Authenticated Data Structures for Graph and Geometric Searching

Ms.MEENU GUPTA, YOGESH CHAUHAN, SALMA KHAN, SANDEEP CHAUHAN

Electronicss And Communication
Dronacharya College Of Engineering

Abstract

Following in the spirit of data structure and algorithm correctness checking, authenticated data structures provide cryptographic proofs that their answers are as accurate as the author intended, even if the data structure is being maintained by a remote host. We present techniques for authenticating data structures that represent graphs and collection of geometric objects. We use a model where a data structure maintained by a trusted source is mirrored at distributed directories, with the directories answering queries made by users. When a user queries a directory, it receives a cryptographic proof in addition to the answer, where the proof contains statements signed by the source. The user veries the proof trusting only the statements signed by the source. We show how to eficiently authenticate data structures for fundamental problems on networks, such as path and connectivity queries, and on geometric objects, such as intersection and containment queries. Our work has applications to the authentication of network management systems and geographic information systems.

Introduction

In this paper we are verifying information that at first appears authentic is an often neglected task in data structure and algorithm usage.

Fortunately, there is a growing literature on correctness checking that aims to rectify this omission. Following early work on program checking and certifcation several researcher shave developed efficient schemes for checking the results of various data structures graph algorithms and geometric algorithms These schemes are directed mainly at defending the user against an inadvertent error made during implementation. In addition, these previous approaches have primarily assumed that usage is limited to a single user on an individual machine. In particular, we are interested in efficiently verifying paths and connectivity information in transportation and computer networks (that is, combinatorial graph structures), even when the network is changing. In addition, we are interested in verifying complex geometric queries in spatial data bases, such as ray shooting queries, point location queries, and range searching queries, which are used extensively in geographic. information systems.

The main challenge in providing an integrity service in such contexts is that the space of possible answers is much larger than the data size itself. For example, there are $O(n2)$ diferent paths in a tree of n nodes, and each of these paths can have $O(n)$ edges. Requiring an

authenticator to digitally sign every possible response is therefore prohibative, especially when the data is changing due to the insertion or deletion of elements in the set. Ideally, we would like our authenticator to sign just a single digest of our data structure, with that digest being built from the careful combination of cryptographic hashes of subsets of our data. If we can achieve such a scheme, then verifying the answer to a query in our data base can be reduced to the problem of collecting the appropriate partial hashes for a user to recompute the digest of the entire structure and compare that to the digest signed by the authenticator. Even when we follow this approach, however, we are faced with the challenge of how to subdivide the data in a way that allows for efficient assembly for any possible query. For simple data structures, such as dictionaries, this subdivision is fairly straightforward (say using a linear ordering and a Merkle hash tree but the subdivision method for complex structures, such as graphs, geometric structures, and structures built using the fractional cascading paradigm is far from obvious.

A Model for Authenticated Data Structures

Our data structure authentication model involves three parties: a trusted source, an untrusted directory, and a user. The *source* holds a structured collection *S* of objects, where we assume that a set of *query operations* are defined over *S*. If *S* is _xed over time, we say that it is *static*. Otherwise, we say that *S* is *dynamic* and assume that a set of *update operations* are defined that modify *S*. For example, *S* could be a network whose vertices and edges store data items and on which the following two query operations are defined: a *connectivity query* on *S* asks whether two given vertices of *S* are in the same connected component and a *path query* returns a path, if it exists, between two given vertices. We could also define update operations of *S* that add and/or remove vertices and edges. As a second example, *S* could be a collection of line segments in the plane forming a polygonal chain, where an *intersection query* returns all the segments intersected by a given query line. In this case we could de_ne update operations that insert and/or remove segments. The *directory* maintains a copy of the collection *S* together with *structure authentication information*, which consists of statements about *S* signed by the source. The *user* performs queries on *S* but instead of contacting the source directly, it queries a directory. The directory provides the user with an answer to the query together with *answer authentication information*, which yields a cryptographic proof of the answer. The answer authentication information should include a time stamp and information derived from signed statements in the structure authentication information. The user then veries the proof relying solely on the time stamp and the information derived from statements signed by the source (subject to standard cryptographic assumptions). If *S* is dynamic, the directory receives together with each update *update authentication information*, which consists of signed time-stamped statements about the update and the current state of *S*. The data structures used by the source and the directory to store collection *S*, together with the protocols and algorithms for queries, updates, and verifications executed by the various parties, form what we call an *authenticated data structure* .In a practical deployment of an authenticated data structure, there would be various

instances of geographically distributed directories. Such a distribution scheme reduces latency, allows for load balancing, and reduces the risk from denial-of-service attacks.

Security Model

We provide a security model that aims to provide eficient, secure, dynamic, and trusted solutions to the problems we consider. Our work is motivated by our desire to improve over limitations of two naïve approaches:

_ *Central Server:* Queries to collection *S* are handled by a single trusted server *ST* that gives a digitally signed response to every query.

_ *Signed Collection:* Collection *S* is signed in its entirity by a trusted party and distributed to every client. Central server solutions are prone to network delays and provide a single point of failure. Signed directories are expensive to make dynamic since upon update, the entire collection *S* needs to be signed and distributed to every client. Our approach results in solutions with the following properties:

_ Queries are handled by a collection of potentially geographically distributed directories.

_ Responses include authentication information that allows clients to verify responses with the same trust as if the response was digitally signed by a trusted third-party.

_ The size of update information sent to the directories is proportional to the number of updates, so update time is small.

_ Directory computers can be in untrusted locations and provide trusted responses. In this paper we show how to provide solutions that meet these criteria for important graph and geometric.

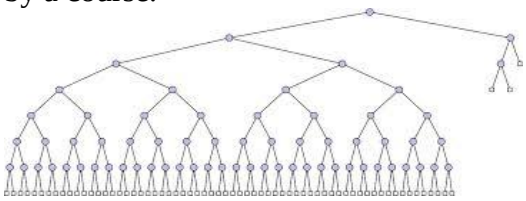
Hashing over the data structure

Let *h* is a commutative cryptographic collision-resistant hash function. We assume that a set of rules have been defined, so that *h* can operate on elements of catalogs, nodes of graph *G* and previously computed hash values. The hashing scheme can be viewed as a two level hashing structure, built using the path hash accumulator scheme: *intra-block* hashing is performed within each block defined in the data structure and *inter-block* hashing of performed through all blocks of the data structure. In the sequel, we describe each hashing structure.

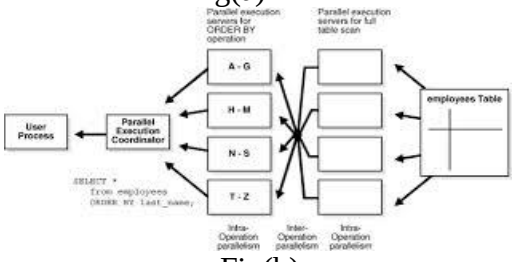
Intra-block hashing: Consider any edge (*u; v*) of *G*, i.e., *u* is one of the parents of *v*. Also, consider any two neighboring bridges (*y0; z0*) and (*y; z*) that define block *B*. Assume that *z; z0 2 Av*. We define *P* to be the sequence of elements of *B* that exist in *Av* plus the non-proper elements of the corresponding bridges that lie in *Av*. That is, *P = fp*1*; p*2*; :::; ptg*, a sequence in increasing order, where, if *z0 _ z*, *p*1 = *z0* and *pt = z*. We refer to *P* as the *hash side* of *B*. Using the path hash accumulator scheme, we compute the digest *D*(*P*) of sequence *P*. For each element *pi*, we set *N*(*pi*) = *f*proper(*pi*)*; vg* and in that way the path hash accumulator can support authenticated membership queries and authenticated path property queries. Here one such property of *P* is the corresponding node *v*. We iterate the process for all blocks defined in the data structure: for each block *B* in the data structure having a hash side *P* in *Av*, *HB* is the hash of *v* and the digest of the *D*(*P*). We also define *Bs* to be a fictitious block, the augmented catalog *As*. The hash side of *Bs* is all the block itself and in such a way the hash value *HBs* is well defined

and can be computed. All the path hash accumulator schemes used define the first level hashing structure.

Inter-block hashing: The second level hashing structure is defined through a directed acyclic graph *H* defined over blocks. That is, nodes of *H* are blocks of the data structure. Suppose that *w* is a parent of *u* and *u* is a parent of *v* in *G*. If *B* is a block of an edge (*u; v*), then we add to the set of edges of *H* all the directed edges (*B;B0*), where *B0* is a block of edge (*w; u*) that shares elements from *Au* with *B*. Additionally, if *v* is a child of the root *s* in *G*, then for all blocks *B* in edge (*s; v*) we add to the set of edges of *H* all the directed edges (*B;Bs*). The construction of *H* is now complete. *Bs* is the unique root of *H* shows the graph *H* that corresponds to a path. Each block (node) *B* of *H* is associated with a label *L*(*B*). If *B* is a leaf (sink) in *H* then *L*(*B*) = *HB*. If *B* is the parent of blocks *B*1*;B*2*; : : :;Bt* in *H*, listed in arbitrary order, then *L*(*B*) equals the path hash accumulation over *B*1*;B*2*; : : :;Bt* using *N*(*Bi*) = *fBi;HBi g*. This hashing over *H* corresponds to the second level hashing structure. Finally, we set *D*(*D*)= *L*(*Bs*) to be the digest of the whole data structure *D*, that is signed by a course.



Fig(a)



Fig(b)

Authentication information

the hash scheme that we have developed over the catalog graph *G*, a query graph *Q* and a query element *x*, we describe now what is the authentication information given to the user. Let *x* be the query element and let *v* be any node of the query graph *Q*. Let *sv* be the successor of *x* in *Cv*. In the location process, while locating *x* in the augented catalog *Av*, we _nd two consecutive elements *z* and *y* of *Av*, such that *z _ x _ y*. Elements *y* and *z* may be either proper or non-proper. They are both elements of a block *B*, such that the entrance bridge of *Av* is the higher bridge of *B*. We have that *y* is the successor of *x* in *Av* and that *sv* = *y*, if *y* is proper, or *sv* = proper(*y*), if *y* is non-proper. We call *y* and *B*, respectively, the target element and the target block of *Av*. Two useful observations are that: (1) in the location process, the traversal of the query graph *Q* is chosen so that each node of *Q* is visited once and (2) any two blocks visited by the location process (target blocks) that correspond to incident edges in *Q* share elements of the common augmented catalog, and, thus, are adjacent in graph *H*. It follows that all the target blocks define a subgraph *T* of *H*. *T* consists of the all target blocks and the edges of *H* that connect neighboring target blocks.

Lemma 1 *For any query graph Q, graph T is a tree.*

For any node *v*, let *yv* be the target element of *Av* and *Bv* the target block of *Av*. The answer authentication information will consist of:

1. *Intra-block:* for each node *v* of *Q*, the target element *yv* of *Av* and a verification sequence *pv* from *yv* up to the path hash accumulation of the hash side of *Bv*, and
2. *Inter-block:* for every node (or target block) *Bv* of *T* that is not a leaf, the

veri_cation sequences from every child of *Bv* in *T* up to the pash hash accumulation *L*(*Bv*).

Lemma 2 *If n is the total number of proper elements in the catalogs of G and d is the bounded degree of G, then for any query graph Q of k nodes, the size of the answer authentication information is* $O(\log n+k \log d) = O(\log n + k)$.

Verification of an answer

We assume that the answer given to the user is a set $A = f(av; v)$: *v* is node of *Gg*, where *av* is supposed to be the successor of *x* in *Cv*. The answer authentication information consists of two verification sequences for each node (target block) of *T* : one intra-block and one inter-block. These sequences form a hash tree in our two level hashing scheme. The verification process is basically defined by this hash tree. Intuitively, an intra-block verification sequence of a target block *Bv* provides a *local proof* that *av* is the successor of *x* in *Cv*, and then, all these local proofs are accumulated through inter-block verification sequences into the signed digest.In particular, given the elements *x*, *y*, *z*, and a node *v*, if the predicates: 1) *z* _ *x* _ *y*, 2) *y* and *z* are consecutive elements in *Av*, and 3) if *x* 6= *y* and *y* is non-proper then proper(*y*) is the next proper element of *y* in *Av*, hold simultaneously, then they constitute a proof that the successor of *x* in *Cv* is element proper(*y*). Such a proof must be given for every *v* of *Q*.Given *A*, *x* and the answer authentication information, the user _rst checks to see if there is any inconsistence between values *av* and *yv* for every *v* of *G*, i.e. if *av* 6= *yv*, if *yv* is proper, or if *av* 6= proper(*yv*) otherwise. If there is at least one inconsistence, the user rejects the answer. Otherwise, all that is needed is to verify the signed digest *D*(*D*) of the data structure. Observe, that the user possesses all the data needed for the computation of the signed digest.

Lemma 3 *If n is the total number of proper elements in the catalogs of G, then for any query graph Q of k nodes, the answer veri_cation time is* $O(\log n + k \log d) = O(\log n + k)$, *where d is the bounded degree of G.* If the digest is veri_ed, then based on the collision-resistance property of the hash function *h*, the user has a proof that the answer is correct: for each *v* of *G*, the user can verify all the three conditions previously discussed. A faulty answer can lead to a forged proof only if some collisions of *h* have been found. Thus, the security of our scheme is reduced to the collision-resistance property of *h*.

Lemma 4 *For any catalog graph G of k nodes and of total size n, both intra-block and inter-block hashing schemes can be computed in* $O(n)$ *time using* $O(n)$ *storage.*

Theorem 1 *Given a catalog graph G of bounded degree d and of total size n, the authenticated fractional cascading data structure D for G solves the authenticated iterative search problem for G with the following performance : D can be constructed in* $O(n)$ *time and uses* $O(n)$ *storage; given an element x and a graph*

query Q with k vertices, x can be located in every catalog of Q in $O(\log n + k)$ *time; and the answer authentication information has size* $O(\log n + k)$*; the answer veri_cation time is* $O(\log n + k)$.

Applications

Our authenticated fractional cascading scheme can be used to design authenticated data structures for various fundamental two-dimensional geometric search problems, where iterative search

is implicitly performed In all of these problems, the underlying catalog graph has degree bounded by a small constant. In the following, *n* denotes the problem size.

Theorem 2 *There is an authenticated data structure for answering line intersection queries on a polygon that can be constructed in O*(*n* log *n*) *time and uses O*(*n* log *n*) *storage. Denoting with k the output size, queries are answered in O*(log *n*+*k*) *time; the answer authentication information has size O*((*k*+1) log *n k*+1)*; and the answer verification time is O*((*k* + 1) log *n k*+1).

Theorem 3 *There are authenticated data structures for answering ray shooting and point location queries that can be constructed in O*(log *n*) *time and use O*(*n* log *n*) *storage. Queries are answered in O*(log *n*) *time; the answer authentication information has size O*(log *n*)*; and the answer verification time is O*(log *n*).

Theorem 4 *There are authenticated data structures for answering orthogonal range search, orthogonal point enclosure and orthogonal intersection queries that can be constructed in O*(*n* log *n*) *time and use O*(*n* log *n*) *storage. Denoting with k the output size, queries are answered in O*(log *n* + *k*) *time; the answer authentication information has size O*(log *n* + *k*)*; and the answer verification time is O*(log *n* + *k*). All these results have applications to the authentication of geographic information systems.

References

[1] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. 2001 DARPA Information Survivability Conference and Exposition*, volume 2, pages 68{82, 2001.

[2] V. King. A simpler minimum spanning tree veri_cation algorithm. In *Workshop on Algorithms and Data Structures*, pages 440{448, 1995.

[3] P. C. Kocher. On certi_cate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.

[4] G. Liotta. Low degree algorithms for computing and checking Gabriel graphs. Technical Report CS-96-
28, Center for Geometric Computing, Dept. Computer Science, Brown Univ., 1996.

[5] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. Stubblebine. A general model for authentic data publication, 2001.http://www.cs.ucdavis.edu/~devanbu/_les/model-paper.pdf.

[6] K.Mehlhorn and S. N¨aher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[7] K. Mehlhorn, S. N¨aher, M. Seel, R. Seidel, T. Schilz, S. Schirra, and C. Uhrig. Checking geometric programs or veri_cation of geometric structures. *Comput. Geom. Theory Appl.*, 12(1{2):85{103, 1999.

[8] R. C. Merkle. Protocols for public key cryptosystems. In *Proc. Symp. on Security and Privacy*, pages 122{134. IEEE Computer Society Press, 1980.

[9] R. C. Merkle. A certi_ed digital signature. In G. Brassard, editor, *Proc.*

CRYPTO '89, volume 435 of *LNCS*, pages 218{238. Springer-Verlag, 1990.

[10] M. Naor and K. Nissim. Certi_cate revocation and certi_cate update. In *Proc. 7th USENIX Security Symposium*, pages 217{228, Berkeley, 1998.

[11] W. Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668{676, 1990.

[12] J. D. Bright and G. Sullivan. Checking mergeable priority queues. In *Digest of the 24th Symposium on Fault-Tolerant Computing*, pages 144{153. IEEE Computer Society Press, 1994.

[13] J. D. Bright and G. Sullivan. On-line error monitoring for several data structures. In *Digest of the 25th Symposium on Fault-Tolerant Computing*, pages 392{401. IEEE Computer Society Press, 1995.

[14] J. D. Bright, G. Sullivan, and G. M. Masson. Checking the integrity of trees. In *Digest of the 25th Symposium on Fault-Tolerant Computing*, pages 402{411. IEEE Computer Society Press, 1995.

[15] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(3):133{162, 1986.

[16] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163{191, 1986.

[17] R. F. Cohen and R. Tamassia. Combine and conquer. *Algorithmica*, 18:342{362, 1997.

[18] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *Proc. ACM Conference on Computer and Communications Security*, 2001.

[19] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.

[20] O. Devillers, G. Liotta, F. P. Preparata, and R. Tamassia. Checking the convexity of polytopes and the planarity of subdivisions. *Comput. Geom. Theory Appl.*, 11:187{208, 1998.

[21] G. Di Battista and R. Tamassia. On-line maintenance of triconnected components with SPQR-trees. *Algorithmica*, 15:302{318, 1996.

[22] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, and M. Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33{54, 1992.

[23] U. Finkler and K. Mehlhorn. Checking priority queues. In *Proc. 10th ACM-SIAM Symp. on Discrete Algorithms*, pages S901{S902, 1999.