for transmission is less then 20K. In case of 5Mbps network the overhead is 0.1 ms. All together the overhead of Server and Network is 0.7 ms). While the main pressure comes on client side. To compare it with insecure implementation of data-deduplication, PoW on client side adds reduction and mixing phases and Markle-tree calculation. As result of the tests, reduction phase adds less then 28% time over insecure solution. Mixing phase and Mekle-tree calculation behavior depends on th size of the file. For small size files(less then 64MByte), the time up-growth is 200% , but it stays

[7] "The measurements were performed on an Intel machine with Xeon X5570 CPU running at 2.93GHz. We implemented the protocol in C++ and used the SHA256 implementation from Crypto++"

constant(1158ms) once the file grows above 64MByte. PoW is also compare with the solutions to avoid deduplication and always send a whole file to the server. Protocol is observed in two setups: network with 5Mbps and 100Mbps. The results are following: PoW always consumes less time in 5Mbps then transferring the whole file. Once the file grows over the 1GByt PoW requires 1% time of the file transfer. In case of 100Mbps network, the protocol has lower bound for file size. For files larger then 64K, PoW consumes less time then solution without deduplication . And for files larger then 1GByt, it requires 4% of time of the whole file transfer.

	Dedup Time $= T_d$	File Transfer Time $= T$
PoW	$3.28T_d + 0.7ms$[8] and $1.28T_d + 1.165s$[9]	$0.1T$[10]; $0.4T$ [11]; $> T$ [12];

Table 1. PoW Time Comparison

- Proofs of Ownership in Remote Storage Systems - 2011 +
- Boosting Efficiency and Security in Proof of Ownership for Deduplication - 2012 -
- Provable Ownership of File in De-duplication Cloud Storage - 2013
- Leakage-Resilient Client-side Deduplication of Encrypted Data in Cloud Storage - 2013
- A Secure Client Side Deduplication Scheme in Cloud Storage Environments - 2014
- A Tunable Proof of Ownership Scheme for Deduplication Using Bloom Filters - 2014
- An efficient confidentiality-preserving Proof of Ownership for Deduplication -2014

[8]For files less then 64Mb: $T_d + 0.28T_d + 2T_d + 0.7ms$
[9]For files more then 64Mb: $T_d + 0.28T_d + 2T_d + 0.7ms$
[10]In 5Mbps network and file size more then 1Gb
[11]In 100Mbps network and file size more then 1Gb
[12]In 5Mpbs for any size of file and In 100Mbps for files larger then 64K

5. Evaluation

Demonstrate why the developed framework is secure and efficient.

1. Time complexity evaluation.
2. Cost analyses.

6. Future Work

In science folklore, the merit of a research question is compounded by the number of interesting follow-up research questions it raises. So to show the merit of the problem you worked on, you list these questions here.

7. Conclusions

Short summary of the contribution and its implications. The goal is to drive home the result of your thesis. Do not repeat all the stuff you have written in other parts of the thesis in detail. Again, limit this chapter to very few pages. The shorter, the easier it is to keep consistent with the parts it summarizes.

References

[1] M. B. Babette Haeusser, Alessio Bagnaresi and A. Woodcock, *Guide to Data De-duplication:The IBM System Storage TS7650G ProtecTIER De-duplication Gateway*. IBM Redbooks, 2008, pp. 20–27.

[2] S. Srinivasan, *Security, Trust, and Regulatory Aspects of Cloud Computing in Business Environments*. USA: Information Science Reference (an imprint of IGI Globalo), 2014, pp. 77–78.

[3] M. H. T. K. M. R. U. V. Moritz Borgmann, Tobias Hahn and S. Vowe, *On the Security of Cloud Storage Services*. Germany: SIT Technical Reports, 2012, pp. 55–110.

[4] B. P. Danny Harnik and A. Shulman-Peleg, "Side channels in cloud services, the case of deduplication in cloud storage," *IEEE Security and Privacy Magazine*, vol. 8, no. 2, pp. 40–47, 2010.

[5] B. P. Shai Halevi, Danny Harnik and A. Shulman-Peleg, "Proofs of ownership in remote storage systems," *ACM Conference on Computer and Communications Security*, pp. 491–500, 2011.

[6] M. Dutch, "Understanding data deduplication ratios," pp. 5–9, 2009.

[7] L. Reyzin, "Some notions of entropy for cryptography," *Information Theoretic Security*, vol. 6673, pp. 138–142, 2011.

[8] R. C. Merkle, "A certified digital signature," *In Proceedings on Advances in cryptology*, vol. CRYPTO, no. 89, pp. 218–238, 1989.

[9] P. B. Miltersen, "Universal hashing," *Lecture note*, p. 12, 1998.

A. Appendix 1