Secure, Selective Group Broadcast in Vehicular Networks using Dynamic Attribute Based Encryption

Nanxi Chen and Mario Gerla
Department of Computer Science
University of California, Los Angeles
chnx, gerla@cs.ucla.edu

Dijiang Huang
Department of Computer Science and Engineering
Arizona State University
dijiang@asu.edu

Xiaoyan Hong
Department of Computer Science
University of Alabama
hxy@cs.ua.edu

***Abstract*—Ciphertext-policy attribute based encryption (CP-ABE) provides an encrypted access control mechanism for broadcasting messages. Basically, a sender encrypts a message with an access control policy tree which is logically composed of attributes; receivers are able to decrypt the message when their attributes satisfy the policy tree. A user's attributes stand for the properties that he current has. It is required for a user to keep his attributes up-to-date. However, this is not easy in CP-ABE because whenever one attribute changes, the entire private key, which is based on all the attributes, must be changed. In this paper, we introduce fading function, which renders attributes "dynamic" and allows us to update each one of them separately. We study how choosing fading rate for fading function affects the efficiency and security. We also compare our design with CP-ABE and find our scheme performs significantly better under certain circumstance.**

I. Introduction

Cautious landlords replace the house locks after tenants leave because they worry that tenants might keep copies of the keys. The same concept applies to protecting confidential information. Whenever a user leaves a communication group that has been exchanging and sharing confidential information, the remaining group members will replace the key used to encrypt the messages with a new one. However, given the high cost of key redistribution, this can impact performance especially when the group is made of thousands of users and the group members are likely to move in and out frequently.

Sahai et al. [2], [3]'s recent Attribute Based Encryption (ABE) scheme makes it possible to dynamically reassign group keys when requirements and conditions change. To introduce the concept of ABE, consider the following example: There are often several restrictions to redeem a coupon, say, California resident, UC or CSU students, plus AAA or UHaul membership, etc.. One must show resident ID, student ID and AAA or UHaul ID etc. to get the coupon. In the ABE context, the coupon is the object or information that we must protect, and the IDs are so-called attributes. The secret message (the coupon) is encrypted with an access control policy tree that contains the logical combination of the different attributes. The policy tree for the above coupon example would be "*CA resident* ***AND*** *(UC student* ***OR*** *CSU student)* ***AND*** *(AAA membership* ***OR*** *UHaul member)*". Each qualified user can apply and obtain a private key from certifying authority (Key Master). The key is associated with the various qualifications (i.e., attributes) of the applicant. The users can decrypt only if the attributes satisfy the policy tree.

Attributes can be expanded to represent all kinds of properties related to applicants, e.g., skin color, car brand, size, occupation and time window when these properties are valid, etc.. A policy tree defines a target multicast group to which a secret must be delivered - for example, a group key to be used for future communications. ABE saves the trouble to issue a group key in advance to each foreseen multicast group (thus avoiding combinatorial explosion). Or, conversely, it avoids the problem (and latency) of finding and certifying all the qualified members on the spot whenever the need arises. ABE requires the customers to pre-qualify (off line) for the attributes that may correspond to multicast groups they will be asked to join. Thus, the work is done ahead of time; and, it does not require combinatorial complexity.

To prevent users holding certain attributes forever, ABE adds expiration timers to revoke private keys. The problem with this revocation scheme is that the entire private key expires after a period of time. It works well in some scenarios but significantly reduce the performance in the applications such as Situation Aware Trust, proposed by Xiaoyan Hong et al. in [5], where attributes tend to change frequently. In SAT, locations are also encoded into attributes. Considering that the location attribute can be as specific as a street or a neighborhood, a mobile user's attribute is expected to change in a matter of less than one minute. Each time the location attribute changes, the entire private key which may be associated with hundreds of attributes must be changed. This is not efficient since the cost of generating new private key is proportional to the number of attributes associated with that private key [1]. If there are 100 attributes associated with a private key and even only one changes, the authority must generate a new private key with 100 attributes at considerable expense of CPU resources. In fact, the bigger the key, the longer transmission time - not a welcome proposition in applications like vehicular networks with short road-side unit and vehicle contacts.

To save CPU resources, bandwidth and time, we avoid updating those attributes that stay unchanged. To achieve this, we introduce in this paper the concept of attribute fading function, making attributes "independent" and "dynamic". With fading function, an attribute associated with a private key has its own expiration time. When an underlying property changes, the user requests a new attribute from the authority to represent his new property and the out-of-date attribute expires after a certain period of time. By this mean, a user can update partial attributes, rather than all of them, in one update. Our simulation results show that this approach significantly reduces the overhead comparing with traditional ABE especially when there are a number of "dynamic" attributes associated with users' private keys.

The rest of paper is organized as follows: Section II gives the essential background on ABE and SAT. Section III identifies the potential inefficiencies of the current ABE system. Section IV introduces the design of "dynamic" Attribute Based Encryption. Section V describes simulation experiments and results. Section VI concludes the paper.

II. Background

A. Attribute Based Encryption

Sahai and Waters et al. [2], [3] introduced Attribute Based Encryption (ABE) as a new mechanism for encrypted access control. There are several versions of ABE; the one discussed in this paper is so-called Cipher-Policy Attribute Based Encryption (CP-ABE) [1].

CP-ABE utilizes identity based encryption [7], [8] and threshold secret sharing scheme [6]. To some extent, CP-ABE is an extension of conventional PKI to groups: An authority generates public and private keys. Public key is for encryption while users keep their own private keys to decrypt. In CP-ABE, public and private keys are a not one-to-one pair, instead, there is only one public key and a potentially large number of private keys, one for each user in the target group.

A user's private key is associated with an arbitrary number of attributes. One attribute corresponds to one property. Properties such as name, ages and employers are different from person to person, so users have different attributes associated with their private keys. The publisher uses a policy tree, i.e., the logical combination of various attributes, and the public key to encrypt a message. Only clients with those attributes associated with their private keys satisfy the policy tree can decrypt the message.

In Fig. 1, the policy tree is logically composed of five different attributes. Two users try to decrypt. Kevin has attributes CA resident, UC student and AAA member so that he can decrypt the cipher while Sarah cannot.

B. Situation Aware Trust

Xiaoyan Hong et al. developed Situation Aware Trust (SAT) to provide adaptive and proactive security in mobile scenarios like VANET. Attributes in SAT identify a group of entities (e.g., taxis associated with a company, police cars in a city), a type of events (e.g., accidents, congestions), or the property of

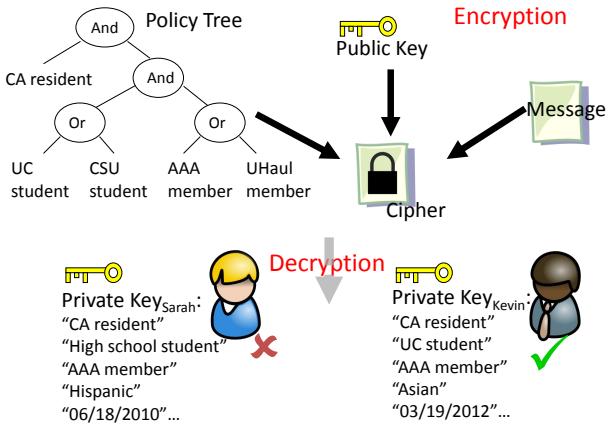


Fig. 1. An example of Policy Tree and Private Keys associated with attributes

events (location-based services, road traffic updates). They can be further classified as dynamic attributes and static attributes, depending on whether the attributes change frequently or not [5].

In SAT, vehicles fulfill a set of attributes form a "policy group". A policy group is specified by the information source and is organized automatically without relying on a trust party to manage the group [5]. For example, a company A's taxi driver can broadcast a message encrypted with policy tree "*company A **AND** Westwood Blvd. **AND** 10-11am*" to tell his colleagues that conventioneers will be waiting for pick up at Westwood Blvd. Hotels. The message may be broadcast to a very large area of the city. However, only company A's taxicabs that have been "certified" to be near Westwood Blvd. in the time window 10-11am can decrypt the message. Thus competitive advantage is ensured. At the same time, company A's taxi drivers likely to work the morning shift in remote areas will not be notified and will not waste time and gas unnecessarily.

III. Motivation

As we have learned in the previous section, attributes are the most vital concept in ABE. Let us briefly review the relationship between properties and corresponding attributes:

- Attributes associated with a user's private key reflect user's CURRENT state or properties.
- A user is not expected to hold an attribute (corresponding to specific state or property) for which he is no longer (or not yet) certified.

A key revocation mechanism is necessary to prevent a user from keeping a private key with expired attributes. In [4], the authors suggest adding expiration timestamp attributes into a user's private key when it is issued. Access control policy tree is also expected to include a subtree which defines the range of acceptable expiration times [1]. When one's private key expires, a new key must be required from the authority.

CP-ABE is also protected against collusion attacks. A user cannot give his attributes to others, neither can he borrow attributes from his expired private keys. From the mathematical construction of ABE (which is attached as an appendix), one finds that this is achieved by using a random number when generating private keys. Each private key uses a unique random number. All the attributes associated with the same private key are bonded by this random number.

The above two features ensures the security of CP-ABE, but also bring unexpected problems. Since no attribute level operation is allowed in CP-ABE, the only way to update an attribute (when the underlying property changes) is to update the entire private key. However, this may spell disaster in some cases. SAT is a case in point. As we have mentioned in the previous section, locations are encoded into attributes in SAT. Location attributes tend to change very frequently since cars are moving around all the time. That requires a moving car to update his private key time to time. On the other hand, attributes such as the color of cars, car owners and so on seldom change regardless of car speed. In conventional CP-ABE, each time the location attribute changes, these attributes must also be updated since the entire private key must be reissued.

This is equivalent of a landlord to change all the locks/keys of the entire apartment building whenever a single tenant moves out, which would be ridiculously expensive. By the same token, we would like to update only the attribute that changes rather than updating the entire private key.

IV. Dynamic Attributes

A. Key Insight

Here is another story from our acquaintance, landlord. The landlord owns a house with 5 rooms. At first, he advertises the house for rent on the Craigslist. 5 people (they are friends to each other) come together and rent the house. They sign a one-year contract and each one has a copy. However, a month later, one tenant leaves and another friend of theirs moves in. The landlord renews the house renting contract with them to incorporate the new tenant's name. Unfortunately, this group of people really like changing their home and several people move in and out in the following several months. Every time when a new tenant comes in, the landlord has to meet all the tenants and update their contracts. The tiring moving-in-and-out stuff finally drives the landlord crazy. He decides to adapt a new strategy - make the rooms for rent and sign room renting contract with each tenant. In this way, he gets rid of the trouble of meeting all the tenants when new tenants move in.

This life case inspires us. Why do we need a separate expiration timestamp attribute (a house renting contract) for the private key (house)? Why not make each attribute (room) have its own expiration time (room renting contract)? If that works, whenever we need to update an attribute, we make the old attribute expired, and then request a new attribute from authority and add it into private key.

The second step (i.e., adding a new attribute to the user's private key) is so not difficult. But it is a challenge to remove (or revoke) an attribute that is already associated with a private key. This is because the party encrypting the message does not obtain the receiver's certificate (attribute) online, and he is not able to check whether it is expired or not [1]. We do not want to assume that there is a tamper-proof "hardware", so no one can force a user to drop his old attributes. The only way to make an old attribute useless is never to post it again in policy tree (when it is expired). Our scheme is designed under this direction and we will focus on how to expire an attribute in the following sections.

B. Design

Key Master is the authority in our system. It is responsible to formalize and maintain the set of attributes that can be used in policy trees. It publishes all the attribute values, that can be used to encrypt a message, and the corresponding descriptions. Key Master is also responsible to generate keys, including public and private keys, as well as the attributes associated with private keys. Usually, for a given application, there is only one public key. Key Master generates private keys and attributes for users according to their requests. Key Master verifies each request for compliance before issuing the key to the user.
Sender is the individual that encrypts a message and sends it to others. Before encrypting a message, the sender looks up the desired attributes in the Key Master's attribute list and downloads the corresponding attribute values. He downloads the public key as well. Then he encrypts a message as instructed in Section II-A and sends out the cipher.
Receiver receives ciphers and tries to decrypt. A receiver can get a private key before or after receiving ciphers. Generally, to enable "proactive trust" [5], a receiver is supposed to obtain a private key at the start of the day (or mission) and keep his private key up-to-date proactively.
Lifetime of an attribute is a fixed time window during which an attribute's value remains *unchanged*.

We assume that all the parties in the system, including Key Master, Senders and Receivers, have coarsely synchronized time clocks using, for example, GPS or equivalent techniques.

In the following parts, we give a high level description about the design. Readers can refer to the appendix for the underlying mathematical constructions.

1) Setup: To make each attribute have an expiration time, we introduce a *fading function* for each attribute. The elapsed time is the input of the fading function. A fading function can be of any form as long as it outputs different values for different inputs. Key Master assigns the fading function to each attribute and publishes fading functions together with corresponding attribute descriptions and attribute values. To be clear, we call the published attribute value the "initial attribute value" in the following sections.

2) Encryption: The objective of the fading function is to produce a time changing attribute. When a sender wants to

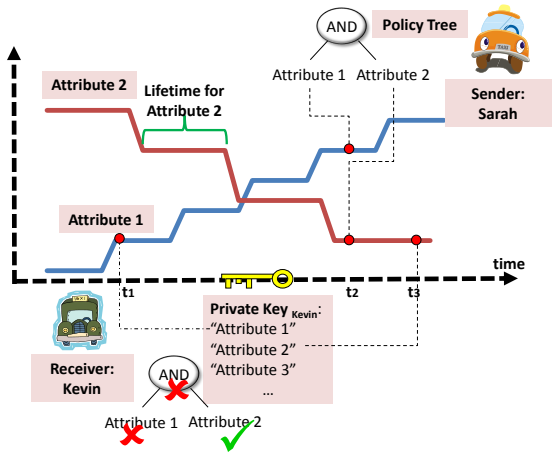


Fig. 2. Dynamic Attribute

encrypt, he downloads the initial attribute values and corresponding fading functions at first. From fading function he computes the "latest attribute value", taking initial attribute values and current time as inputs. The sender then uses the latest attribute values to compose a "policy tree". Finally, a message is encrypted with the policy tree and broadcast to all the users.

3) Key Generation: Key Master creates a record (a profile) in its memory for each new coming receiver. Records are distinct from receiver to receiver. In addition, Key Master keeps this information secret and no receiver will ever know his or others' record. Key Master utilizes the unique record to generate private keys and attributes for receivers. Because of the uniqueness of the record, receivers cannot exchange their attributes with each other (preventing collusion).

4) Decryption: A receiver requests a private key from Key Master at the beginning of the day/mission, and then keeps his attributes associated with his private key up-to-date. Only if the receiver's attributes satisfy the policy tree with which a message is encrypted, the receiver can decrypt the message. Attribute A matches attribute B if and only if they have the same attribute value.

5) Attribute Update: Each time a receiver's underlying property changes, he requests an attribute update. After Key Master receives the request, it computes the latest attribute value first using the appropriate fading function. Then it looks up the receiver's record and uses it to generate an attribute based on the latest attribute value. This new generated attribute is distributed to the receiver who requests. The receiver adds the new attribute to his private key. This attribute is only for that receiver because it is built on his record. And this "magic" record also guarantees that the newly generated attribute has no problem of associating with that user's private key because all the attributes associated with his private key uses the same record.

Update is not finished until the old attribute expires. Expiration happens when the attribute value that a sender encrypts with is different from the value based on which a receiver's old attribute is generated. Fig. 2 explains this process. A sender encrypts a message with two attributes at time t_2. The receiver has both attributes. He obtains from Key Master attribute 1 at time t_1 and attribute 2 at time t_3. Between t_2 and t_3, attribute 2's value stays unchanged, which means Key Master uses the same value as the sender does, and receiver can match the sender on attribute 2. However, from t_1 to t_2, attribute 1's value has changed and the receiver has failed to update attribute 1 value. So at t_2, the receiver's attribute 1 value at does not match the sender's, resulting in his failure to decrypt the message.

An attribute's value changes by time in our design, so we name it with "Dynamic Attribute". Dynamic Attribute Based Encryption, DABE for short, is the name of our proposed scheme.

C. Discussion

1) Fading Rate: Attribute values do not, and should not continuously change. Ideally an attribute value will be stable for a while and then suddenly jump to another value, then stay at that point for the another fixed time, jump to a third value, stay stable and go on. The fixed time span that an attribute value stay unchanged is called Lifetime for that attribute. As showed in Fig. 2 Lifetime controls the fading speed of an attribute, that is how fast an attribute becomes invalid. If Lifetime is too small, an attribute in the private key set expires very soon after it is issued so that receiver needs to update it frequently. Too large Lifetime is not desirable either. "Fading" is very important for security consideration: a receiver can use the unexpired attribute to decrypt even after he loses the corresponding property. Therefore, choosing an appropriate Lifetime is desired. The general question about the Lifetime for a particular attribute is out of the scope of this paper. We will analyze the Lifetime affect on system security and efficiency in a later section (Section V).

Attributes can have different Lifetimes because they use different fading functions. Basically, an attribute that tends to change frequently, e.g., local location attributes (street, neighborhood), should have relatively smaller Lifetime to make the old attributes expired as soon as possible. In contrast, an attribute supposed to change less frequently or not change at all, such as high level locations (state, country), may have a bigger value to avoid unnecessary updates.

2) Security: Collisions may be caused by introducing the fading function. It is possible that attribute A's value at time t_1, say att_{A,t_1}, is exactly the same as attribute B's value at t_2, say att_{B,t_2}, if fading function is not carefully designed. This is never allowed to happen since a receiver can use att_{B,t_2} to decrypt cipher encrypted by att_{A,t_1} [1], which totally screws up ABE.

Key Master is the one responsible for designing and publishing fading functions. By carefully choosing fading function for

[1]From the mathematical point of view, two attributes having the same value are regarded as the same attribute

each attribute at Key Master, collision can be avoid. Generally, functions like non-descending functions are suggested. And replacing old fading functions after a period of time is also recommended for security consideration.

Another security concern is that a sender may choose out-of-date attribute value, other than the "latest", to encrypt a message. This is allowed, and sometimes even desired, i.e., a sender broadcasts a message to receivers who were at Westwood Blvd. 5 minutes ago. It is the sender's responsibility to choose which value to use and who can see his message. In most cases, the sender uses the latest attribute value to ensure the real-time constraint.

Besides fading, DABE is identical to CP-ABE. Thus it is as secure as CP-ABE.

3) Overhead: DABE causes unnecessary updates if a receiver still owns the property when the corresponding attribute expires. In that case, he should request an attribute from Key Master to replace the expired attribute. This is not only DABE's concern. CP-ABE also has this kind of problem because Key Master cannot predict the correct expiration time for a private key. By allowing attribute updates individually, DABE avoids CP-ABE much more cumbersome private key update. When attributes change fast, the overhead produced by private key updates far outweighs the unnecessary updates in DABE. Our simulation results in Section V confirm this claim.

Fading functions do not bring much overhead. The set of available attributes is maintained by Key Master. So, before encrypting, a sender has to check with Key Master to see which attribute can be used at least once. A sender can download fading function when he downloads the initial attribute value. The sender does not need to check with Key Master anymore as long as the fading function is not replaced, if he wants to reuse the same attribute later.

V. Simulation

We evaluate the performance of DABE using a scenario similar to SAT. Locations are encoded into attributes. Instead of using road names [5], we use Quadrant Location System to represent a car's location.

In a real system, the vehicles have several other attributes (static and dynamic) besides location. However, to test DABE we need to concern ourselves with dynamic attributes. And, instead of considering various types of dynamic attributes (e.g., fuel level, remaining ammunitions, remaining food, Taxicab responding to a call or cruising around or customers on board , etc.) we just assume for simplicity and without loss of generality that ALL dynamic attributes are location related, and differ from each other due to differing granularity or reference coordinates.

A. Quadrant Location System

In Quadrant Location System (QLS), an area is divided into 4 squares. Each square is a quadrant, labeled from 1 to 4 in counterclock order. Each quadrant is recursively subdivided into 4 smaller quadrants until granularity requirement is reached. Quadrants having the same size are in the same layer. As Fig. 3 shows, there are 4 quadrants in layer 1 and 16 in layer 2. A object's Quadrant location is a series of digits. Each digit represents the object's location (which quadrant) in that layer. Digits on the left represents lower layer (with bigger size) and digits on the right stands for the higher layer quadrant locations. In Fig. 3, A is in the third quadrant in layer 1 and the first quadrant in layer 2, so A has encoded quadrant location "3,1".

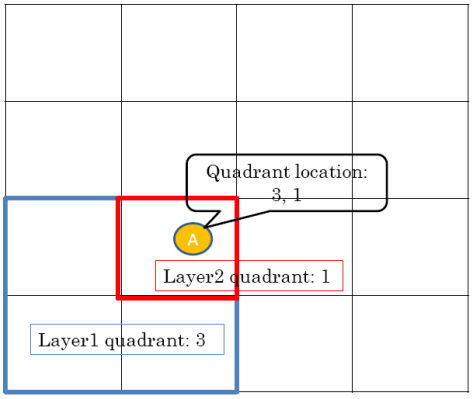


Fig. 3. Quadrant Location System

More precisely, each quadrant represents an attribute. In the above example, A uses two attributes to indicate his current location, one for layer 1 and the other for layer 2. Ideally, when a car enters a new quadrant, he gets an attribute for the new quadrant and the attribute for the quadrant that he previously stays expires.

B. Impact of Fading Rate

1) Efficiency and Security: It takes longer time to traverse a lower layer quadrant (i.e., smaller layer number and bigger size) than a higher layer quadrant. That means that a lower layer attribute is updated less frequently than a higher layer attribute. Accordingly, higher layer attributes are supposed to have smaller Lifetime. In our experiment, we evaluate the impact of fading rate on security and efficiency. More specifically, to measure efficiency and security, we define two metrics.

Multiple Keys: In one Lifetime, a car may traverse several quadrants in the same layer and get an attribute for each. Attribute value does not change during one Lifetime interval, so all these attributes are simultaneously valid. We claim this is a security violation: a user should not have valid attributes for locations other than the current one. The number of illegal attributes, which is equal to the number of quadrants that a car traverses in one Lifetime minus one, is defined as Multiple Keys.

Update Frequency: If a car resides in one quadrant for a long time, it will get a new attribute for the same quadrant for each Lifetime expiration. For example, if a car takes 60s to traverse a quadrant and Lifetime for that quadrant is 15s, it will carry out 4 updates as undesirable waste. Update

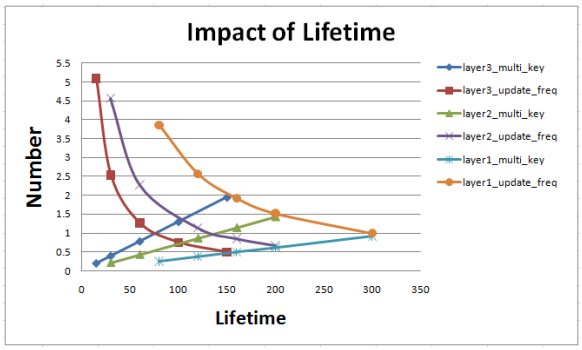


Fig. 4. Impact of Lifetime on Multiple Keys and Update Frequency

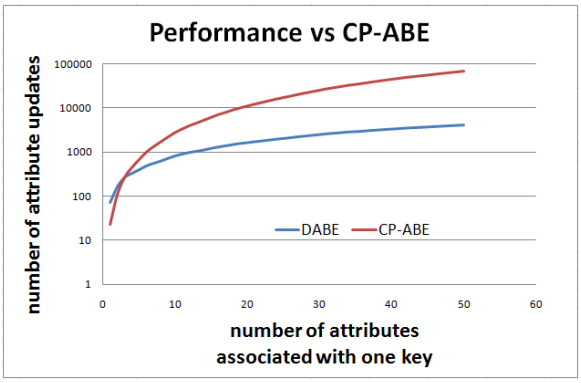


Fig. 6. Performance vs CPABE

Frequency reflects efficiency.

2) Setup: Our simulation uses VanetMobiSim [13]. Ten nodes move in a $2400m \times 2400m$ area. Each simulation run lasts for 3600 seconds. We put 10 nodes (cars) in the simulation. The cars have maximum speed of 30m/s and minimum speed of 14m/s. For realism, we use TIGER maps [10] and Intelligent Driving Model [11], [12]. Two TIGER maps are used, one is Los Angeles$(+34063690, -118445289)$, and the other is Washington D.C.$(+38890000, -77020000)$. We define a 3-layer QLS (Quadrant Location System). That is, every car uses 3 attributes to identify its location. Layer 1 quadrant has edge length of 1200m, layer 2 600m and layer 3 300m respectively. We run each experiment 10 times to get average statistics. A different random seed is chose for each run[2].

3) Variable Lifetime experiment: Fig. 4 shows that the Multiple Keys behavior grows linearly with Lifetime while Update Frequency decreases. These results are expected because of the very definition of Lifetime and Update Frequency. When Lifetime gets bigger, in most cases, a car travels longer distance thus passing more quadrants in one Lifetime and updating less and less. At the same time the car will pick up multiple "illegal" keys for quadrants it is not residing in.

However, the above curves only reflect the average behavior. Fig. 5(a) shows the cumulative distributions[3]. The lower curve is Lifetime $= 200$s, and the upper one is for Lifetime $= 30$s. As Lifetime decreases, most users pick up fewer Multiple (illegal) Keys. Small Lifetime also improves worst case, e.g., users have 1 or 0 Multiple Keys during most time when Lifetime is 60s while they have more than 1 Multiple Keys for nearly half of the simulation time when Lifetime is 200s. Update Frequency is the opposite of course. Fig. 5(b) indicates that the Lifetime increase improves worst case situation. Some users end up updating 10 times or more in one quadrant if Lifetime is 30s; with Lifetime $= 200$s, the worst we can get is updating twice in one quadrant. It is quite obvious that there is a tradeoff between Multiple Keys (leading to security violation) and Update Frequency (leading to extra line and processing overhead). This is expected. It reflects the usual tradeoffs between security and efficiency.

[2]Random seed affects initial location, path selection and speed

[3]CDF stands for Cumulative Distribution Function

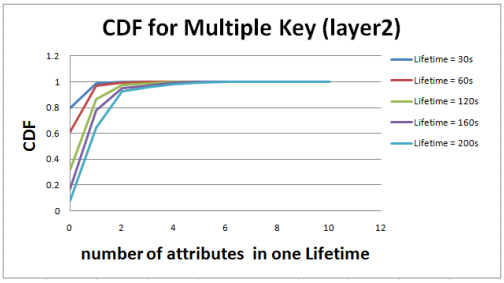
C. Scalability

1) Setup: We next analyze the scalability of DABE and compare the results with CPABE. Instead of using a single QLS, we use a set of 3-layer QLSs to create an environment with a large number of dynamic attributes. To make these attributes different, each QLS has a vertical and/or horizontal shift from original QLS. That is, if A has location of "$1, 1, 4$" in one QLS, it may have location of "$1, 2, 4$" in another QLS. If there are n QLSs, a car will have $3n$ attributes. We use 40s, 75s and 160s as Lifetime for 3 to 1 (larger Lifetime for larger geographic area). From layer 1 results in Fig 4 we recall that for Lifetime $= 160$s, the Multiple Keys probability is 0.5 and the Update Frequency is 2 per Lifetime. So are layer 2 and layer 3 attributes. This is an acceptable security/efficiency tradeoff leading to meaningful scalability results.

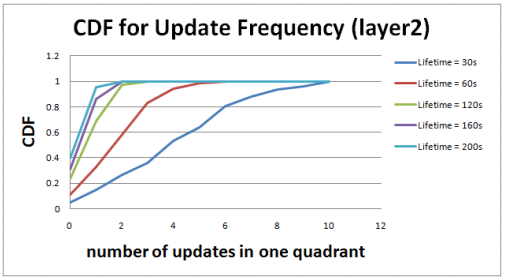
As we know, CP-ABE uses the expiration timestamp attribute to revoke private keys. Assume this approach works perfectly (while it might not in reality), a car updates its ENTIRE private key only when necessary (when it enters a new quadrant). In this perfect CP-ABE system, the old private key expires exactly when the car gets the a new private key. However, it always updates of the entire private keys, not just the attribute in the keys.

We increase the size of the private key from 1 attribute to 50 (proportionally to the QLS domains used) and measure how many attributes are transmitted from Key Master to cars when updating. For example, if a private key is associated with 15 attributes, in CP-ABE, there are 15 attributes transmitted in one update. Instead, only 1 attribute is transmitted each time in DABE. The more attribute updates, the more network load, and the more CPU resources. So attribute updates can be viewed as an index for system performance. Attributes are randomly picked from 50 QLSs (i.e., 150 attributes in total); we assume each QLS contributes one attribute at most to the private key. That whether an attribute is from layer 1, layer 2 or layer 3 is decided randomly.

2) Comparison with CP-ABE: Fig. 6 shows that the number of attributes that must be transmitted (equal to the number



(a)



(b)

Fig. 5. A deeper look at the impact of Lifetime on Multiple Keys and Update Frequency (Cumulative Distributions)

of attributes that Key Master has to generate) significantly increases when the number of attributes in the private key grows. It illustrates that the private key refresh approach in CP-ABE is not scalable with the increase in dynamic attributes. As for DABE, we note that it suffers more overhead than CP-ABE when the number of attributes in the private key is very small. It does shine, however, when the number of attributes associated with the private key grows. In fact, it scales enormously better than CPABE, with one order of magnitude improvement for 50 dynamic attributes. This result coincides with our analysis that "the overhead produced by private key updates far outweighs the unnecessary updates in DABE". It is worthwhile to point out that we assume CP-ABE updates only when necessary, so the performance of CP-ABE is a little overestimated leading to a CONSERVATIVE comparison.

VI. Conclusions

This work is a follow-up of previous research on Situation Aware Trust. SAT uses descriptive attributes to efficiently perform security policy control. It also builds mechanisms for predicting future trust situations, and; it transforms trust from Internet social communities to VANET trust in order to enhance and promote VANET applications. ABE was used as the basis to integrate the policy control and security services for SAT.

One of SAT main contributions was to divide attributes into two categories, static and dynamic, in order to improve efficiency. However, this does not help much since updating is still in terms of (private) keys, instead of attributes. In this paper, we introduce the concept of fading function to CP-ABE, allowing attributes "dynamic". DABE provides an efficient mechanism for attribute revocation that does not require the reissuing of the entire key. Namely, attributes can be updated independently from entire private keys, making key management in presence of dynamic attributes much more efficient and scalable.

We are now in the way to finding out the DABE's performance in the practical scenarios. Besides, we plan to investigate the feasibility of incorporating value compare predicates in policy tree [1] in the future so that the sender can control the lifetime of attributes.

References

[1] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *proceedings of the 28th IEEE Symposium on Security and Privacy*, Oakland, 2007.

[2] A. Sahai and B. Waters. Fuzzy Identity Based Encryption. In *Advances in Cryptology C Eurocrypt, volume 3494 of LNCS, pages 457C473*, Springer, 2005.

[3] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute Based Encryption for Fine-Grained Access Conrol of Encrypted Data. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2006.

[4] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters. Secure Atrribute-Based Systems. In *ACM conference on Computer and Communications Security (ACM CCS)*, 2006.

[5] Xiaoyan Hong, Dijiang Huang, Mario Gerla and Zhen Cao. SAT: Building New Trust Architecture for Vehicular Networks. *the Third International Workshop on Mobility in the Evolving Internet Architecture (MobiArch'08), ACM SIGCOMM workshop*, Seattle, WA. August 22, 2008.

[6] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612C613, 1979.

[7] A. Shamir. Identity Based Cryptosystems and Signature Schemes. *In Advances in Cryptology CCRYPTO*, volume 196 of LNCS, pages 37C53. Springer, 1984.

[8] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Pairing. *In Advances in Cryptology CCRYPTO*, volume 2139 of LNCS, pages 213C229. Springer, 2001.

[9] M. Raya, P. Papadimitratos, V. Gligor, J. Hubaux and S. EPFL. Data-Centric Trust Establishment in Ephemeral Ad Hoc Networks. *in Proceedings of IEEE Infocom*, 2008.

[10] http://www.census.gov/geo/www/tiger/index.html

[11] M. Treiber, A. Hennecke, and D. Helbing. Congested traffic states in empirical observations and microscopic simulations. *Phys. Rev.* E62, Issue2, August 2000.

[12] J. Haerri, M. Fiore, F. Filali, and C. Bonnet. VanetMobiSim: generating realistic mobility patterns for VANETs. *ACM International Workshop on Vehicular Ad Hoc Networks (VANET)*, 2006.

[13] http://vanet.eurecom.fr/

Appendix
Mathematical Construction

The mathematical construction of DABE is based on Bethencourt et al. [1] 's constrcution of CP-ABE. Most part are exactly the same. Readers should pay attention to the differences.

Definition: $\mathbb{G}_0$ is a bilinear group of prime order p. g is the generator of $\mathbb{G}_0$. $e : \mathbb{G}_0 \times \mathbb{G}_0 \rightarrow \mathbb{G}_1$ denotes a bilinear map.

Define Lagrange coefficient $\Delta_{i,S}(x) = \prod_{j \in S, j \neq i} \frac{x-j}{i-j}$ where $i \in \mathbb{Z}_p$ and S is a set of elements in $\mathbb{Z}_p$.

Define a hash function $H : \{0,1\}^* \rightarrow \mathbb{G}_0$ as a random oracle.

$att(j)$ represents the initial value of attribute j. Attribute value is time-dependent. It is depicted by $F(att(j), f_j(t))$, wheref_j is a fading function for attribute j and F is an operating function to compute latest attribute value.

Setup: Key Master chooses a bilinear group $\mathbb{G}_0$ of prime order p with generator g and two random exponents $\alpha, \beta \in \mathbb{G}_0$. Public key $PK = \mathbb{G}_0, g, h = g^\beta, e(g,g)^\alpha$ is published. Key Master keeps *Master Key* $MK = \beta, \alpha$ as secret.

Encryption$(PK, M, \mathbb{T})$: A party first chooses a polynomial q_x for each node x in the policy tree $\mathbb{T}$. Set the degree d_x of the polynomial q_x to be one less than the threshold value k_x of that node (for OR, threshold is 1; for AND, threshold is the number of children). Starting with the root node R, choose a random $s \in \mathbb{Z}_p$ and sets $q_R(0) = s$. Then randomly choose d_R other points of the polynomial q_R to define q_R completely. For any other node x, it sets $q_x(0) = q_{parent(x)}(index(x))$ (In $\mathbb{T}$, every children node are numbered starting from 1. Index(x) returns such a number associated with node x), and chooses d_x other points randomly to completely define q_x. The ciphertext is given as follows:

$$CT = (\mathbb{T}, \tilde{C} = Me(g,g)^{\alpha s}, C = h^s;$$
$$\forall j \in S : C_j = g^{q_j(0)}, C'_j = H(F(att(j), f_j(t)))^{q_j(0)})$$

Key Generation(MK, S): Key Master chooses a random $r \in \mathbb{Z}_p$ and random $r_j \in \mathbb{Z}_p$ for each attribute $j \in S$. Then compute private key as

$$SK = (D = g^{\frac{\alpha+r}{\beta}},$$
$$\forall j \in S : D_j = g^r H(F(j, f_j(t_i)))^{r_j}, D'_j = g^{r_j})$$

Key Master stores that random number r for that user and uses it to generate attributes for that user in the future:

$$ATT = (\forall j \in S : D_j = g^r H(F(j, f_j(t_i)))^{r_j}, D'_j = g^{r_j})$$

Decryption(CT, SK): A receiver uses his own private key to decrypt an encrypted message. He/she first uses a recursive algorithm DecryptNode(CT, SK, x).

If the x is a leaf node for time t, let $i = att(x)$. When x is not in a user's private key, DecryptNode$(CT, SK, x) = NULL$. If a user has attribute x in his private key, then:

$$\begin{aligned} DecryptNode(CT, SK, x) &= \frac{e(D_i, C_x)}{e(D'_i, C'_x)} \\ &= \frac{e(g^r H(F(x, f_j(t_t)))^{r_j}, g^{q_y(0)})}{e(g^{r_j}, H(F(x, f_j(t_t)))^{q_y(0)})} \\ &= \frac{e(g^r, g^{q_y(0)}) e(H(F(x, f_j(t_t)))^{r_j}, g^{q_y(0)})}{e(g^{r_j}, H(F(x, f_j(t_t)))^{q_y(0)})} \\ &= e(g,g)^{r q_x(0)} \end{aligned}$$

When x is a non-leaf node, for all nodes z that are children of x, it calls DecryptNode(CT, SK, z) and stores the output as F_z. If all the children have valid returns (not NULL), compute

$$\begin{aligned} F_x &= \prod_{z \in S_x} F_z^{\Delta_{i, S'_x}(0)} (i = index(z), S'_x = \{index(z) : z \in S_x\}) \\ &= \prod_{z \in S_x} \left(e(g,g)^{r \cdot q_z(0)}\right)^{\Delta_{i, S'_x}(0)} \\ &= \prod_{z \in S_x} \left(e(g,g)^{r \cdot q_{parent(z)}(index(z))}\right)^{\Delta_{i, S'_x}(0)} \\ &= e(g,g)^{r q_x(0)} \text{(using polynomial interpolation)} \end{aligned}$$

If a user has all the attributes to satisfy the policy tree, he/she can successfully obtain $A = DecryptNode(CT, SK, R) = e(g,g)^{r q_R(0)} = e(g,g)^{rs}$ by recursively calling DecryptNode function from leaf node to root node. Finally, decrypt message by

$$\tilde{C} / (e(C, D)/A) = \tilde{C} / (e(h^s, g^{(\alpha+r)/\beta} / e(g,g)^{rs}) = M$$