S-TGDH, secure enhanced group management protocol in ad hoc networks

Frédéric Cuppens, Nora Cuppens-Boulahia, Julien Thomas

HAL Id: hal-00426444

https://hal.science/hal-00426444

Submitted on 26 Oct 2009

S-TGDH, secure enhanced group management protocol in ad hoc networks

Frederic Cuppens, Nora Cuppens, and Julien Thomas

GET/ENST Bretagne, 2 rue de la Châtaigneraie, 35576 Cesson-Sévigné Cedex, France

{**julien.thomas, frederic.cuppens,nora.cuppens**}**@enst-bretagne.fr**

Abstract— **We present and analyse a secure protocol for group management, in large and dynamic ad hoc networks. The protocol we suggest relies on the TGDH protocol.**

In comparison with the previous solution, our algorithm helps to uniformly dispatch the group key calculus on each node, and the global cryptographic tree is optimized. Moreover, we propose an authentication algorithm.

Our algorithm provides several well-known security properties, such as nodes authentication, messages freshness, passive attacks resistance and known key attack resistance.

I. INTRODUCTION

Group communication has been studied for a while, in wired and wireless networks. With the emergence of security needs in wireless network and more specifically in ad hoc networks (for military's applications or consumers ones, for instance with multi-players games or tele conferences), several protocols suggest a way to get secured group communications. To provide group communication privacy, we need to have a common secret key. Efficiently managing group keys in mobile networks, such as ad hoc networks, is difficult as every time the group changes (a node joins or leaves the group), the group key needs to be modified.

In this paper, we present and analyze a secured and authenticated group key management (S-TGDH), based on an existing solution, TGDH [KPT00]. The rest of this paper is organized in five sections. Section II reviews previous approaches for the distributed group key management and presents briefly the TGDH protocol. Section III and IV present our contributions. Section V gives complexity analysis of S-TGDH and some comparisons with TGDH. Finally, section VI concludes and discusses future works.

II. RELATED WORKS

A variety of solutions have been proposed for group management. Some of them rely on centralized management, for instance Simple Key Distribution Center (SKDC). The logical key hierarchy (LKH) protocol [WGL98] is another group management protocol in which each member gets a place in a cryptographic tree managed by a centralized server which performs all the group management operations.

Other papers study decentralized algorithms. In decentralized group management protocol, each node of a group has partial keys. When we put all of them together, we are able to create the group key, needed to (de)cipher communications. Several proposals have been suggested to manage the group key. We have for example the clique suite proposed by Steiner and al [AST00], in which every member introduces

its partial-key into the result generated by its preceding member and sends the new result to its following member.

Kim and al [KPT00] propose TGDH, which arranges keys in a tree structure (see the following section). TGDH is similar to the One-Way Function Tree (OFT) protocol [SM03] except that TGDH uses Diffie-Hellman instead of one-way functions for the group key generation.

Tree based solutions

An interesting aspect in TGDH and the other tree-based solutions is that the protocols try to dispatch the group key calculus on nodes of the group, depending on the tree structure. As our proposition relies on TGDH, we present the main aspect of this solution. More explanations are available in [KPT00].

Cryptographic principles: The TGDH cryptographic algorithm relies on the Diffie-Hellman solution adapted to the group problem. Thus, the arithmetic operations are performed in a group of prime order p with the generator α, where p (prime integer) and α (exponentiation base) are the ones described in the Diffie-Hellman protocol [Res99]. As described in the following section, the notion of Group Diffie-Hellman is due to the fact that the group key is generated using the Diffie-Hellman problem.

Each member possesses two personal keys: a private partial-key K_v and a public one BK_v, where $BK_v = \alpha^{K_v} \mod p$. The public group key is a function of the other nodes public keys and the current node private key. Let's say: for v, $groupkey = f(K_v, BK_0, ..., BK_{v-1}, BK_{v+1}, ...BK_n)$, where n is the group size and f is the group key function.

When the number of nodes increases, the number of parameters for f raises up too. However, the evaluations $f(K_v, BK_1, BK_2, ..., BK_w)$ and $f(K_w, BK_1, BK_2, ..., BK_v)$ may perform similar partial calculi (on BK_1 and BK_2). In order to prevent several nodes to perform similar calculi, we use the cryptographic tree to dispatch the global key evaluation.

The following example explain how the group key is computed and how the calculi are dispatched on the group's nodes. Let's suppose we have 6 members. One of trees that TGDH can produce is represented in the figure 1. One should remember that the intermediate nodes (ie the ones that are not leaves) such as <2,0> are virtual nodes, which means that

they have no real existence and that their keys are computed from their children : $K_{l,v} = (BK_{l+1,2v+1})^{K_{l+1,2v}} \bmod p$
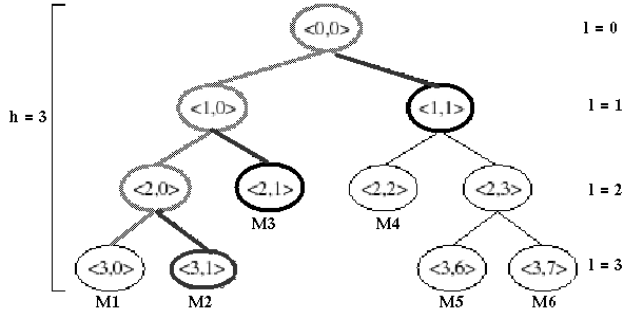

Fig. 1.   TGDH tree example

In this figure, the copath of the node M_1 is represented (in dark line). The copath of a member is the set of neighbour nodes (virtual or not) associated to the path we get when we want to reach the root of the tree.

The important aspect is that only the BKs on the copath are needed to evaluate the group key. When we use keys of virtual nodes, it means that the BK is already a computation of children's BKs. For instance, the BK of $<2,3>$ is a computation of those of M5 and M6. Thus, the function $f(K_v, ..BK_5, BK_6, .BK_w)$ is replaced by $f(K_v, ..BK_{parent}, ...BK_w)$, which reduces the number of operations a node has to perform. By comparison, when M_1 wants to compute the group key it needs to perform f with 3 BKs, and not 5 (the other members).

In TGDH specifications, we have five membership events:

- Join: a new member is added to the group
- Leave: a member is removed from the group
- Merge: a subgroup is added to the group
- Partition: a subgroup is insulated from the group
- Key refresh: the group key is updated

We describe only the two first events, due to lack of space (Merge and Partition are respectively meta-events of Join, and Leave).

Join protocol: Let's suppose we have n nodes in the group. The new member M_{n+1} initiates the join protocol by sending a join request message that contains its own blinded key BK. When current group members receive this message, they first determine the insertion node in the tree. If the cryptographic tree is well balanced, M_{n+1} joins at the root node. Otherwise, the insertion node is the shallowest rightmost node, where the join does not increase the height of the key tree. The sponsor is the rightmost leaf node in the sub-tree designed by the insertion node (see figure 2).

When the sponsor is found, it creates a new intermediate node, and promotes the new intermediate node to be the parent of both the insertion node and the new member node. The sponsor broadcasts then the new tree, which contains all blinded keys.
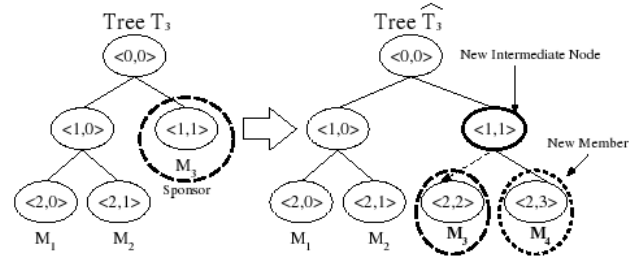

Fig. 2.   TGDH - join protocol

Leave protocol: Let's suppose that a member M_l leaves the group. In this case, the sponsor is the right-most leaf node of the sub-tree rooted at the leaving member's sibling node, as shown in figure 3. The former sibling of M_l is promoted to replace M_l's parent node. The sponsor picks a new secret, computes all keys on its copath, and broadcasts the new set of blinded keys to the group. This information allows all members to recompute the new group key.
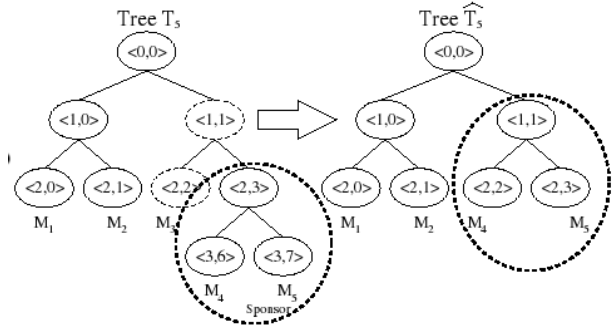

Fig. 3.   TGDH - leave protocol

III.   AUTHENTICATION ALGORITHM

In this section, we present the first part of our TGDH's variant: we introduce our authentication algorithm (not specified in TGDH's paper), which suggests

- a way to authenticate other nodes starting from a weak group password
- a way to authenticate source message during exchanges

First, we expound the main principles of the authentication part of our protocol and the corresponding authentication algorithm. Then, we present the different security properties our algorithms suite assures. The final part, about the keys management, describes which kind of key we advise to use, as it is important in security in general, though it does not immediately interact with our solution.

A.   Principles

In addition to the group management problem, the authentication in ad hoc network cannot rely on a centralized authority. Thus, before creating a group, we must find a way to diffuse authentication parameters to the other nodes of the group. Often, these parameters are RSA public keys, which is

the case in our proposition. However, in order to prevent from having a message number explosion, a temporary centralized authority has to be used. We call it the initiator node (M_n), as it performs the group exchange key initiation.

In many protocols, this RSA key distribution is not discussed. Thus, we can suppose that these parameters are exchanged via a private channel, or by static configuration. Theses solutions are not scalable and it is unrealistic to suppose that the static configuration can be performed in public (or personal) networks.

The Asokan-Ginzboorg protocol [AG00] suggests a way to create a strong secret starting from a weak key. The figure 4 sums up the algorithm, whose secured solution is proposed by Graham Steel, Alan Bundy, and Monika Maidl [SB05].

1. $M_n \rightarrow \text{ALL}$ $\quad: \{\!| \ M_n \ , E \,|\!\}_P$
2. $M_i \rightarrow M_n$ $\quad: \{\!| \ M_i \ , R_i, S_i \,|\!\}_E, i = 1, \ldots, n-1$
3. $M_n \rightarrow M_i$ $\quad: \{\!| \ \{Sj, j = 1, \ldots, n\} \,|\!\}_{R_i}, i = 1, \ldots, n-1$
4. $M_i \rightarrow \text{ALL}$ $\quad: M_i, \{\!| \ S_i, h(S_1, \ldots, S_n) \,|\!\}_K$, some i.

Fig. 4.   Improved Version of the Asokan-Ginzboorg protocol

In fact, it can also be used to handle RSA keys distribution: when creating a group, nodes only have to decide of a common temporary key. Starting from this weak secret key, we can use a challenge protocol (the group creation protocol) to perform an authenticated key exchange protocol. This can be seen as a strong assumption, but in fact it is quite obvious: when people want to create a group, they have a secret they all share, in order to be assured that others are the one they pretend to be (for example, they know each others' faces).

B. Algorithm

The exchange algorithm we propose is an adapted version of [SB05], which joins together the authentication and the group initiation processes. In the TGDH specification, no assumption is made about the group creation. It can be done for instance by the following initialization protocol: the initiator sends a create group message. Upon receiving an invitation response from a node M_i, the initiator adds M_i to the tree (thus, the place of M_i only depends on when it sends its invitation response). After receiving all the responses, the initiator sends the group tree to each member. We have here a 3-steps protocol.

If we perform a similar protocol, by using a variant of the Asokan Ginzboorg protocol, we get the 4-steps protocol described in the table I.

TABLE I
AUTHENTICATION PROTOCOL IN S-TGDH

1. $M_n \rightarrow M_i$ $\ :\ \{public_n, c1i\}_p, n1, signature_n (i = 1..n-1)$
2. $M_i \rightarrow M_n$ $\ :\ \{public_i, c1i, c2i\}_{public_n}, n2, signature_i$ $(i = 1..n-1)$
4. $M_n \rightarrow M_i$ $\ :\ \{c2i\}_{public_i}, n3, signature_n (i = 1..n-1)$
3. $M_i \rightarrow M_n$ $\ :\ agree, n4, signature_i (i = 1..n-1)$

In the first step, M_n, the initiator node, asks all the nodes if they want to join the group, using the weak secret. It sends its public RSA keys and a first challenge $c1i$, used to authenticate the node M_i. When M_i receives the message, it first deciphers the message, using the same secret. It can here perform a signature check.

At step 2, M_n can check if M_i knew the weak secret P by testing if $private_n(public_n(ci1')) = ci1$.

The step 3 is performed by M_n when it received the messages (step 2) from all the members. It can join the tree structure with this message. At this step, M_i can also check M_n identity, by testing if $private_i(public_i(ci2')) = ci2$.

The last step is needed for group confirmation from each node (to be sure no error occurred). If the underlying protocols are reliable, the step 4 can be ignored. But it depends on the routing protocol (for example, OLSR [CeA+03] uses UDP). Note that each message comes with its signature, which provides identity confirmations.

We made here the assumption that the group creation is based on a common group password, as we assume that no node has several identities in the network. Thus, if this property cannot be assured, it is better to replace the common group password by a 2-nodes password.

C. Security properties

In cryptographic protocols, several security properties have to be satisfied in order to keep the confidence in the group security. We present here those properties our algorithms preserve. Most of them are taken from the clique protocols specification [AST00]. These different properties have to be satisfied in order to comply with two basic properties : **outsider cannot read** and **outsider cannot send**. (note : some of these properties were already guaranteed by TGDH, we mark them by T)

Group management:

- **station authentication:** upon receiving a message from a node, we should be able to detect if the claiming node is the real one (no masquerade attack). This is provided by the signature mechanism, using the RSA keys.
- **keys independence** (T)**:** this property can be divided in two basics one, as described in the Hi-KD [HBBC05] protocol :
  - **Forward confidentiality**[1]**:** an old node must not be able to get a group key used after it left the group
  - **Backward confidentiality:** a new node must not be able to retrieve group secret keys used before it joined the group.

  These two properties are ensured by the key update mechanism upon each group event and the Diffie-Hellman properties (it is impossible to get a private key, no matter which public keys an attacker has).

---

[1]As precised before, this definition of the Forward confidentiality is a Hi-KD definition [HBBC05], and not the international definition, which is equivalent to the PFS definition mentioned later. We add this Hi-KD definition as we think that it defines well the security aspect of the group update process.

- **non-contributory protocol:** all the nodes in the group do not play the same roles, but a node's role cannot be chosen by itself. The fact that the role cannot be chosen is very important in order to prevent nodes from taking particular role.
- **message freshness:** in order to prevent malicious nodes from performing message replays, the protocol must certify the freshness of each message. In our solution, this is assured by nonces.

Group key:

- **key agreement protocol** (T)**:** a protocol is said to be a key agreement protocol if the secret key is computed from two or more parties and that no party can predict the resulting value.
- **implicit key authentication**(T)**:** each member M_i of the group M is assured that no member $M_q \notin M$ can learn the group key unless it is disclosed by a dishonest member $M_j \in M$
- **key integrity** (T)**:** the group key is computed only by using values issued by indoor members of the group. Extraneous contribution(s) to the group key cannot be tolerated even if it does not provide the attacker(s) with any additional knowledge.
- **Perfect Forward Secrecy (PFS)** (T)**:** the compromise of a key (long-term or session's key) cannot result in the compromise of past session keys.
- **known key attack resistance:** compromise of session keys does not allow: 1) a passive adversary to compromise keys of other sessions, or 2) an active adversary to impersonate one of the protocol parties. The signature mechanism prevents an adversary to impersonate a node, unless it manages to obtain this node's RSA private key.
- **passive attacks resistance:** a protocol is said to be resistant to passive attacks if it can prevent people from discovering secrets (such as the group key) by passive attacks, like eavesdropping. The group encryption mechanism, associated to group key updates (in order to prevent the group from using the same key during a long time), prevents attackers from discovering the current session key.

As all these properties are satisfied by our algorithms, we are able to keep the confidence in the group security.

D. Keys & signature management

Key validity: As we are in a distributed environment, updating a key is not immediate. Thus, it should be taken into account that an old group key should not be revoked immediately after the new one is built.

In [BBB+06], several advices are given about key management. One aspect, the cryptoperiod, has to be taken into account for key validity. "A cryptoperiod is the time span during which a specific key is authorized for use by legitimate entities, or the keys for a given system will remain in effect." In fact, a BK (or K) key in S-TGDH has three states. The first one, pre-partial key, refers to the fact that

the key as been created, but is not a part of the global secret key of each node (for instance due to the propagation delay for an update operation). Then we got the partial-key and finally the post-partial key. Some of the node use old keys (in post-partial state) and others use new keys (in pre-partial state), due to delays.

This can be summed up in figure 5. The time for each phase depends on the network, but also on the sensitivity of the information.
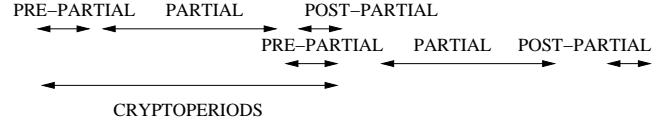

Fig. 5. Cryptoperiod explanation

Key size: An important aspect in cryptographic protocols is the key size. Nowadays, asymmetric key protocols use keys of 1024 bits length (and symmetric ones of 128 bits). According to a report [Wil01] of Lorraine C. Williams, 2048 bits length for asymmetric key and 128 bits length for symmetric key is far enough for the moment.

We thus recommend the use of these key lengths (depending on the country laws). For the global key, the use of a $hash_{128}$ or $hask_{256}$ functions is sufficient.

Nonces management: Two kinds of nonces can be used in cryptographic protocols. The first kind of nonces, based on random variables, assumes that all nodes know the previous used values. However, they should know all the nonces used since the RSA keys has been created, and not only since the group has been created. Thus, this solution is not well adapted to our problem. The second kind of nonces relies on temporal leashes [HPJ01]. This solution, most interesting for group management, implies several requirements such as :

- nodes should be able to synchronize their clocks
- the propagation time can be evaluated

For the first requirement, solutions as GPS can be used. Several synchronization protocols exist in ad hoc networks, but we think GPS is better due to traffic increase problems. The propagation time (or a validity time) is needed to give a validity interval for a nonce.

IV. ALGORITHMS OPTIMIZATIONS

For network protocols, two of the main complexity problems are how to manage a message and how many management messages are sent on the network. The optimizations of the first problem help to decrease the part of the node capacity associated to the protocol management, while the optimizations of the second help to reduce the consumption of energy and to increase the useful bandwidth.

TGDH proposes to create a cryptographic tree, which helps to dispatch the complexity within the nodes of the group. However, the way the sponsor is chosen and the fact that all the intermediate nodes in the tree are virtual lead to a non-uniform distribution of the global complexity, as

the left children nodes would have more and more works to perform. Our idea consists in having the less virtual nodes as possible which will help to dispatch uniformly the work. The following sections describe how we update the different TGDH algorithms to remove as many virtual nodes as possible.

Moreover, having less virtual node reduces the size of the tree, which has two consequences:

- the tree propagation is faster
- less BK keys are needed to build the group key

In order to lessen the complexity, we present in this section some modifications of the joining and leaving algorithms of TGDH. As we modified the tree aspect (by removing as many virtual nodes as possible), several cases have to be studied when these events occur. In order to satisfy the station authentication property, each message is sent together with its signature, using the RSA keys discussed before.

A. Joining algorithm

Join authorization: As we have now a secured and fully authenticated group, adding a node to the group has to be authorized. We make the assumption that when a node that belongs to the group wants another one to join the group, the authorization is valid. As the group key is known by all the members, this assumption is not so strong: nothing can prevent a node from giving the group key to a node which does not belong to the group. However, this assumption does not take into account malicious behaviors for which the goal is to attack the group management protocol, and not only to listen to communications. This problem is discussed in the paragraph *Trustworthy join*, as we can consider, in our case, multiple malicious nodes as multiple identities of a single malicious node.

Using the terminology of TGDH, the node that invites the new node is the sponsor. It has to perform three actions: first, it updates its partial keys, in order to assure the *backward secrecy property*. Then, it exchanges information with the new node. Finally, it sends the join message to all the members of the group, acting as the new node's sponsor.

Algorithm principles: When a sponsor adds the node to the tree, it should do it in a way that reduces the number of virtual nodes. One choice would be to get the best place in the whole tree, and to assign this position to the new node. However, this is not possible in a large network, or in an active ad hoc network, as conflicts could occurred (several sponsors would choose the same position, according to the fact that message propagation is not instantaneous). We then should perform local calculi, as there would not have any conflict.

Depending on the local context, two different joins can occur:

- if the parent is a virtual node, then the current node can (see the following requirements) choose to replace it by the new node (figure 6). The key replacement algorithm

is then simple: the BK key of the new node is computed by the sponsor node and propagated with the new sub-tree in the ADD message.

- if the parent node is not virtual, the node create a sub-tree with a virtual parent, whose children are itself and the new node (figure 7). Then, the sponsor node creates the virtual node BK and propagates it.

The key update algorithm (which comes after the BK keys replacement) is the same as the group creation one.

Algorithm requirements: In order to optimize the group management process, the algorithm must meat the following requirements:

- only leafs can act as sponsor, in order to avoid merges
- only left child can replace parent virtual node, again in order to avoid merges
- the global tree must stay well balanced, to dispatch work uniformly. This means that a node can create a sub-tree only if its depth does not exceed too much the average one.

These requirements are quite strong, but not so difficult to implement: when a node which is not a sponsor wants to add a new node, it can delegate it to another node. The fact that all the messages are authenticated helps to do that.
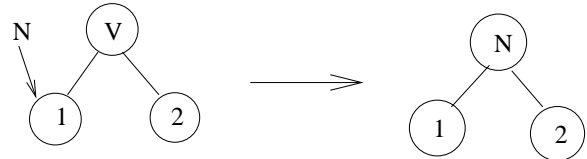
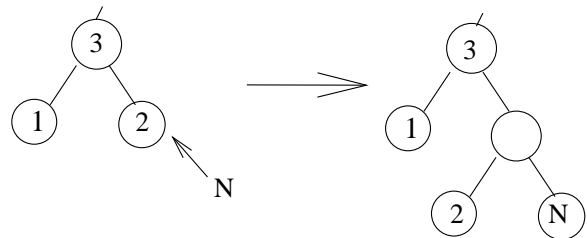Fig. 6.   Adding a node: first case

Fig. 7.   Adding a node: second case

Trustworthy join: In order to have correct group decisions, it is important to prevent a node from having multiple identities. If this is not possible, we propose a modification of the S-TGDH joining algorithm: a node is allowed to join the group only if at least τ_A nodes (similar to the threshold cryptography [aJL00]) allow it to join the group. The authorization to join a group can depend on several parameters, such as reputation[LI04].

Thus, the new node has to collect proofs from τ_A nodes, where $proof = \{nonce, nonce_{private\_RSA\_key}\}$.

This threshold value τ_A depends on several context properties such as the trustworthiness of each node and the dispersion of the nodes (scalable solution).

B. Leaving algorithm

In TGDH, a node departure is not authenticated. Thus, a malicious node can eject people from a group by only sending fake message, taking the identity of the node it wants to eject. With S-TGDH, the message authentication guarantees that this is impossible. Thus, the leaving algorithm is similar to the TGDH one. But we also introduce a leaving proof, to improve the security. Finally, we discuss about the silent leaving problem.

Leaving sponsoring: In order to properly leave a group, a node must tell its sponsor node it wants to leave. The sponsor then updates its group keys in order to respect the *forward secrecy property*. However, the way the sponsor node is chosen is not trivial. Depending on the way we do it, we could obtain conflict between several leaving nodes, but also with new nodes (which is more complicated to deal with).

According to how we manage the sponsor node for a new node, the election of the leaving sponsor node has to happen in the following way (see figures 8 & 9): the sponsor node is the rightmost leaf in the neighbour subtree. The use of its neighbour and not its own children prevents merge with local adding (as adding a new node only modifies the sub-tree of the sponsor node). The use of the right child provides a deterministic way to choose a sponsor, to perform check operation (it could have been the left, but using the right one dispatch the work well).

As for the adding operation, the leaving operation generates a BK message from the sponsor, which in turn generates a global key modification.

The leaving of a node generates a hole in the tree. In order to prevent from having an increase of virtual nodes, several cases should be taken into account:

- the parent node is virtual (figure 8). Then, the neighbour node can go one step upper in the tree, taking the place of this virtual parent
- the parent node is not virtual (figure 9). Then, a virtual node takes the place of the parent node, and this parent node takes the place of the leaving node
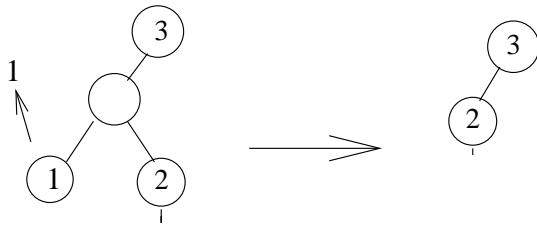

Fig. 8.   leaving operation: first case

Our algorithm provides an efficient way to realize leaving operation without merges. However, we can see a conflict problem when a node and its sponsor want to leave at the same time. In order to prevent deadlocks from happening, due to partial view problems, nodes have to wait for their sponsor's confirmation, before leaving the group.
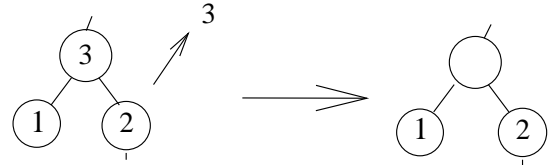

Fig. 9.   leaving operation: second case

Leaving proof: Though the messages are authenticated, fake leaving can happen from the inside : the sponsor node can claim to have received a leaving message from a node, and then every node in the group can trust it. Thus, when a node begins to act maliciously, it can destroy the group easily.

In order to make the group more stable, we decide to implement a proof system : when a node wants to leave the group, it sends a leaving message to its sponsor, message in which it puts a proof. The sponsor can then check this proof and forward it to all the members of the group. This proof uses the signature system $\{p, (p)_{private\_RSA\_key}\}$.

Silent leaving: As we have seen, deadlocks could have occurred if two nodes want to leave at the same time. However, other deadlocks can happen if nodes do not comply with the protocols. This can be intentional, or not (for example due to crash problems).

If this happens, local nodes can detect it and inform the others that a bad behaviour occurred. However, if only a single node is needed to correct this problem, a malicious node can here again send fake messages and destroy the group. Resolving this kind of attacks is quite complex, as the malicious node already belongs to the group. A convenient way to have a scalable and trustworthy solution is to use threshold system, in a similar way as threshold cryptography [aJL00]: a decision is taken if τ_L nodes agree on it. The value of the threshold τ_L depends on several parameters, such as the ones stated for τ_A. Obviously, the bad behaviour detection is efficient if less than

- τ_L nodes are contaminated, to have correct alerts
- $n - \tau_L$ nodes are contaminated, to signal bad behaviours

Using reputation mechanism can also prevent malicious nodes from performing collusions: only nodes with a good reputation are allowed to participate in the group decision process. By putting together the leaving and join restrictions, we get a more resilient group management protocol.

V. COMPLEXITY ANALYSIS

A. Optimizations analysis

The figure 10 sums up how a node can manage BK messages in the TGDH protocol, which are the most important messages exchanged when the group is created.

One can clearly see that the protocol is simple, but the problem with TGDH is that a lot of BK messages for virtual nodes are sent. The number of messages sent when a node updates its BK depends on its depth in the tree. Let's consider
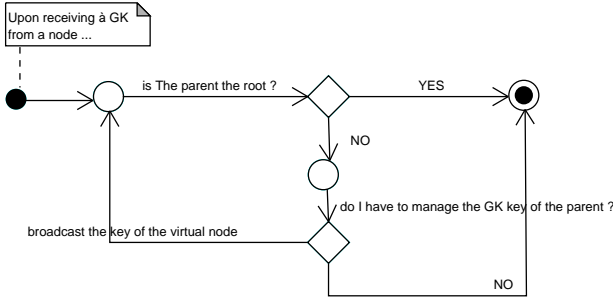
Fig. 10.   TGDH management messages

n nodes in the group. In TGDH, the depth of the tree is given by the formula $h = E(log2(n)) + 1$ and h BK keys have to be rebuilt (in order to get the tree root). Another problem is that as all intermediate nodes are virtual, so one of the leaves has to propagate these messages. The average number of BK keys a node should manage is $\sum_{i=1}^{h_s-1}(i \cdot (1/2)^{i+1}) + h_s \cdot (1/2)^h$.

The optimization we propose with S-TGDH (illustrated by the figure 11) helps to reduce the number of messages sent during a BK key update process, and also the number of messages sent by each node. As for TGDH, the number of messages sent during a key removal depends on the depth of the considered node. But contrary to TGDH, all the intermediate nodes are not obligatory virtual. Ideally, we have no virtual node. Thus, the average number of sent messages is $h_s = E(log2(\frac{n}{2} + 1))$ and the other interesting part is that a node may have to take into account only *one* BK message, if the parent is not a virtual node.
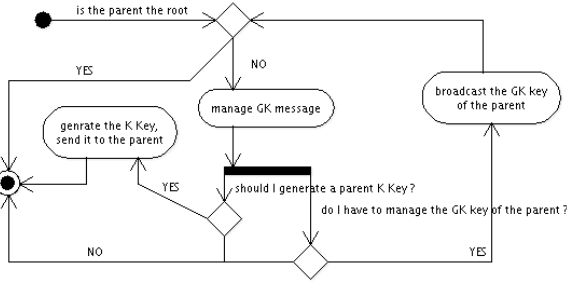

Fig. 11.   S-TGDH management messages

We also notice that the number of messages associated to a BK removal process depends on an implementation choice: either we can use a multicast process if the routing protocol allows it, or we can use unicast messages. Depending on this choice, the number of messages sent by the source and the intermediate nodes is quite variable. Thus, as it does not change the comparison between TGDH and S-TGDH, we choose to compare the number of BK keys that have to be changed and the average per node.

B. Algorithms complexity

In order to estimate the algorithm cost of each sub-protocol, we make the assumption that an optimal messages propagation algorithm can be chosen (see previous section).

The results are summed up in the tables II and III.

Group creation: The number of messages sent during the creation process (ie the authentication phase) is $4 \cdot (n-1)$, where n is the number of nodes if we consider unreliable network communications ($3 \cdot (n-1)$ otherwise). Comparing to the $3 \cdot (n-1)$ messages suggested by TGDH, the process produces an increase of 33% in the worst case though our approach provides a fully authenticated protocol. Besides, as explained below, this overhead is produce only during the group creation process and is compensated by the optimisations made in the other processes. All the algorithm complexity is supported by the initiator (as other nodes only have to check the challenge from the initiator node and to decrypt messages).

For the initiator node, the complexity is linked to the problem *where to insert a node in the graph*. With the implementation we propose, the complexity is minimal: the maximal cost is obtained if no virtual node is caught during the sub-tree analysis. We then have $h \cdot single\_cost$. The global cost is $\sum_{i=0}^{n} E(log2(i+1))$ which give a complexity of $n \cdot log2(n)$.

Group initiation: In this case, the number of nodes that get a BK message is maximal, since all the leaves send their BK keys. So we have $n_{leaf} = 2^{H_s-1} = \frac{n+1}{2}$ leaves at a depth $H_s = E(log2(n+1))$. The global cost is then $n_{leaf} \cdot \sum_{i=2}^{H_s}((2^i - 2) \cdot BKcost)$ for the BK messages cost, and $(2^{h_s-1} - 1) \cdot Kcost$ for the K messages cost where $h_s = E(log2(\frac{n}{2} + 1))$ is the mean depth for the node to take into account. (for TGDH, only the leaf receives BK messages , so cost $= n \cdot \sum_{i=1}^{H_t-1}((2i - 1) \cdot BKcost)$)

The different algorithm costs are evaluated according to our implementation of S-TGDH. Moreover, the algorithm complexity in S-TGDH is quite the same as in TGDH as the additional checks on S-TGDH only happened on the copath, which makes them fast.

- Kcost: $SC \cdot O(log2(n/2))$, mainly in order to build its public key
- BKcost: $SC \cdot O(n) + SC \cdot O(log2(n/2))$, mainly to create the parent private key, and perform checks

SC (Security_Cost) represents the cost to create the key.

Updating a key: The number of nodes that get a BK message is $\sum_{i=H_s-h_s+2}^{H_s}(2^i - 2)$ where $H_s = E(log2(n+1))$, as mentioned before (the sum on i is due to the removal of the intermediate BK keys, until we reach the tree root). The cost for the K messages is $h_s \cdot Kcost$.

C. Authentication overhead

With S-TGDH, all the messages are authenticated. The overhead is produced by the signature attached to each message, which is in fact given by the formula $signature = hash((message)_{key})$. The hash function, as for other cryptographic protocols, generates an output of constant length. For instance, the commonly used md5 hash function produces an output of 128 bits.

By default, the packets maximum size is the same as for all the Ethernet protocols: 1500 bytes. If we consider a transfer that tries to use as many payloads as possible, we got a decrease of 3% of the performance (using a 256 bits length signature). But very often, the packets do not have the maximum size. Thus, the performance decreases come only from the time spent to send the packets. For the WiFi, it takes 250 μs with 1 mbps (slowest speed) to send the signature.

Another impact of the packet's encryption is the time needed to decrypt it. Fortunately, only the recipient has to decrypt the packet, as the addresses are not encrypted. As we use a symmetric cryptographic process, the deciphering process is quite fast (for instance, one can have to perform only a xor operation).

D. Graphical comparisons

The following graphics illustrate the different results we described in the previous sections and compare them with the one we got for TGDH. We can see that both protocols produce the same cost (with better performance for S-TGDH), but S-TGDH generates fewer messages and distributes equitably the cost on each node.

TABLE II
AUTHENTICATION COST

	messages	Initiator	other nodes
TGDH	$3 \cdot (n-1)$	$O(n \cdot log2(n))$	-
S-TGDH	$4 \cdot (n-1)$	$O(n \cdot log2(n))$ & n challenges checks	1 challenge check

TABLE III
GROUP MANAGEMENT COST

	BKmessages cost	Kmessages cost
TGDH creation	$BK_{cost} \cdot O(n \cdot log2(n)^2)$	$K_{cost} \cdot O(n)$
TGDH update	$BK_{cost} \cdot O(log2(n)^2)$	$K_{cost}.O(log2(n))$
S-TGDH creation	$BK_{cost} \cdot O(n^2)$	$K_{cost} \cdot O(n)$
S-TGDH update	$BK_{cost} \cdot O(n)$	$K_{cost}.O(log2(n))$

VI. FUTURE WORKS

We have studied how to enforce security in the TGDH protocol, to propose S-TGDH, which improves several security aspects in group management. Moreover, we propose an authentication algorithm embedded in the S-TGDH group creation protocol. The optimization we propose helps to dispatch uniformly the complexity over every node.
However, message confidentiality in group communication is not the only security aspect to study. Thus, in future works, we will study other aspects such as anonymous routing and group cooperation, which would be interesting extensions of our current solutions. Moreover, proposing a way to evaluate reputation at the group layer (in addition to the ones at the lower levels) is useful to have a good group decision, in order to detect internal attacks (see section IV-B). Furthermore, a lot of security flaws in security protocols come from the fact that these protocols were not formally specified. So, we are currently investigating formal modeling and proofs of our current proposition. Finally, the current implementation of S-TGDH is a java simulator, which simulates the algorithm behaviour. An important dimension in ad hoc network is the mobility [Dav00]. Thus, implementing our protocol using the well known simulator NS-2 will help us to test our solution in mobile networks, using adequate mobility models, and to analyze the interaction with the underlying layers (for instance, OLSR or AODV).

REFERENCES

[AG00]   N. Asokan and Philip Ginzboorg, *Key-agreement in ad-hoc networks*, Computer Communications **23** (2000), no. 17, 1627–1637.

[aJL00]   Stanis aw Jarecki and Anna Lysyanskaya, *Adaptively secure threshold cryptography: Introducing concurrency, removing erasures (extended abstract)*, Lecture Notes in Computer Science **1807** (2000), 221–241.

[AST00]   Giuseppe Ateniese, Michael Steiner, and Gene Tsudik, *New multiparty authentication services and key agreement protocols*, IEEE Journal on Selected Areas in Communications **18** (2000), no. 4, 628–639.

[BBB+06] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid, *FIPS 800-57: Recommendation for key management–part 1: General (revised)*, NIST, U.S. Department of Commerce, May 2006.

[CeA+03] T. Clausen, P. Jacquet (editors), C. Adjih, A. Laouiti, P. Minet, P. Muhlethaler, A. Qayyum, and L.Viennot, *Optimized link state routing protocol (olsr)*, RFC 3626, October 2003, Network Working Group.

[Dav00]   V. Davies, *Evaluating mobility models within an ad hoc network*, Master's thesis, Colorado School of Mines, 2000.

[HBBC05] H. Ragab Hassan, A. Bouabdallah, H. Bettahar, and Y. Challal, *Hi-kd: Hash-based hierarchical key distribution for group communication*, IEEE INFOCOM '05, 2005.

[HPJ01]   Y. Hu, A. Perrig, and D. Johnson, *Packet leashes: A defense against wormhole attacks in wireless ad hoc networks*, Tech. report, Department of Computer Science, Rice University, December 2001.

[KPT00]   Yongdae Kim, Adrian Perrig, and Gene Tsudik, *Simple and fault-tolerant key agreement for dynamic collaborative groups*, CCS '00: Proceedings of the 7th ACM conference on Computer and communications security (New York, NY, USA), ACM Press, 2000, pp. 235–244.

[LI04]   Jinshan Liu and Valérie Issarny, *Enhanced reputation mechanism for mobile ad hoc networks.*, iTrust, 2004, pp. 48–62.

[Res99]   E. Rescorla, *Diffie-hellman key agreement method*, RFC 2631, June 1999.

[SB05]   G. Steel and A. Bundy, *Attacking group protocols by refuting incorrect inductive conjectures*, Journal of Automated Reasoning (2005), 1–28, Special Issue on Automated Reasoning for Security Protocol Analysis.

[SM03]   Alan T. Sherman and David A. McGrew, *Key establishment in large dynamic groups using one-way function trees*, IEEE Transactions on Software Engineering **29** (2003), no. 5, 444–458.

[WGL98]   Chung Kei Wong, Mohamed G. Gouda, and Simon S. Lam, *Secure group communications using key graphs*, Proceedings of the ACM SIGCOMM '98 conference on Applications, technologies, architectures, and protocols for computer communication, 1998, pp. 68–79.

[Wil01]   Lorraine C. Williams, *A discussion of the importance of key length in symmetric and asymmetric cryptography*, http://www.giac.org/practical/gsec/Lorraine_Williams_GSEC.pdf, January 2001.