Secure Distributed Storage Overview

C. C. Palmer

IBM Research, Dartmouth College

February 18, 2013

SDS Advantages

Summary of the advantages

SDS Details

efficiency of storage
high availability and transparency
integrity of storage
complete confidentiality
proactively secure

SDS Advantages

- efficient distributed storage
- highly available storage
- transparency
- integrity of storage
- complete confidentiality
- proactively secure

efficient distributed storage

The central idea of the Secure Distributed Storage (SDS) solution is to efficiently spread important information over several separate servers. Let us assume that we want to store a file in an SDS system. The SDS system does not simply send a copy of the file to each server, since that would require every server to have enough storage to hold a complete copy of the file. Instead, the file is processed in a special way that breaks up the file into pieces, with each piece being notably smaller than the original file. Each piece is then sent to a different server.

highly available storage

Because of the special process that was used to break up the file into pieces, one only has to retrieve the original pieces from just over half of the servers in order to completely reassemble the original file. It doesn't matter which pieces are retrieved, as long as at least half of them are. This allows for the other servers to have gone off-line or otherwise be unavailable. In addition, there are means for protecting against some of the servers being hacked as well

transparency

An SDS system consists, by definition, of several distinct servers. When depositing a file into an SDS system, the user can initiate the deposit of the file via any one of these servers. That is, any one of the available servers in an SDS system can act as the user's "gateway" for storing files. Similarly, when requesting a retrieval the user may direct the request to any available server of the SDS. This also eliminates the need for the user's system to contact each of the distributed servers when attempting a deposit or retrieval.

integrity of storage

When files are deposited into an SDS system, a receipt is returned to the user that declares that the file was safely distributed and stored. This gives the user a guarantee that the file was not damaged or changed between the time it left the user's control and when it was finally stored. Furthermore, this gives the user a guarantee that he will be able to retrieve the file intact when ever he needs to do so.

complete confidentiality

The SDS system also makes it possible for a user to maintain the confidentiality of a file being stored into or retrieved from the SDS. When storing the file, the user can encrypt it before sending it to one of the servers for distribution. But all the user did is simply trade one important piece of information (the file) for another (the encryption key), thus something must be done to protect the confidentiality and availability of the encryption key. The SDS system provides a secure means of storing and retrieving that key in the SDS as well. This maintains complete confidentiality of the user's file from the time it leaves the user's system on a deposit until it returns there after a retrieval. No single system in the SDS, including the system that is handling the deposit or retrieval, ever has the user's file or secret key in plaintext.

proactively secure

Finally, an SDS system can be configured to be self-protecting. That is, since any of the servers might be attacked by hackers at any time, there must be some means for reestablishing the trust of that system. If a system was unavailable, then when it becomes available once again it can send the other servers a "what did I miss?" message and they will all cooperate to bring that server up to date. Similarly, if a server has been hacked, that condition can be detected and that server is either ignored or reinitialized to bring it up to date. In addition, the SDS system is capable of changing the representation of the data which it holds, so that information "captured" by a hacker who has broken into the system is useless. This be achieved by having the system periodically shuffle the pieces of a file stored across all the servers.

efficiency of storage

The special techniques used to distribute the pieces of the user's file to the SDS systems, or servers, are driven by two factors: n, the number of servers, and t, the maximum number of servers that the SDS should be able to tolerate being unavailable or hacked. So, for a simple example, suppose there are $n = 5$ servers and we want the system to only operate if at least half of those servers are functional. Thus, we would have $t = 2$, since the loss of more than 2 servers would violate our goal.

efficiency of storage

The SDS system breaks up the user's file F into n pieces, each having a size on the order of $\frac{|F|}{n-t}$, where $|F|$ represents the size of the user's original file. Then the total amount of storage for a file F across all the n servers is on the order of $\frac{n}{n-t}\ |F|$.

Clearly, this is an improvement over the practice of replicating the file on several systems wherein the total amount of storage for a file F across all the servers is on the order of $n\ |F|$.

high availability and transparency

The technique by which the user's information is broken up into n pieces has the unique property that any $n - t$ pieces are sufficient to reconstruct the file. This way, if some of the servers are down or unavailable, the file can still be retrieved.

This high availability is also true when depositing the file. The user can submit the file to any of the servers in the SDS system. In general, the servers in an SDS system are *uniform* in that all of them are able to perform all of the operations required for proper deposit and retrieval of information. This approach is again preferable to a RAID solution wherein a single system is the gateway to the replicated copies, resulting in a single point-of-failure.

How the SDS is able to reconstruct the file from only $n - t$ pieces is best explained with a simple example. Suppose we have a minimal SDS system with three servers ($n = 3$) and we're willing to accept at most one server being unavailable or hacked and possibly hostile ($t = 1$). For simplicity, let's assume that the contents of the user's file is represented by the number 10. The user's file is spread over the three servers, S_i:

	S_1	S_2	S_3
r_1	7	0	3
r_2	1	9	0
r_3	0	6	4

Let's assume that servers S_1 and S_3 are chosen. Of the three rows of values corresponding to S_1 and S_3, only row r_1 has non-zero values in the columns for servers S_1 and S_3. Therefore, that row is selected, its values are added together, and the resulting sum is the value of the original file: $7 + 3 = 10$. As you can see, if any other pair of servers is chosen, the same process yields a similar result. It is also important to note that none of the servers alone contain enough information to completely restore the file's value.

integrity of storage

The process of depositing a file into an SDS involves several steps.

- The user initiates the process by contacting one of the servers. Call this server the gateway. In the message, the user sends the file, F, along with a signature on the file using SK_U.
- The gateway forwards this message to all of the other servers.
- Each of the other servers receive the message, check that the signature verifies correctly, and then forwards the message (again) on to all of the other servers.
- Each server that receives at least one verifiable message proceeds to store the file F. Those servers also compute a partial signature on the message they received and send that partial signature back to the original gateway server.

- Once the gateway has received at least $t + 1$ of these partial signatures, the gateway combines all of the partial signatures into a regular signature on F and the original message U and forwards that signature back to the user as a *receipt*. Once the file has been verified by the server, the actual information dispersal process begins. This includes dividing up the file into a number of pieces corresponding to the number of working servers, determining the piece that should be stored locally, and storing cryptographic hashes of *all* of the pieces.
- The user then verifies the signature on the receipt using VK_{SDS}.

As long as the user receives the receipt and can verify it with the public verification key of the SDS, the user can be confident that the file was indeed stored exactly as it was when it was submitted to the gateway. Once it is verified, the user need not retain the receipt since another means is used to authenticate the user on a retrieval request.

integrity of storage

The process of retrieving a file from an SDS also involves several steps:

- The user contacts one of the servers. Again, call this server the gateway. In the message, the user sends a signed request for the file F.
- The gateway forwards this request to all of the other servers.
- Each server verify the message with the user's VK_U. If the message verifies and the user is the rightful owner of F, the server sends its piece of the file along with the hashes of all the other servers' pieces to the gateway.
- Since some of the servers might have corrupted information or might be actively hostile, the gateway determines which hashes are to be trusted via majority.
- The gateway reconstructs F by using the remaining pieces and the information dispersal algorithm and sends F to the user.

As you can see, even in the presence of malicious break-ins, file damage, or other situations leading to a server being unavailable, as long as there are $t + 1$ proper servers still available the file can be successfully restored with provable integrity.

complete confidentiality

By confidential, we mean that the file is never readable by any combination of a minority of the servers, furthermore, even the gateway itself cannot read the file during the deposit or retrieval activities. This is accomplished via slight modifications to the above procedures. For deposit, the process is now:

- The user generates a symmetric encryption key, FK, such as a DES key, and encrypts the file F with FK to produce $\overline{F}$.
- The user then encrypts FK with the user's public encryption key EK_U to obtain $\overline{FK}$.
- The user sends both $\overline{F}$ and $\overline{FK}$ in a message to the gateway server, along with a signature on the file using SK_U.
- The gateway forward the message to all the other servers.
- Each of the other servers receive the message, check that the signature verifies correctly.
- If the signature verifies, each server still store its share of $\overline{F}$ (as described before) and then forwards the message (again) on to all of the other servers. Each server will maintain a full copy of $\overline{FK}$ since it is so small.

- A receipt for the entire message is produced as before and is sent to the user who verifies the signature on the receipt.
- Now both the user's original file and the symmetric encryption key used to encrypt it are stored in the SDS. This has the distinct advantage of the user no longer having to worry about holding onto and protecting the encryption key of the file. In fact, the user should delete the key, leaving the only copy in the SDS.

Now, using cryptographic blinding, we can describe a scheme by which an encrypted file $\overline{F}$ may be confidentially retrieved from the SDS:

- The user generates a random integer r and encrypts it with the user's public encryption key EK_U to obtain b.
- The user computes a signature on s and some information identifying which file is to be retrieved, $S(b, F\#)$, and sends that to one of the SDS systems. Call this system the gateway.
- The gateway forwards that message to all the other SDS servers who verify the signature and that the user who signed the message is indeed the owner of the file identified in the message.

- Using a technique known as partial decryption, each SDS server will compute a partial decryption of the product $b\,\overline{FK}$, where $\overline{FK}$ is the encryption of the DES key used to encrypt F.
- The file $\overline{F}$ is reconstructed as before.
- Each server sends the result of this partial decryption, $(r \cdot FK)_i$, to the gateway.
- The gateway reassembles at least $(t+1)$ of the received $(r \cdot FK)_i$ values into $r \cdot FK$ and returns that value to the user.
- The user recovers the original file key by dividing out the blinding factor: $\frac{r \cdot FK}{r} = FK$.

proactively secure

An SDS system is defined as having n servers and a maximum of t servers that the SDS should be able to tolerate being unavailable or hacked. In order to further protect the SDS, prudent users will make an estimate length of time that one of the SDS servers might be expected to operate correctly before it is broken into and corrupted by a hacker. Let's call this the server corruption time, c. As long as t of the servers are still operational, the SDS will be able to accept new deposits and perform retrievals.

Correspondingly, after a period of time ct, we would expect the maximum number of tolerable failures, t, to have occurred. From that point on, if one more server is hacked, the SDS will no longer be trusted, if it operates at all. Assuming that no more than t servers will fail over the useful lifetime of an SDS system is unrealistic.

Proactive security(Garey 1997) is a scheme that allows us to extend the lifetime of an SDS system indefinitely by providing a way to protect and recover from such attacks.

The idea is to introduce *refreshment phases* into the system. During a refreshment phase, a server that has been broken into but is no longer under the control of the hacker can be restored to its original state.

Any data that was destroyed or modified by the hacker is restored with the help of the other servers. Any secret information, such as the user's cryptographic keys, is also randomized in a special way so that any secret information the hacker obtained since the last refreshment phase will no longer be useful.

These refreshment phases are initiated on some period that is considered sufficiently shorter than the average SDS failure time $ct + t$, regardless of whether or not any break-ins have occurred.