

Efficient Attribute Authentication with Applications to Ad Hoc Networks

Markus Jakobsson
Indiana University at Bloomington
School of Informatics
Bloomington, IN 47408
markus@indiana.edu

Susanne Wetzel
Stevens Institute of Technology
Department of Computer Science
Castle Point on Hudson
Hoboken, NJ 07030
swetzel@stevens.edu

ABSTRACT

We present a family of certification methods with applications to attribute certification, which in turn has ample applications to ad hoc networks by way of the use of centrally managed recommendation mechanisms. Our construction is based on a Merkle tree consisting of subtrees, each of which corresponds to some aspect of an attribute. We study how the ordering of these subtrees can impact the cost of representing, maintaining, and verifying attribute certificates. We describe the applicability of our construction to vehicle ad hoc networks, detail our proposed methods, and evaluate their suitability to the proposed settings.

Categories and Subject Descriptors

C.2.0 [Computer-Communications Networks]: Security and Protection; C.2.1 [Network Architecture and Design]: Wireless Communication

General Terms

Algorithms, security

Keywords

Attribute authority, certificate, hash graphs, light-weight cryptography, recommendation, spatial Merkle tree

1. INTRODUCTION

With a proliferation of wireless networks, a large number of wireless services will soon be offered to passersby, ranging from direction services, restaurant recommendations, and advertisements — to services for locating friends and users with similar interests, detect the need for equipment service, and alert users of other dangerous situations. As with any set of services that can be offered with a very low entry cost

to the market, the service quality will vary widely. Recommendation services are helpful for users to decide what service to use — whether the quality of service of the service type in question is subjectively or objectively measurable. Recommendation services are particularly useful in settings where nodes are mobile, as a node's past experience is not likely to help in making future decisions due to its constantly changing neighborhood. Recently, many researchers have studied recommendation mechanisms for peer-to-peer networks (e.g., [17]), with direct applications to ad hoc networks. Most of these contributions offer solutions in which data is distributively generated and kept, and where there is no central authority collecting and categorizing recommendation information. While this is a setting that is intellectually very stimulating, there are various security drawbacks associated with such an approach, e.g., relating to a local “hostile take-over” by a large enough (and vocal enough) group of adversaries. To enable some degree of protection against such attacks, it appears that the introduction of a central repository of recommendation information is beneficial. This is both to “average out” the impact of a local attack, and to allow partitioning of users into cliques — in terms of preferences and reliability. This central and trusted party would therefore — much like the Better Business Bureau — partition the feedback into classes. This trusted party, which we will call the *attribute authority*, would then create attribute certificates for the relevant parties. These certificates are very much like those created by certificate authorities, but instead of relating to the *identity* of the users in question, they would relate to their *behavior*.

Attribute certificates may be useful in many ad hoc settings, but particularly in those with great mobility. As mentioned, one reason is that when users are likely not to remain in the same neighborhood for long, then they are less likely to benefit from distributively generated and maintained recommendations, as they have not established trust relationships with nodes in a neighborhood they are just entering. The attribute certificates would correspond to the feedback and recommendations collected by some central authority from nodes that have interacted with the certified nodes in question. While attribute certificates may be difficult to use in applications that are extremely limited in terms of computational power, they are useful when this is not the case. For sure, this is the case for vehicular ad hoc networks where a node corresponds to a vehicle. Finally, attribute certificates are useful in settings where recommendations may be

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

VANET'04, October 1, 2004, Philadelphia, Pennsylvania, USA.
Copyright 2004 ACM 1-58113-922-5/04/0010 ...\$5.00.

of very high value to the end users, as attribute certificates are likely to be more reliable than recommendations of non-centralized recommendation systems (due to lowered risks of abuse, and increased possibilities of partitioning of users into “interest groups”). In other words, they appear to be very suitable for use in ad hoc networks established between vehicles and other nodes, whether these are stationary nodes or vehicles as well.

Example: *A service provider may measure the tire pressure in the tires of the user’s car¹ and communicate alerts to the communication system of the user’s car. While most such alerts are likely to be valid, some may not. The equipment of some service providers may be poorly maintained and tuned, and may give incorrect readings. While such service providers still operate in “good faith”, some other service providers may give incorrect information with the intention of increasing their sales — of tires, of air, or of other services when the user stops to correct the situation. Recommendation mechanisms allow us to remedy the type of problem arising from incorrect information — whether benevolent or adversarial.*

In order to make recommendations meaningful, though, it is important to re-certify attributes to keep them up-to-date and in agreement with recent feedback. Services with rapidly changing QoS will require particularly frequent updates (i.e., re-certification) of attribute certificates. Since it is likely that there will only be a handful of attribute authorities, their burden of collecting data and performing re-certification will be substantial, with the latter increasing as the time intervals of validity of the certificates decrease. *In this paper, we propose a technique for creating low-cost updates of certificates.* While the main emphasis is on limiting the computational bottleneck at the attribute authorities, our proposed solution carries the additional advantage of having very low computational costs for nodes wishing to verify the validity of attribute certificates. While this may not be a direct requirement in settings where topology changes are low, it may have an impact on nodes that move very fast, or are possible targets of Denial of Service (DoS) attacks. And of course, the less computationally demanding verification of attribute certificates is, the better.

Contributions. We present a family of solutions to the problem of providing attribute certificates with low generation costs, refreshal costs, and verification costs. Our solutions avoid the use of expensive certificate revocation lists by means of a fine-granular refreshal technique. All of our solutions are based on hash graphs. We evaluate the requirements on computation and storage that these impose on the three different types of parties we consider: the attribute authority, the parties associated with the attributes, and the mobile verifiers of the attribute certificates.

All members of our family of solutions use the same set of building blocks, and it is only the order in which these are assembled that make up the difference between the different

¹Many new tires will have Radio Frequency ID (RFID) tokens integrated in them, making pressure readings at short distances possible [12].

family members. However, this order, as we will show, has a notable impact on the costs for the various parties. We evaluate the usage costs of some of these possible solutions with respect to the different parties, and discuss what solutions are suitable for what scenarios. In particular, we propose a solution that is well-suited for VANETs.

The building blocks of our proposed solution are different hash graphs. Among these, there is the well-known hash chain, and the much-used Merkle tree structure. We also propose a variant of the latter, which we call *spatial* Merkle trees; these are particularly well suited for authentication of information arranged in a two-dimensional pattern (and can easily be extended to three or more dimensions). This makes this data structure useful to represent geographical information, and may be of independent interest. We discuss efficient methods for representing and traversing such structures, drawing on previous work on fractal traversal of standard Merkle trees.

2. RELATED WORK

Certificates. In general, standard certificates bind a public key and an identity. Certificates are issued by a trusted third party, the so-called certification authority (CA). Standard certificates consist of a data part and a signature part. The former contains an identifying string for the entity along with the corresponding public key. In addition, it may include data such as the validity period of the public key, a serial number of the certificate, etc. The signature part of the certificate contains a digital signature by the certification authority on the data part of the certificate, thus attesting to the binding of the public key to the identity of the user. Before issuing a certificate for a particular party, it is therefore crucial that the CA ensures proper identification of the requesting party. Parties receiving certificates can easily verify them by checking the correctness of the CA’s signature on the certificate (assuming that the CA’s public key is generally known and certified as well). However, in order for a recipient of a certificate to obtain sufficient assurance, he will also need to check that a certificate has not yet been revoked. For that purpose, CA’s maintain so-called certificate revocation lists (CRLs) which typically consist of a complete list of revoked certificates as well as the corresponding signature of the CA. The main drawback of using CRLs (in particular in the setting of ad hoc networks) is that they are very expensive. This is due to the fact for CRLs to be effective, they need to be transmitted frequently. Furthermore, CRLs tend to be very long.

Hash Chains. A hash chain is a sequence of values $\langle v_0, v_1 \dots, v_n \rangle$ which is defined by means of a one-way function $h : \{0, 1\}^* \rightarrow \{0, 1\}^l$, which typically is chosen as a one-way hash function, e.g., SHA-1 [4]. Starting with a secret v_0 chosen uniformly at random from $\{0, 1\}^l$, the hash chain is defined as $v_{i+1} = h(v_i)$ for $1 \leq i < n$. The correctness of value v_{i+1} can easily be checked by means of disclosing v_i and verifying $v_{i+1} = h(v_i)$.

Applying the idea of secretly building a chain in one direction and using it for public verification purposes in reverse direction, Micali introduced a light-weight certificate structure [10]: In addition to the typical information contained in a certificate, the certificates introduced by Micali comprise

The local method suffers from the fact that information is gathered locally only, the global method requires a frequent connectivity with a trusted center².

3. PARAMETERS

The type of recommendation system we consider can have a number of parameters: *time*, *approximate geographical location of access point*, *service type*, a *cluster* structure containing sets of nodes with similar QoS requirements, and finally, the *recommendation value* associated with an access point (in the context of a time period, the service type, and a given user cluster)³.

Clearly, each service provided may be associated with several access points, each one of which may both have different location and reputation. Similarly, a service provider may be associated with more than one service type. Finally, given how different users may judge the quality of service by different measures, it is also important to consider a fourth system parameter: *user cluster*. This may either be based on the application the user runs, or on subjective criteria that are hard to articulate [11, 13, 15].

Example: *A service provider, say a provider of communication bandwidth, may have multiple access points. Each one of these experience different congestion, based on the time of the day. One of the access points of our service provider may offer high bandwidth with noticeable delays, which would make it suitable for web browsing and email synchronization, but not as suitable for voice over IP (VoIP) applications. A user primarily interested in VoIP may have a poor impression of the quality of service (QoS) offered, whereas another user, interested in accessing his email, would have a much more positive opinion of the service provided to him.*

4. CERTIFICATE STRUCTURE

Components and their sizes. In the following we will consider a set of four Merkle trees. A first with leaves relating to different user clustering, a second with leaves corresponding to time, a third in which the leaves represent geographic location and a fourth one in which the leaves correspond to different access points. In addition, the Merkle tree structure will be combined with hash chains, where higher recommendation values are preimages to lower values.

Hierarchical ordering. We will put together the above described components in a way that minimizes the computational cost for generation and verification of recommendations. Let us first focus on what arrangement would minimize the verification costs for the average user. We can make the following observations:

- If a node has verified the correctness of a given leaf of a Merkle tree, then the cost of verifying the correctness of a neighbor leaf will never be higher than

²It is possible to combine both methods in a so-called hybrid system, which is expected to remedy the disadvantages of both the local and the global systems to some extent.

³The issue of how feedback is given and interpreted is orthogonal to our solution and is not discussed in this paper.

that of verifying the correctness of a more distant leaf. Most of the time, the cost of verifying a neighbor leaf will be substantially lower than that of verifying the correctness of an arbitrary leaf.

- The preferences of a user change very rarely, except in the case where the preferences depend on the application the user is running. There, a user is likely to have a low number of fixed preferences. It is likely that one can arrange users in cliques based on their preferences, given knowledge of the application in question.
- Stating the obvious, we have that time changes the same way to all users, and in a predictable way. (More generally, it is sufficient for the proposed solutions in this paper to assume a system which provides loose time synchronization.)
- Almost as obvious, we have that users move between (or stay within) geographic neighborhoods in a continuous manner, making it possible to anticipate a user's future location given his current location.

The above components can be put together in a variety of ways. We will start by describing one such way. We will later analyze the associated costs of this combination and order of building blocks, along with those of some alternatives. We will show that the combination described in the sequel is particularly well-suited for VANETs (see Sections 6 and 7).

1. We let the top-level Merkle tree be that relating to the *user clustering*. Here, the root is a value that is associated with a given attribute authority, and the leaves correspond to different clusters of user preferences; some of these are application specific. Each user would be informed by the attribute authority of what clusters he belongs to, which is determined given the feedback generated by the user. In Figure 3, we give an example of a user clustering.

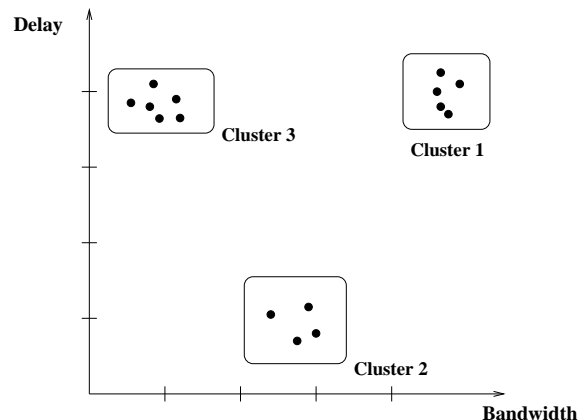


Figure 3: The clustering of these 15 users yields three clusters. Cluster 1 represents high bandwidth and high delay corresponding to downloading music. Cluster 2 with medium bandwidth and low delay is well-suited for VoIP. Cluster 3 is for sending text messages, requiring low bandwidth only, and tolerating relatively large delays.

- Let the second-level Merkle trees correspond to *time*. The root of this Merkle tree coincides with a leaf of the first Merkle tree. There will therefore be the same number of second-level Merkle trees as there are leaves of the top-level Merkle tree. The first leaf of a second-level Merkle tree corresponds to a first time interval; the second leaf to a second time interval, and so on until a last leaf, which corresponds to the last time period that can be represented by the system without performing a replacement of the Merkle trees.
- The third-level Merkle trees correspond to *geographical location*. While the predicate of being a neighbor is one-dimensional in standard Merkle trees, we propose a novel structure below, suited to map geographic locations to Merkle tree leaves in a continuous fashion. We refer to this new structure as a *spatial* Merkle tree. The root of the third-level (spatial) Merkle-trees coincide with the leaves of appropriate second-level Merkle trees, and the leaves correspond to geographic neighborhoods.
- The fourth-level Merkle tree is used to represent different access points within one geographical neighborhood. Thus, each leaf of this tree corresponds to a different access point. Here, different access points in one geographical neighborhood may often be run by different service providers, but this is not a requirement of the solution. As before, the root of this tree coincides with a leaf of the tree on the level above.
- On a fifth-level, we use a hash chain representing recommendation values. The end-point of the hash chain coincides with a leaf on the fourth-level; each consecutive preimage of the endpoint corresponds to an increasingly better recommendation.

We illustrate this organization in Figure 4. Note that other organizations of the components are also possible, and lead to different computational trade-offs between different parties; we discuss such variations in Section 7.

How are the values chosen. The values of the above tree structure are chosen as follows: The left-most nodes on level-5 are independent random numbers from a large enough interval that they cannot be guessed. The node neighboring the left-most node is computed as the hash of the left-most node value. All the values of the remaining nodes on level-5 are computed as functions of the value of the node that is their neighbor to the left⁴. The value of each level-4 leaf is a hash function of the value of the right-most level-5 node. The values of all remaining nodes of the tree are hashes of the values of all of their children. In other words, the values of all nodes (but the left-most level-5 values) are functions of at least one left-most level-5 value. The value of the root is a function of all left-most level-5 values. Note that the value of the root is constant after these values have been selected. In other words, the root value does not change when a certain set of values are disclosed.

⁴In case different size hash chains are used, the values should be computed as a function of the length of the chain as well as the neighbor to the left.

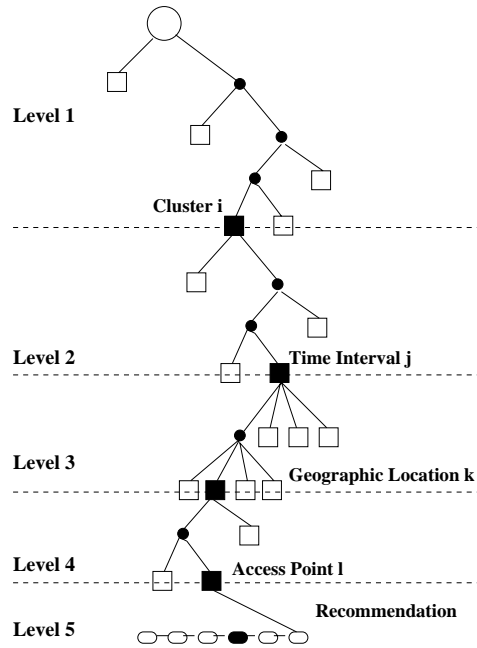


Figure 4: The root value of level-1 is publicly known. The black objects are the ones to be authenticated on each level. The authentication path consists of the white squares, which are the siblings to the current path.

Who knows what values. The root value is public, and is assumed to be known by everybody. The nodes on level-5 are all secret until disclosed — such a disclosure corresponds to the issuance of a certified recommendation. All the nodes in between are neither secret nor public in the sense that anybody is *allowed* to know these values, but nobody is *assumed* to know them. All left-most level-5 values are either known by the attribute authority, or can be generated using some pseudo-random function from some seed known only to this party. The latter is the most practical. For concreteness, the attribute authority can compute the i th left-most level-5 value as $hash(seed, i)$, where $seed$ is a random and secret value known only to the attribute authority.

Binding the identity of the AP to a recommendation. A recommendation corresponds to a node in the level-5 chain of the tree structure, with a higher (better) recommendation corresponding to the value of a node further to the left in the chain. So far, we have not discussed the binding of an identity to the recommendations. This can easily be done by indexing the hash function used for the computation of the level-4 leaves using the identity of the access point that the recommendation corresponds to. In other words, instead of letting this leaf's value be computed as $hash(n)$, where n is the value of its child, one would let it be computed as $hash(n, id)$, where id is the identity of the access point. This has to be done at setup time, when the root of the tree is computed, and corresponds to a setting in which the lowest recommendation that can be given corresponds to the disclosure of the value of the right-most level-5 node. No other hash functions in the tree would have to be indexed

in order for this binding to be implemented⁵.

4.1 Spatial Merkle trees.

Partitioning a multi-dimensional structure. We can partition any square two-dimensional structure (such as a map of some area) in four squares of the same size. Each such square, in turn, can be partitioned into four squares, and so on. This gives us a hierarchical partitioning of the map into squares of decreasing size. (Similarly, a t -dimensional structure can be partitioned into hypercubes of the same dimension.)

Mapping a multi-dimensional structure to a tree. We associate the largest square with the root of a tree. Each node n of the tree is associated with a particular square s in the two-dimensional structure, where the four children $n_{NW}, n_{NE}, n_{SW}, n_{SE}$ ⁶ of the node s correspond to the four squares s_1, \dots, s_4 that make up s (see Figure 5). Thus, a two-dimensional structure can be represented by a Merkle tree with fan-out four. (Similarly, a t -dimensional structure can be represented by a Merkle tree with fan-out 2^t .)

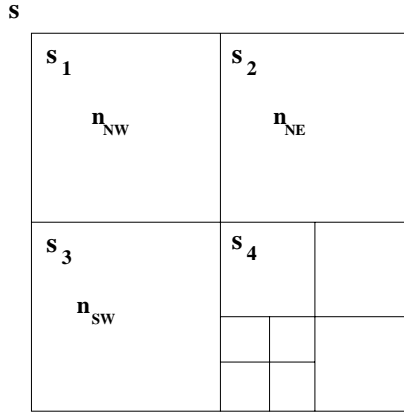


Figure 5: The square s is associated with node n which has four children $n_{NW}, n_{NE}, n_{SW}, n_{SE}$ where the latter is once again divided into four squares etc.

Using a Spatial Merkle tree. By organizing the tree as above, we have that two geographical locations in the proximity of each other will correspond to two leaves of the spatial Merkle tree, which are also in each other’s proximity. The latter means that they share a large portion of the path between individual leaves and root (at least in the common case), and thus, that the representation of one is similar to the representation of the other. This in turn will imply that the computational task of verifying the correctness of

⁵ A certificate is implicitly bound to a particular cluster, time interval and geographic location by means of the location and size of the respective trees. For example, the further left a leaf is in the second-level tree, the earlier a time interval it is. Moreover, the granularity is determined by the height of the respective tree (see also Section 5 for possible parameter choices).

⁶ NW, NE, SW, SE for northwest, northeast, southwest and southeast.

information associated with one leaf will overlap with the computational task of verifying the other; the same goes for generating the associated values. Therefore, given continuous movement of nodes, their computational requirements will be minimized when using a spatial Merkle tree representing the same number of dimensions (typically: two) in which the nodes move.

5. USE OF CERTIFICATES

What is a certificate. An attribute certificate consists of a bottom-level recommendation value and the authentication path corresponding to this node. The authentication path, as previously mentioned, consists of all the nodes that are siblings to nodes on the path from the node representing the recommendation value, and up to the root. This is depicted in Figure 4. Note that the nodes on level-5 are part of the authentication path, but only those nodes that are required in order to verify the validity of the recommendation value with respect to the known root value. When a user wishes to verify the attribute certificate of a given node, this is the information he is given. Note that such a collection of values corresponds to a certain user cluster (as determined by the branching on level-1). Therefore, if the user cluster of the verifier is not known, a set of such collection of values has to be sent over and verified.

The root of the top Merkle tree is publicly known, and all lower-level values are verified with respect to this known value. This is done by iteratively hashing the appropriate nodes until a value corresponding to the root is obtained. If this is equal to the known value, then the entire path is considered valid, along with the bottom-level recommendation value.

What does the service provider need to store. In our solution we have two service providers: The access point who provides a service the user node is interested in and the attribute authority who provides the certificates based on which the user node will decide which access point it will eventually use. While the attribute authority will store the complete information for all Merkle trees and hash chains, the access point will need to store only those authentication paths he is associated with.

Note that an access point may be given authentication paths in batches, where each batch contains several time-consecutive authentication paths from level-4 and up. It is only for the current time that the access point is given a level-5 value, though, as its recommendation values for future time intervals is not known. Later, during those time intervals, the attribute authority may send over only the appropriate level-5 recommendation value along with the node it represents.

Note further that if time were not represented on level-2, but lower down, then this would make such time-consecutive attribute certificates have a greater portion of the authentication paths in common, which would reduce the amount of information to be communicated and sent to the access point. It would also reduce the computational load of the attribute authority, as we will see later on, but would also increase the computational requirements for a typical mobile node. This suggests that there are various trade-offs between various costs; we will examine these later on.

What does the attribute authority need to store. The attribute authority selects a random and secret seed at system initialization. From this, it generates all the left-most level-5 node values, and from these, it generates the entire tree, and output the value of the root. After that, the attribute authority does, in principle, only have to store the seed, since the entire tree can be re-generated from this, and therefore, any portion can be regenerated from the seed when this portion is needed. However, this is unnecessarily costly from a computational point of view. The computational costs can be minimized by storing the *entire* tree, but this is unnecessarily expensive from the point of view of storage. To balance these two costs against each other and minimize their product, one can use techniques introduced in [7].

How is a certificate verified. Certificates are verified from the bottom up. I.e., starting with each recommendation received, the node will first work it's way up the fourth Merkle tree. Once the root of the fourth tree is verified, the node will verify the root of the fourth Merkle tree which at the same time is a leaf of the third-level Merkle tree by using the verification information for the third tree. This procedure is iterated for level-two Merkle tree. Eventually, the node will verify the root of the second Merkle tree which coincides with a leaf of the first-level Merkle tree using the public root information of the first tree using the verification information for the first tree.

For an example of this process, see Figure 4. There, the black level-5 node is verified by first applying a hash function three times to obtain the level-4 leaf marked by a black square. This value is then hashed along with the value of its sibling, represented by a nearby white square. This results in a level-3 leaf. This is repeated until a value corresponding to the root is obtained; this is compared for equality with the known root value. If these values are equal, then the level-5 recommendation value, and all other black nodes in the tree, are considered valid. Note that once a value is considered valid, future verifications sharing an authentication path may end at with a comparison of this value (if stored). This slightly reduces the computational requirements to verify time-consecutive or geographically nearby attribute certificates. In particular, it is beneficial when a verifier wishes to check the validity of several attribute certificates corresponding to different access points in the same approximate geographic location. These certificates will have in common a very large portion of the authentication path, as can be seen from Figure 4.

How is a certificate generated and updated. For each time interval, the attribute authority has to generate one attribute certificate for each access point and user cluster. It sends a set of such attribute certificates to each access point, who stores them. The attribute authority may also send attribute certificates directly to verifiers.

Studying Figure 4, it is easy to see that several access points in one geographical location will share a large portion of their authentication paths. This suggests that much of the work of the attribute authority to generate attribute certificates can be batched with respect to geographic location, with great computational savings. Similarly, the authentication paths of time-consecutive attribute certificates for a collection of access points also have a common compo-

nent. This points to the benefits of storing partial results for the next time unit. This observation is further elaborated on in [7, 14], where almost optimal amortization techniques are proposed. The same techniques, referred to as fractal traversal, can be applied to the structures we consider. In Sections 6 and 7, we will study the exact costs for a variety of related authentication structures.

Finally, we observe that the authentication paths for several subsequent time intervals may be generated and sent in batches to the certified nodes. While this does not reduce the computational costs of the attribute authority, it does reduce the communication overhead between this entity and the certified nodes (the service providers.)

How is the recommendation level modified. The hash chains representing recommendation values are constructed such that elements in the hash chain closer to the end of the chain correspond to lower values of recommendation. I.e., the value of recommendation for v_{i+1} is lower than that of v_i . Consequently, it is impossible to lower the recommendation for a particular setting before the end of the current time interval. (Instead, the attribute authority would wait until the next time period to re-issue a certificate with a lower recommendation). On the other hand, it is possible to increase recommendation during a time interval by giving a preimage to a previously issued image of the hash chain.

Once a time period ends, all attribute certificates associated with that time period expire, and have to be replaced with new attribute certificates. Since the authentication paths of two consecutive certificates may coincide to some extent, some portion of the new certificate may be equal to the old certificate. The remainder would correspond to a new path down the tree, for a new time period, and with the appropriate recommendation value disclosed. The latter involves the disclosure of a level-5 node value in the portion of the tree that the new time period (and all other parameters) correspond to.

Possible parameter choices. As indicated in Figure 2, the depicted graph allows for representation of 16 different geographic locations. In reality this is not sufficient. In order to obtain a finer granularity, and cover a large enough area, a much larger tree is needed. The example suggests an area of 0.14 square kilometers per leaf which implies the use of a complete tree of height 30 in order to evenly cover the earth. The height of the tree can be reduced by representing a smaller area. Furthermore, using incomplete trees instead allows for the application to adjust to geographic particularities, i.e., provide locally increased granularity for densely populated areas and leave those parts of the tree empty which correspond to sparse areas.

Representing a time granularity of one day requires a complete binary tree of height 9 to cover approximately one year. Just adding one additional level, allows for almost three time intervals per day.

The parameter choices for the number of clusters and access points depend directly on the application as well as the number and kind of users.

The length of the hash chain determines the number of different levels of recommendation. For example, a hash chain of length three allows to distinguish between low, medium and high recommendation. The longer the hash chain is, the more levels of recommendation can be implemented.

6. COST ANALYSIS

Attribute authority's cost. Using the fractal traversal techniques described in [7], we can amortize the computational task of generating time-consecutive sets of certificates. This technique requires a total of $2 \log N / \log \log N$ units of computation and less than $1.5 \log^2 N / \log \log N$ storage units.

Here, the value N corresponds to the number of leaves of the tree that is obtained by removing all nodes that do not belong to a tree indicating a time interval, but which are descendants to such a node. If we call the original tree T , we call the tree that results from such an operation T' . In Figure 2, this corresponds to keeping all nodes that are part of a level-1 or level-2 tree, but removing all other nodes. Note that we do not remove any nodes; the above description is only made to allow the reader to visualize what the value N corresponds to. In other words, T' is merely used to describe the costs, and not used in the actual computation.

One unit of computation corresponds to the cost of computing one node; this can be approximated by the computational effort associated with computing the associated hash function evaluations, given that these dominate the costs. It can be seen that in order to compute the value of one of the leaves of T' , one actually has to compute the entire subtree of descendants of the corresponding node of T . This is so since the nodes of this subtree will not be stored by the algorithm specified in [7]. On the other hand, the values of nodes of T that correspond to a node in T' will (under certain circumstances) be stored after being computed. Therefore, it will be less expensive to compute nodes of T that correspond to internal nodes of T' , than to compute nodes of T that correspond to leaves of T' . To simplify the analysis, however, we use the cost of computing the leaves of T' as the cost of computing any node of T' ; this clearly creates an upper bound of the actual costs. To further simplify the analysis, we assume that T is a binary tree, i.e., that the nodes of the subtree describing geographical location do not have four children. Such a structure could indeed be used, but would make the mapping less convenient.

Turning to the cost of computing a node n of T that is a leaf of T' , we see that this cost depends directly on the height $H'(n)$ of the subtree to which n is the root. This cost is approximately $2^{H'(n)}$, not taking into consideration the cost of generating the leaves of T from the random seed. This gives us an upper computational bound of $2H(n) \log N / \log \log N$ per node of T' . We will ignore the exact storage costs in this analysis, noting that it will be very close to that in [7], using the value N to denote the leaves of the tree T (as opposed to T').

We note now that the above operation has to be performed once for each valid branch of all subtrees above the subtrees describing time intervals. Here, we say that a branch is valid if and only if it is used for at least one attribute certificate using the time interval in question. Turning to Figure 2, we see that this corresponds to the number of leaves L of the tree of level-1. Refer to the height of T as H_T , and the height of T' as $H_{T'}$. We see that $H(n)$ for a leaf n of T' equals $H_T - H_{T'}$. More generally, therefore, we have that $L = 2^{H_{T'}} / \tau$. Here, τ is the number of leaves of a subtree describing time intervals, or on other words, the number of time intervals for which our proposed system can be used after being setup. Remember now that $N = 2^{H_T}$.

Thus, we see that the computational task per time interval is upper bounded by $2^{H_{T'}} \times 2(H_T - H_{T'}) H_{T'} / (\tau \log \log H_{T'})$. It is clear that the impact of the height of T' dominates this cost, suggesting that it is beneficial to let the subtrees indicating time interval be close to the root of T for the sake of reducing computational complexity for the attribute authority.

Verifier's cost. The cost of verifying an attribute corresponds to the size of the authentication path that needs to be taken, i.e., which is included in a certificate and which has not previously been taken. (This assumes that the verifier stores appropriate nodes of already verified authentication paths, which is easy to schedule how to do.) This means that the more similar two consecutive authentication paths are, the better. What changes between two verifications? Time could tick up one (or more) steps; the location could change; the cluster may change (although it is not as likely). If we elect to model aspects of the attribute certificate (such as time, or cluster) in a way that less frequently changing aspects are kept close to the root of T , that will therefore minimize the computational cost for the verifier.

Let us now consider a few examples of variations of the description given in Figure 2.

7. VARIATIONS

Keeping this description brief, we see that if the aspect of the tree corresponding to time is far from the root, then this will cause a drastic increase of the computational costs of the attribute authority, while keeping time and other rapidly changing aspects far from the root will minimize the cost for the verifier. The tradeoff is far from fair, though, as the attribute authority experiences an exponential increase (or decrease) of costs in terms of the distance of the time-subtree from the root, while the increase/decrease is only linear for the verifier.

If we consider a verifier that is unlikely to move fast, then it is better for this node to have geographical distance on top of the time in the tree than the other way around. Rapidly moving nodes, on the other hand, will benefit from time being modeled above location in the tree. This also is beneficial to the attribute authority, as we have seen.

Verifiers with rapidly changing clusters may benefit from the clustering being modeled far from the root; however, if the verifier switches between a small set of clusters, avoiding a majority, then this is not of high importance. The latter is the typical case for a user running a small set of applications, each one of which corresponds to a particular cluster. Therefore, and as we suggested in Figure 2, clustering may be high up in the tree.

If the verifiers have acutely low computational ability, static preferences, and does not move, then time should be on the very bottom. On the other hand, mobile verifiers with reasonable computational ability can afford time being high up. This supports the use of the structure we have proposed for VANETs, and supports the approximate ordering of aspects, as given in Figure 2.

8. SECURITY CLAIMS

The purpose of the system is for an inquiring node to obtain a certificate for the recommendation of an access point (at a particular location at some point in time with respect

to a particular service) in order to eventually decide which access point to use for a particular service. As explained earlier, the individual access points store the certificates they are associated with. It is in the interest of the access point to have as high a recommendation as possible in order to get as much business as possible. In turn, for the inquiring node it is crucial to obtain the correct recommendation in order to assure that he will be able to choose the service provider offering the best service for his needs.

Consequently, the system is considered to be secure if an unauthorized improvement of the recommendation is impossible. By construction, elements in the chain closer to the end of the hash chain correspond to lower values of recommendation. Thus, improving the recommendation requires the knowledge of an element in the hash chain that is closer to the beginning of the chain. Since the hash chain was generated as $v_{i+1} = h(v_i)$ ($0 \leq i < n$), an unauthorized increase of the recommendation requires inverting the hash function h . However, h is a candidate one-way hash function such as SHA-1 [4], i.e., it is easy to compute $h(n)$ for all n , but computationally infeasible for a given n' to find any n such that $n' = h(n)$. Thus, the system is computationally secure as an unauthorized improvement of the recommendation is computationally infeasible.

9. REFERENCES

- [1] M. BLAZE, J. FEIGENBAUM, AND J. LACY. Decentralized Trust Management. *Proceedings of IEEE Symposium on Security and Privacy*, 1996.
- [2] S. BUCHEGGER AND J.Y. LE BOUDEC. The Effect of Rumor Spreading in Reputation Systems for Mobile Ad Hoc Networks. *Proceedings of Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks (WiOpt03)*, 2003.
- [3] D. COPPERSMITH AND M. JAKOBSSON. Almost Optimal Hash Sequence Traversal. *Proceedings of Financial Cryptography*, 2002.
- [4] FIPS PUB 180-1, Secure Hash Standard, SHA-1. <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.
- [5] M. GOODRICH, R. TAMASSIA, N. TRIANOPOULOS, AND R. COHEN. Authenticated Data Structures for Graph and Geometric Searching. *Proceedings of CT-RSA*, 2003.
- [6] M. GOODRICH, M. SHIN, R. TAMASSIA, AND W. WINSBOROUGH. Authenticated Dictionaries for Fresh Attribute Credentials. *Proceedings of iTrust*, 2003.
- [7] M. JAKOBSSON, T. LEIGHTON, S. MICALI, AND M. SZYDLO. Fractal Merkle Tree Representation and Traversal. *Proceedings of CT-RSA*, 2003.
- [8] Y. LIU AND Y.R. YANG. Reputation Propagation and Agreement in Mobile Ad Hoc Networks. *Proceedings of IEEE Wireless Communications and Networking Conference (WCNC)*, 2003.
- [9] C. MARTEL, G. NUCKOLLS, P. DEVANBU, M. GERTZ, A. KWONG, AND S. STUBBLEBINE. A General Model for Authenticated Data Structure. *Technical Report CSE-2001*, 2001.
- [10] S. MICALI. Efficient Certificate Revocation. *Proceedings of RSA '97 and U.S. Patent 5,666,416*.
- [11] M. PAZZANI. A Framework for Collaborative, Content-Based and Demographic Filtering. *Artificial Intelligence Review*, 1999.
- [12] <http://www.rfidjournal.com/article/articleview/269/1/1/>.
- [13] P. RESNICK, N. IACOVOU, M. SUCHAK, P. BERGSTROM, AND J. RIEDL. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. *Proceedings of ACM 1994 Conference on Computer Supported Cooperative Work*, 1994.
- [14] Y. SELLA. On the Computation-Storage Trade-Offs of Hash Chain Traversal. *Proceedings of Financial Cryptography*, 2003.
- [15] U. SHARDANAND. Social Information Filtering: Algorithms for Automating Word of Mouth. *Proceedings of ACM CHI'95 Conference on Human Factors in Computing Systems*, 1995.
- [16] M. SZYDLO. Merkle Tree Traversal in Log Space and Time. *Proceedings of Eurocrypt*, 2004.
- [17] A. TWIGG. A Subjective Approach to Routing in P2P and Ad Hoc Networks, *Proceedings of iTrust 2003*, 2003.