On the Security of an Efficient Dynamic Auditing Protocol in Cloud Storage

Jianbing Ni, Yong Yu, Yi Mu, Senior Member, IEEE, and Qi Xia

Abstract—Using cloud storage, data owners can remotely store their data and enjoy the on-demand high quality cloud services without the burden of local data storage and maintenance. However, this new paradigm does trigger many security concerns. A major concern is how to ensure the integrity of the outsourced data. To address this issue, recently, a highly efficient dynamic auditing protocol (IEEE Transactions on Parallel and Distributed Systems, doi:10.1109/TPDS.2013.199) for cloud storage was proposed which enjoys many desirable features. Unfortunately, in this letter, we demonstrate that the protocol is insecure when an active adversary is involved in the cloud environment. We show that the adversary is able to arbitrarily modify the cloud data without being detected by the auditor in the auditing process. We also suggest a solution to fix the problem while preserving all the properties of the original protocol.

Index Terms—Auditing protocol, cloud storage, data integrity, security analysis

1 INTRODUCTION

Cloud storage allows data owners to remotely store their data and access them via networks at anytime and from anywhere. Despite the obvious benefits such as improved scalability and accessibility, data replication and backup of cloud storage, it also brings new security challenges to the cloud data security. Once the data are outsourced, the data owners relinquish the control over the fate of their data. The server may hide data loss accidents to maintain the reputation, or discard the data which have not been or are rarely accessed to save storage space [1]. Therefore, it is highly essential for data owners to check the integrity and availability of the cloud data.

Several novel and efficient auditing protocols such as [2], [3] have been proposed to ensure the integrity of the static data. However, static storages are far from sufficient for cloud applications. The data stored in the cloud may be frequently updated by the users, including insertion, deletion, modification, append in, reordering, etc. Recently, an efficient auditing protocol [4] was proposed to support data dynamic operations and batch auditing. This protocol, which employed the data fragment technique and homomorphic verifiable tags, is suitable for large scale cloud storage systems since it enjoys low storage overhead and communication cost.

In this letter, we revisit the dynamic auditing protocol in [4] and show that there is a security flaw when an active adversary is involved in the protocol. Specifically, an active adversary is able to arbitrarily modify the cloud data and produce a valid auditing proof to pass the auditing verification. As a result, this adversary can fool the auditor and the data owner to believe that the data are well maintained in the cloud, while the data has already been corrupted. In this attack, the information that the adversary needs to possess is how data are modified, instead of the content of the data or any data tags. We then give a solution to fix the problem without sacrificing any desirable features of the original protocol.

2 REVIEW THE PROTOCOL

The data component $M$, which is divided into $n$ blocks and each block is further split into $s$ sectors, is the file to be stored in the cloud. The data component is denoted as $M = \{m_{ij}\}_{i=1}^{n} \in \{0,1\}$ and the abstract information of $M$ is denoted as $M_{info}$. Let $G_1$, $G_2$ and $G_T$ be three multiplicative cyclic groups of prime order $p$, $g_1$ and $g_2$ be the generators of $G_1$ and $G_2$, $e : G_1 \times G_2 \rightarrow G_T$ denotes an admissible bilinear pairing and $h : \{0,1\}^n \rightarrow G_1$ is a secure hash function that maps the $M_{info}$ to a point in $G_1$. Here we briefly review their protocol which involves five phases: KeyGen, TagGen, Chall, Prove and Verify. We skip the parts of dynamic auditing and batch auditing and readers are referred to [4] for the details.

KeyGen($\lambda$) $\rightarrow$ ($pk_1$, $sk_1$, $sk_2$, $sk_3$). This algorithm takes the security parameter $\lambda$ as input and chooses two random numbers $sk_1$, $sk_2$, $sk_3$ $\in Z_p$. It outputs the public tag key as $pk_1 = g_2^{sk_1}$ in $G_2$, the secret tag key $sk_2$ and the secret hash key $sk_3$.

TagGen ($M$, $sk_1$, $sk_2$, $sk_3$) $\rightarrow$ $T$. This algorithm takes the data component $M$, the secret tag key $sk_2$ and the secret hash key $sk_3$ as inputs. It chooses $s$ random values $x_1, \ldots, x_s$ $\in Z_p$ and computes $u_i = g_1^{x_i}$ in $G_1$ for $j \in [1, s]$. Then, for each block $m_i$ ($i \in [1, n]$), it computes a tag $t_i$ as

$$t_i = \left(h(sk_3, W_i) \prod_{j=1}^{s} u_j^{m_{ij}}\right)^{sk_2},$$

where $W_i = FID[i]$, in which $FID$ is the unique identifier of the data component $M$ and $i$ denotes the block number of $m_i$. It outputs the set of data tags $T = \{t_i\}_{i \in [1, n]}$.

Chall ($M_{info}$) $\rightarrow$ $C$. This algorithm takes the abstract information $M_{info}$ as input. It selects some data blocks to construct a challenge set $Q$ and picks a random $v_i$ $\in Z_p^*$ for each $m_i$ ($i \in Q$). Then, it picks a random $r$ $\in Z_p^*$ and computes the challenge $R = (pk_1)^r$ and outputs the challenge $C = (\{i, v_i\}_{i \in Q}, R)$.

Prove ($M$, $T$, $C$) $\rightarrow$ $P$. This algorithm takes the data component $M$, the data tags $T$ and the challenge $C$ as inputs and outputs a proof $P = (TP, DP)$. The TP is generated as

$$TP = \prod_{i \in Q} t_i^{v_i}.$$ 

To generate the DP, it first computes $MP_j = \sum_{i \in Q} v_i \cdot m_{ij}$ for each $j \in [1, s]$. Then, it computes DP as

$$DP = \prod_{j=1}^{s} e(u_j, R)^{MP_j}.$$ 

Verify ($C$, $P$, $sk_3$, $pk_1$, $M_{info}$) $\rightarrow$ 0/1. This algorithm takes the challenge $C$, the proof $P$, the secret hash key $sk_3$, the public tag key $pk_1$ and the abstract information $M_{info}$ as inputs. It computes $H_{chal} = \prod_{i \in Q} h(sk_3, W_i)^{v_i}$, and then checks the following verification equation:

$$DP \cdot e(H_{chal}, pk_1) \stackrel{?}{=} e(TP, g_2^r).$$

If the equation holds, output 1; Otherwise, output 0.
3 On the Security of the Protocol

The auditing protocol described above enjoys the desirable feature of data privacy, and can be extended to support dynamic auditing and batch auditing for multiple owners and multiple clouds [4]. Regarding the security of the protocol, three kinds of attacks were considered in [4]. First, an adversary cannot choose another valid and uncorrupted pair of data block and data tag \((m_i, t_i)\) to replace a challenged pair of data block and data tag \((m_i, t_i)\), when it has already discarded \(m_i\) or \(t_i\). Second, an adversary cannot forge the data tag for a data block to deceive the auditor. Finally, an adversary cannot generate a valid proof from the previous proofs or other information, without retrieving the outsourced data. However, stronger adversaries may exist in the real cloud environment. For example, an active adversary may corrupt or alter the data at his will and also modify the protocol messages in the network in order to fool the auditor and the owner to believe that the data are well maintained by the cloud server. Such kind of an adversary can be a malicious programmer who can plant bugs in the software and network protocols running on the cloud.

Assume the adversary modifies each data sector \(m_{ij}\) to \(m'_{ij} = m_{ij} + t_{ij}\) for \(i \in [1, n], j \in [1, s]\). The information the adversary needs to record is \(t_{ij}\) (i.e., how the owner’s data are modified).

In the auditing process, the auditor and the server honestly execute the protocol. That is, in the \textit{Chall} phase, the auditor sends a challenge \(C = (\{i, m_{ij}\}, R)\) to the server. In the \textit{Prove} phase, the server computes \(TP = \prod_{i \in Q} t_i^{m_i}\), and \(DP^*\) as

\[
DP^* = \prod_{j \in Q} e(u_{ij}, R)^{\sum_{i \in Q} m_{ij} t_{ij}} = \prod_{j \in Q} e(u_{ij}, R)^{\sum_{i \in Q} m_{ij} t_{ij}}
\]

and then sends the proof \(P = (TP, DP^*)\) to the auditor. The adversary intercepts the proof \(P = (TP, DP^*)\) on the channel, and modifies \(DP^*\) to \(DP = DP^* / \prod_{j \in Q} e(u_{ij}, R)^{\sum_{i \in Q} m_{ij} t_{ij}}\). It is easy to see that by doing such a simple modification, the adversary derives a correct proof with respect to the original data blocks \(m_i (i \in Q)\). As a result, the modified proof can pass the verification in the auditing protocol. In this way, the adversary successfully fools the auditor to trust that the data in the cloud are well maintained, while the data have been corrupted.

The original auditing protocol is vulnerable to the attack from an active adversary since it does not provide authentication of the response, so we suggest employing a secure digital signature scheme to prevent the proof from being modified. Specifically, in the \textit{KeyGen} phase, the algorithm outputs additional two parameters \((sk_S, pk_S)\) as the cloud server’s secret/public key pair. In the auditing process, before sending the proof \(P = (TP, DP)\) to the auditor, the server uses its secret key \(sk_S\) to generate a signature \(\sigma\) of \(P\) and sends \((TP, DP, \sigma)\) as the response to the challenge. Upon receiving the response, the auditor first verifies the signature \(\sigma\). If it is valid, the auditor performs the \textit{Verify} phase of the original auditing protocol; otherwise, discards the response.

It is easy to verify that the fixed protocol still preserves the properties of the original protocol such as dynamic auditing and batch auditing. For the performance of the fixed protocol, it is slightly heavier in computation and communication than the original protocol, since the server needs to compute a signature \(\sigma\) and forward it to the auditor additionally, and the auditor will perform an extra signature verification. Taking DSA or its elliptic curve version [5] as an example, the server needs to perform 1 exponentiation to generate the \(\sigma\) and the auditor will run about 1.5 exponentiations to verify the validation of \(\sigma\). For the communication overhead, the signature \(\sigma\), which is about 320 bits, will be appended to the original response and sent to the auditor. The storage overhead remains unchanged compared with the original protocol.

4 Conclusion

In this letter, we revisited the dynamic and privacy-preserving auditing protocol for the cloud storage proposed in [4] and demonstrated that an active adversary can modify the auditing proof to fool the auditor and the owner that the remote cloud files are pristine, while the files have been corrupted. We also suggested a solution to remedy this weakness without losing any features of the original protocol.

Acknowledgments

The authors would like to thank the editors and the anonymous referees for their valuable comments. The second author was supported by the University of Wollongong VC Fellowship. This work was supported by the NSFC of China under Grant 61003232, the National Research Foundation for the Doctoral Program of Higher Education of China under Grant 20100185120012, the NSFC of China for International Young Scientists under Grant 61250110543, and the Fundamental Research Funds for the Central Universities under Grants ZYGX2010J066 and ZYGX2011J067.

References