Boosting Verifiable Data Streaming

Dominique Schröder
Saarland University

Mark Simkin
Saarland University

March 7, 2013

1 Summary

Recently, Schröder and Schröder (SS12, ACM CCS'12) addressed the following problem: how can a computationally weak client Alice with long-term storage of size $\mathcal{O}(\log N)$, who generates more data than she can store, stream its entire data to Bob, who has seemingly unlimited storage, to store a sequence of N records. In addition, the stream should be publicly verifiable, meaning that Bob can issue publicly verifiable proofs for particular records stored at particular indices in the sequence. Furthermore, Alice can efficiently update any record. This primitive was called verifiable data streaming (VDS) protocol and it was shown how to obtain an efficient VDS protocol from a chameleon authentication tree (CAT). The VDS protocol ensures that Bob cannot alter or append any records, in the sense that he cannot convince any third party, that has access to the public key, of such a modified or new record being stored in the CAT. A CAT is a binary tree of polynomial depth where the leaves can be authenticated on the fly, i.e., without re- or pre-computing the whole tree. The basic idea of their construction is a combination of hash functions and chameleon hash functions in a Merkle tree. Although this construction has many appealing properties, we could identify several weaknesses:

Number of elements: The depth of the tree has to be fixed during the setup phase and cannot be changed afterwards. In particular, this means that once the tree is full, no further elements can be added. In (SS12) it was argued that this would not be a problem, because the data structure could handle a polynomial depth and therefore an exponential number of elements. In practice however, this turns out to be an issue because one would like to keep the depth of the tree as low as possible in order to minimize the computational costs, the storage on both, the server's and the client's side. The overhead created by the proofs depends on the tree's depth and should also be reduced.

Long proofs: Since the proof size depends on the depth of the tree, even the first few elements in the tree have relatively long proofs. This is highly undesirable from a practical point of view.

Strong assumptions: A careful analysis of the security proof of shows that chameleon hash functions are not necessary for obtaining CATs. In fact, we show that a weaker primitive, namely trapdoor commitments schemes (TDC), are sufficient. This result has two major implications: (1) On the theoretical side our result shows that VDS can be obtained from one-way functions (OWF), because TDCs can be constructed from OWFs in a black-box way. (2) On the practical side we can obtain a more efficient instantiation of CATs, that reduces the clients state by 50%.

	Fully dynamic	***t*-bounded dynamic**	**Optimal depth**	**From OWF**	**ROM**
(SS12, CCS'12)	✗	✗	✗	✗	✗
Construction 1	✓	✓	✓	✗	✓
Construction 2	✗	✓	✓	✓	✗

Table 1: Comparison of the different constructions. ROM means that the security proof only holds in the random oracle model and OWF indicates whether the construction can be build/obtained from one-way functions.

Applications

VDS protocols have many natural applications such as Google drive, Dropbox, Apple's iCloud, and many more. All these applications follow the basic idea that users can outsource most of their data to a seemingly unbounded storage. Probably the most famous example is Google's Chromebook, where users store most of their data in the cloud. Each user is provided with some free online storage, which can be extended at the owners expense. VDS considers the ordering of the elements in the stream, which turns out to be a natural requirement in computer science. A further application of VDS can be found in the stock market. There, one common task is the live streaming of stock quotes, i.e. stock information such as the bid, ask price, or the volume traded of a given stock, to a central server. The server should not be able to alter or rearrange the stock quotes to influence traders, who request stock information from it.

Our Contribution

The contribution of this paper is twofold. We provide theoretical and practical improvements over the existing protocol (see Table 1 for a comparison). On the theoretical side we show that CATs can be obtained from weaker assumptions like the existence of one-way functions. On the practical side we provide two more efficient constructions. We summarize our contributions as follows:

DYNAMIC CATs: On the practical side we introduce the notion of *fully dynamic* chameleon authentication trees (f-CAT). An f-CAT is a CAT whose depth grows dynamically in the number of leaves. In particular, it can store an arbitrary amount of leaves and the trees depth is minimal with regards to the amount of leaves it currently contains. We suggest a first instantiation (Construction 1, see Table 1) that is secure in the random oracle model. Although the random oracle model has been discouraged by several works, we believe that for practical schemes it still is a reasonable model. In particular, to the best of our knowledge, no (natural) scheme whose security relies on the random oracle model has been broken so far.

The second scheme (Construction 2, Table 1) is slightly more restricted in the sense that the tree still grows dynamically in the depth, but only up to a predefined bound t. The value t can be an arbitrary polynomial but needs to be fixed during the setup. We refer to these schemes as a t-bounded dynamic CAT (t-CAT). This construction is secure in the standard model. Since the public-key is large, the second scheme is a tradeoff between efficiency and the underlying assumptions.

IMPLEMENTATION: We provide Java implementations of all schemes and compare their running times using multiple different instantiations of chameleon hash functions. Our results show a performance gain by a factor of up to 2.