Tree-based Group Key Agreement Framework for Mobile Ad-Hoc Networks*

Lijun Liao† and Mark Manulis
Horst-Görtz Institute, Ruhr-University of Bochum
IC 4/158, D-44801 Bochum, Germany
{lijun.liao,mark.manulis}@rub.de

Abstract

Design of protocols for mobile ad-hoc networks (MANETs) is generally tricky compared to wired networks, because on the one hand the increased communication constraints given by the limited bandwidth and frequent network failures, and on the other hand the additional computation and memory constraints due to performance limitations of mobile devices must be considered. We focus on the problem of the establishment of the shared key in mobile ad-hoc groups. This task can be achieved by means of a contributory group key agreement (CGKA) protocol that allows group members to compute the group key based on their individual contributions providing verifiable trust relationship between participants. As shown in this paper there exists currently no CGKA protocol for mobile ad-hoc networks that provides an optimal trade-off between communication and computation efficiency. Based on the comparison results of most suitable CGKA protocols we propose a new framework for the group key agreement in mobile ad-hoc networks. Theoretical analysis and experimental results show that our framework achieves optimal communication and computation efficiency compared to other protocols.

1. Introduction

Security of various group-oriented applications in mobile ad-hoc networks requires the establishment of a shared secret key (i.e., group key) known to all participants. Essential for the trust relationship between mobile participants is the absence of any trusted central authority (e.g., a group manager or a key server) that is actively involved in the computation of the group key. Hence, the group key management should proceed on the contributory basis allowing every participant to take part in the interactive computation of the group key. A class of so-called *contributory group key agreement* (CGKA) protocols has beed defined for this kind of trust relationship. According to [10] CGKA protocols that can be applied in mobile ad-hoc groups must provide *verifiable trust relationship*, i.e., although participants are trusted not to reveal secrets that can be used to compute the group key to other parties, they still should be able to verify the computation steps of the CGKA protocol. Another characteristic of mobile ad-hoc groups is the unpredictability of the occuring network failures and dynamic behaviour of mobile participants. Hence, the protocol for the group key management must be able to handle dynamic changes to the group formation without compromising the stated security requirements. In mobile ad-hoc groups communication constraints due to the limited bandwidth and frequent network failures, as well as computation and memory constraints due to the limited performance capabilities of participating mobile devices are challenging tasks that make the design of CGKA protocols more complicated. According to [10] currently known CGKA protocols for mobile ad-hoc groups provide either communication efficiency at the expense of the computation and memory costs, or computation and memory efficiency requiring a higher communication overhead. Note that not all existing CGKA protocols achieve the same security level with respect to the verifiable trust relationship. Actually, there is no CGKA protocol that achieves an optimal trade-off between communication, computation and memory costs, and provides stated security requirements. In this paper we propose a framework for the contributory group key agreement in mobile ad-hoc networks, called TFAN[1]. This framework consists of a protocol that combines properties of two (according to [10] most applicable) CGKA protocols for mobile ad-hoc networks: the communication efficiency of the

---

* This is a full version of the paper which appears in Proceedings of the 20th IEEE International Conference on Advanced Information Networking and Applications (AINA 2006), 2nd International Workshop on Security in Networks and Distributed Systems (SNDS 2006), April 18-20, Vienna, Austria.

† Both authors were supported by the European Commission through IST-2002-507932 ECRYPT.

1 TFAN - **T**ree-based group key agreement **F**ramework for mobile **A**d-hoc **N**etworks.

μSTR protocol, and the computation and memory efficiency of the μTGDH protocol.

2. Related Work

Several group key management protocols have been proposed for mobile ad-hoc group communication. The protocol of Asokan *et. al.* in [1] is a password authentication based key agreement protocol for small ad-hoc groups those members are on the same location (i.e., in one room). It is assumed that all members share a secret password. Obviously, the password-based authentication is a significant drawback for mobile ad-hoc groups built "on the fly" where no such pre-shared password is available. The protocol does not handle dynamic events and is less-efficient if the group size is not a power of two. The protocol of Besson *et. al.* [4] provides efficient mutual authentication and group key agreement for low-power mobile devices, and supports dynamic changes. However, it requires a wireless infrastructure with some powerful trusted server (base station) that performes heavy computations. According to the trust model in mobile ad-hoc groups, no such trusted authority is allowed. In [10] a number of CGKA protocols (i.e., Burmester-Desmedt (BD) [2], CLIQUES [15], STR [6] and TGDH [7]) that were originally designed for local- and wide-area wired networks has been analysed and optimized from the perspective of static and dynamic mobile ad-hoc groups, considering either the homogeneous or the heterogeneous distribution of the devices with respect to their performance limitations, and introduced verifiable trust relationship between participants. The resulting modified versions of these protocols (i.e., μBD, μCLIQUES, μSTR, and μTGDH) utilize *elliptic curve cryptography* (ECC) and eliminate redundancies of the original protocols to reduce the costs. One of the results of this work regarding dynamic mobile ad-hoc groups is that μSTR protocol is the most communication efficient protocol that should be applied in ad-hoc networks where communication constraints prevail over computation and memory constraints, whereas μTGDH protocol due to its better computation and memory efficiency should be used in the opposite case. Both protocols μSTR and μTGDH provide verifiable trust relationship, i.e., achieve a higher security level compared to μBD and μCLIQUES. Therefore, to achieve an optimal trade-off between communication, computation and memory costs at a higher security level we consider μSTR and μTGDH protocols as part of our framework.

3. Our Results

With respect to the results obtained in [10] we design a framework for the contributory group key agreement in mobile ad-hoc groups (TFAN) that achieves optimal trade-off between communication, computation and memory costs. The framework combines the communication efficient μSTR protocol and the computation and memory efficient μTGDH protocol. This combination is possible because of the similarities in the computation process of the group key in both protocols, which relies on the tree-based extension of the well-known elliptic curve Diffie-Hellman key exchange protocol (ECDH). Additionally to the theoretical security and complexity analysis the optimum of the trade-off between communication, computation and memory costs for TFAN is substantiated by the experimental results obtained from the implementation and simulation of the framework.

4. Mobile Ad-Hoc Group Communication

Constraints of Mobile Devices Mobile devices have limited performance capabilities due to the lower processor power (CPU clocks) and memory capacity, and are active for a limited period of time due to the battery dependent energy supply. Hence, one of the main goals for the design of CGKA protocols for MANETs is to optimize the computation, communication and memory costs of the protocol to increase the availability of the device. We remark that performance of mobile devices in CGKA protocols can be measured according to [11] by a specially designed benchmarking function, which considers the hardware parameters of the devices and performs protocol specific network and cryptographic operations.

Constraints in MANETs Due to the mobility of nodes ongoing communication session in MANETs suffers from frequent path breaks causing network failures to occur. Network failures may cause connection loss for one node (leave event) or a subset of nodes (partition event). Hence, CGKA protocols for MANETs must provide mechanisms to repair occured network failures, i.e., join protocol to repair leave events, and merge protocol to repair partition events. Limited bandwidth in MANETs additionally imposes a constraint on CGKA protocols in maintaining the group. Hence, CGKA protocols should be designed to achieve a minimal control overhead to maintain the group formation.

5. Contributory Group Key Agreement

Group Communication System CGKA protocols make use of an underlying group communication system (GCS) that must provide the following properties: (i) *public broadcast channel*, and (ii) *reliability*, i.e., all messages must reach their destination after being sent, and the order of sent messages must be preserved. Note that reliability in ad-hoc networks can be achieved using reliable multicast protocols for MANETs, like RDG [9]. Note that since communication channel is public CGKA protocols must consider the

existence of a passive adversary that is able to intercept all protocol messages.

Message Authentication In order to prevent impersonation attacks or man-in-the-middle attacks communication channels in CGKA protocols must be authentic. The authentication of messages can be achieved, for example, over digital signatures with certified public keys. We assume that every participant M_i has a certificate for its public key $pkey_i$ and uses the secret key $skey_i$ to sign own messages. Note that conventional PKI techniques with a trusted certification authority may not be available in a MANETs because of node mobility and missing infrastructure. The management of the public key certificates in MANETs is a current research topic, e.g., [3]. For our framework we assume that an appropriate management mechanism for public key certificates is available, and every participant sends authentic messages. Since authentication procedure is independent from the actual CGKA protocol we omit its explicit indication in the description of our protocols.

Verifiable Trust Relationship In mobile ad-hoc groups there is no trusted central authority that is actively involved in the computation of the group key, i.e., all participants have equal rights during the computation process. This is emphasized by the definition of the *verifiable trust relationship* (VRT) for CGKA protocols in MANETs in [10]. VRT assumes that: (i) group members are trusted not to reveal the group key or secret values that may lead to its computation to any other party, and (ii) group members must be able to verify the computation steps of the CGKA protocol.

Setup and Dynamic Events In order to build an ad-hoc group and to agree on the shared group key participants have to perform an interactive setup protocol. With respect to VTR every participant provides own contribution and should be able to verify the protocol steps towards the computation of the group key. After the group is initialized the protocol must respond to occuring dynamic events. Dynamic events may occur due to the network failures as described in Section 4, or may be triggered by the application, e.g., the application requires to join or exclude group members, or to split the group into independent subgroups (partition), or to combine independent groups into one common group (merge). We remark that CGKA protocols do not need to distinguish between events caused by network failures and events triggered by the application, and can treat both event types equally. Hence, CGKA protocol must contain subprotocols for join, leave, merge and partition. Additionally, it is preferred that group members are able to change their contribution and cause the update of the group key. For this purpose a so-called refresh protocol can be designed as part of the suite.

Security Requirements CGKA protocols must fulfill the following security requirements from [7]: *computational group key secrecy* (it must be computationally infeasible for a passive adversary to discover any secret group key), *decisional group key secrecy* (it must be computationally infeasible for a passive adversary to distinguish any bits of the secret group key from random bits), *forward secrecy* (any passive adversary being in possession of a subset of old group keys must not be able to discover any subsequent group key), *backward secrecy* (any passive adversary being in possession of a subset of subsequent group keys must not be able to discover any preceding group key), and *key independence* (any passive adversary being in possession of any subset of group keys must not be able to discover any other group key). Note that all these requirements consider only a passive adversary that might be able to intercept messages since the communication channel is public. Note also that forward secrecy provides security for subtractive events (leave and partition), since it prevents former group members from the computation of the updated group key. Similarly, backward secrecy provides security for additive events (join and merge), e.g., it prevents new members from discovering the previously used group keys.

6. Elliptic Curve Cryptography

There are no other techniques to agree on a shared key over a public channel than to use the public-key cryptography. However, public-key cryptography is costly to be applied in MANETs because of the limited device performance. *Elliptic curve cryptography* (ECC) is considered to be most applicable for mobile devices, because of the smaller key sizes (≈ 160 bits) and more efficient computation compared to other public-key systems. In [10] ECC has been used in the CGKA protocols μSTR and μTGDH in order to reduce the communication, computation and memory costs of the original protocols STR and TGDH, respectively. Operations of ECC are performed in groups of points of elliptic curves defined over finite fields. In the following we give a brief overview of ECC and elliptic curve Diffie-Hellman key exchange protocol (ECDH).

Let E be an elliptic curve over a finite field $\mathbb{F}_q$, such that $\mathbb{F}_q$ is either prime (q is a prime numer) or binary ($q = 2^m$, $m \in \mathbb{N}$) field. $E(\mathbb{F}_q)$ denotes a commutative group of points in E. Considering $G \in E(\mathbb{F}_q)$ as a point with high prime order t that devides $q-1$, there exists a subgroup of points of $E(\mathbb{F}_q)$ generated by G, i.e., $<G>= \{O, G, 2G, \ldots, (t-1)G\}$, where O is the point of infinity. The operation $Q = rP$, where $r \in [1, t-1]$ and $P \in <G>$ is called a scalar-point multiplication and its result Q is a point in $<G>$. Note that due to the hardness of the Discrete Logarithm problem on elliptic curves (ECDL) it is computationally hard to compute r given P

and Q [12]. The protocols μSTR, μTGDH and our framework require to map a point $Q \in < G >$ to an integer in the range $[1, q-1]$. The most natural way is to map Q to its x-coordinate, denoted $(Q)_x$. The following function $map : E(\mathbb{F}_q) \rightarrow \mathbb{N}$ ([14]) can be used for this purpose: if $q = p$ and p is prime then $map(Q) = (Q)_x$, else if $q = 2^m$, $m \in \mathbb{N}$, and $(Q)_x = (a_{m-1} \ldots a_1 a_0)$ with $a_i \in \{0, 1\}$ then $map(Q) = \sum_{i=0}^{m-1} 2^i a_i$.

ECDH Assume two participants M_1 and M_2 wish to agree on a shared secret key using ECDH ([14]) and the group $<G>$ is generated as described above. Every participant M_i generates its secret session random $k_i \in_{\mathcal{R}} [1, t-1]$ and computes its blinded version $K_i = k_i G$ (point in $<G>$). M_1 and M_2 exchange their blinded session randoms in an authentic manner, and compute the shared secret key $K = k_1 K_2$ and $K = k_2 K_1$ respectively. If the shared secret key has to be an integer then participants may apply the mapping function $map()$ on K.

Security of ECDH relies on the computational and decisional Diffie-Hellman assumptions in elliptic curves, denoted ECCDH and ECDDH, respectively. ECCDH means that given G, aG, $bG \in <G>$, it is hard to compute $(ab)G$. According to [12] ECCDH is hard for all types of elliptic curves. ECDDH means that given G, aG, bG, $cG \in <G>$, it is hard to decide whether $cG = (ab)G$. According to [5] ECDDH assumption could only be proven for special types of elliptic curves, i.e., non-supersingular and non-trace-2 elliptic curves. Therefore, only these special curves should be chosen for the implementation of ECDH.

7. Building Blocks

In this section we describe first the general key structure of μSTR and μTGDH procotols whose composition constitutes our framework TFAN. Then we give a brief overview of these protocols.

Binary Key Tree Structure Both μSTR and μTGDH protocols utilize the binary tree structure discribed in Figure 1. The height of the tree, denoted h, is defined as the maximal distance between a leaf node and the root node $\langle 0, 0 \rangle$, defined h. There are two kinds of nodes: intermediate and leaf nodes. The leaf node is associated with a member (device) and has no children. An intermediate node $\langle l, v \rangle$ has two children: the left child $\langle l+1, 2v \rangle$ and the right child $\langle l+1, 2v+1 \rangle$. In the following we describe the structure of the group key in ECC. Each node $\langle l, v \rangle$ is associated with a secret key $k_{\langle l,v \rangle}$ and a public key $bk_{\langle l,v \rangle} = k_{\langle l,v \rangle} G$. Secret keys of leaf nodes are chosen and kept secret by associated members. Every $k_{\langle l,v \rangle}$, $0 \leq v \leq 2^l - 1$, $0 \leq l \leq h$ is computed with ECDH key exchange from Section 6 using secret and public keys of both children nodes $\langle l+1, 2v \rangle$ and $\langle l+1, 2v+1 \rangle$, either as $k_{\langle l,v \rangle} = map(k_{\langle l+1,2v \rangle} bk_{\langle l+1,2v+1 \rangle})$ or $k_{\langle l,v \rangle} = map(k_{\langle l+1,2v+1 \rangle} bk_{\langle l+1,2v \rangle})$, where $map()$ is a point-to-integer mapping function from Section 6. Obviously, the group key $k_{\langle 0,0 \rangle}$ (secret key at the root) can be computed by any member M_i, associated with a leaf node $\langle l_i, v_i \rangle$, if it knows all public keys in its *co-path*, which consists of all sibling nodes in the member's *path*, which in turn consists of all nodes between $\langle l_i, v_i \rangle$ and $\langle 0, 0 \rangle$. Therefore, every member must save only the public keys in its co-path, its own secret key and the tree structure. However, in order to save members' computation costs in the protocols that handle dynamic events all secret keys in the path must also be stored. A member associated with a leaf node $\langle l_i, v_i \rangle$ stores $l_i + 1$ secret keys $k_{\langle l_i - \delta, \lfloor v/2^\delta \rfloor \rangle}$ for $0 \leq \delta \leq l_i$, own public key $bk_{\langle l_i, v_i \rangle}$, and l_i public keys in its *co-path*.

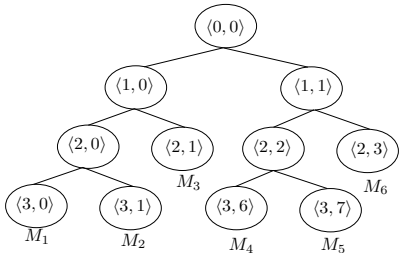

Figure 1. Binary tree

μSTR The μSTR protocol suite uses the tree structure in Figure 1 with the following constraints: For each $l > 0$ there are only two nodes $\langle l, 0 \rangle$ and $\langle l, 1 \rangle$. Considering a group of $n > 1$ members, the nodes $\langle l, 1 \rangle$ with $1 \leq l \leq n-1$ and $\langle n-1, 0 \rangle$ are leaf nodes, which are associated with members (devices). Figure 2 describes the setup protocol. In order to handle dynamic events μSTR defines a special role, called a *sponsor*. The role is temporary and is assigned to different members depending on the current tree structure and the kind of the dynamic event. We stress that a sponsor does not become a trusted central authority since according to VTR its messages can be verified by other members. Figure 3 describes the merge protocol for m trees, denoted $T_i, i \in [1, m]$. The number of members in T_i is de-

1. Each member M_i, where $\langle l_i, v_i \rangle \in \{\langle j, 1 \rangle | 0 < j \leq n-1\} \cup \{\langle n-1, 0 \rangle\}$, selects random $k_{\langle l_i, v_i \rangle}$, computes and broadcasts $bk_{\langle l_i, v_i \rangle}$.
2. The members $\langle n-1, 0 \rangle$ and $\langle n-1, 1 \rangle$ compute $(k_{\langle n-2,0 \rangle}, \ldots, k_{\langle 0,0 \rangle})$. The member $\langle n-1, 0 \rangle$ computes and broadcasts $(bk_{\langle n-2,0 \rangle}, \ldots, bk_{\langle 1,0 \rangle})$.
3. Each member M_i computes missing $(k_{\langle l_i - 1, 0 \rangle}, \ldots, k_{\langle 0,0 \rangle})$.

Figure 2. μSTR Setup

noted by n_i, w.l.o.g. $n_1 \geq n_2 \cdots \geq n_m$ is assumed. The idea of the protocol is to subsequently append a tree T_i on top of the tree T_{i-1} for all $2 \leq i \leq m$. Since every member knows only the public keys in its co-path, only the members $\langle n_i - 1, 0\rangle$ and $\langle n_i - 1, 1\rangle$ know all public keys in T_i. One of them acts as the sponsor M_{s_i} in the first round. The sponsor M_s is, however, a member assigned to the node with the lowest value for l in T_1. The join protocol for one member can be easily derived from the merge protocol. After the joining member broadcasts its public key the tree structure is modified by existing members, and the sponsor broadcasts it together with required public keys so that all members including the new member can compute the updated group key. Figure 4 describes the partition protocol of μSTR. The sponsor M_s is a member associated in the initial tree with the leaf node located directly below the leaf node of the leaving member with highest value for l. If no such node exists then M_s is a member associated with the leftmost node in the updated tree. The leave protocol for a single member is easily derived from the partition protocol.

1. The sponsor $M_{s_i}, i \in [1, m]$ broadcasts all public keys and the tree structure of T_i.
2. Each member updates the tree by merging all trees and removes all secret and public keys in the sponsor's path. The sponsor M_s changes it own secret key, computes all secret and public keys. The newly computed public keys are then broadcasted.
3. Each member M_i computes missing $(k_{\langle l_i-1,0\rangle}, \ldots, k_{\langle 0,0\rangle})$.

Figure 3. μSTR Merge

1. Each member updates the tree by removing the leaving members and removes all secret and public keys in the sponsor's path. The sponsor M_s associated with node $\langle l_s, v_s\rangle$ changes its own secret key $k_{\langle l_s,v_s\rangle}$, computes all secret keys $k_{\langle i,0\rangle}$ for $0 \leq i < l_s$, and public keys $bk_{\langle l_s,v_s\rangle}$ and $bk_{\langle i,0\rangle}$ for $1 \leq i < l_s$, and broadcasts the updated public keys.
2. Each member M_i computes missing $(k_{\langle l_i-1,0\rangle}, \ldots, k_{\langle 0,0\rangle})$.

Figure 4. μSTR Partition

The refresh protocol is similar to the leave protocol where refresher acts as the sponsor M_s with the only exception that the tree structure remains unchanged.

μTGDH The μTGDH protocol suite uses also the tree structure in Figure 1. However, unlike in μSTR the tree is kept balanced, i.e., paths of members assigned to the leaf nodes contain ideally the same number of nodes. Figure 5 describes the setup protocol of μTGDH. The sponsor of the (sub)tree is always the rightmost member. Similarly to μSTR, the sponsor's role in μTGDH protocols that handle dynamic events is temporary and assigned to different members depending on the current tree structure. Figure 6 describes the merge protocol for m trees, denoted $T_i, i \in [1, m]$. The height of T_i is denoted by h_{T_i}, w.l.o.g. $h_{T_1} \geq h_{T_2} \geq \ldots \geq h_{T_m}$ is assumed. The idea is to subsequently merge the two highest trees to a new tree which is then merged to the next tree until all trees $T_i, i \in [1, m]$ are merged. Two trees have to be merged in a way that keeps the resulting tree mostly balanced. For a detailed process of merging two trees we refer to [7]. Similarly to μSTR the join protocol for one member can be easily derived from the merge protocol. In this case only the new member broadcasts its public key in the first round, and the updated public keys together with the updated tree structure are broadcasted by the sole sponsor in the second round.

1. M_i selects random $k_{\langle l_i,v_i\rangle}$, computes and broadcasts $bk_{\langle l_i,v_i\rangle}$, sets $l = l_i - 1$ and $v = \lfloor v_i/2 \rfloor$.
2. M_i updates the tree structure, computes secret key $k_{\langle l,v\rangle}$, and public key $bk_{\langle l,v\rangle}$. The sponsor of the (sub)tree rooted at node $\langle l, v\rangle$ broadcasts $bk_{\langle l,v\rangle}$.
3. Members repeat steps 2 and 3 with $l = l - 1$ and $v = \lfloor v/2 \rfloor$ until every member computes the group key $k_{\langle 0,0\rangle}$.

Figure 5. μTGDH Setup

1. The sponsor M_{s_i} of the tree $T_i, i \in [1, m]$, updates its secret key, computes all secret and public keys (including $bk_{\langle 0,0\rangle}$) in its path, and broadcasts the updated public keys together with the tree structure.
2. Every member M_i merges the trees, removes the public keys in the path of the rightmost sponsor, and in paths of parent nodes of all other sponsors up to the root. The rightmost sponsor changes its secret key and computes the corresponding public key.
3. Every member M_i computes secret keys in its path until it blocks. Additionally, every sponsor broadcasts the public keys it has computed. Members repeat this step until every member computes the updated group key.

Figure 6. μTGDH Merge

Figure 7 describes the partition protocol. For a detailed description of how the partitioned members are removed from the tree and the sponsors are chosen we refer to [7]. Similarly to μSTR the leave protocol for a single member

1. Every member M_i updates the tree structure by removing the partitioned members' nodes and removes all secret and public keys in the sponsors' paths. The rightmost sponsor changes its secret key.
2. Every sponsor computes the secret and public keys in its path until it blocks and broadcasts the updated public keys.
3. Every member M_i computes secret keys in its path until it blocks. Members repeat steps 2 and 3 until all members are able to compute the group key.

Figure 7. μTGDH Partition

can be derived from the partition. In this case there is only one sponsor chosen as a member assigned to the rightmost leaf node in the subtree rooted at the sibling node of leaving member's node. The refresh protocol is similiar to the leave protocol with the only difference that the tree structure remains unchanged.

8. Tree-based Group Key Agreement Framework for Mobile Ad-Hoc Networks

In this section we present TFAN, a **T**ree-based group key agreement **F**ramework for mobile **A**d-hoc **N**etworks.

Idea Although the costs in μSTR and μTGDH protocols are significantly reduced compared to the original STR and TGDH protocols, the computation costs of μSTR are still higher than those of μTGDH, and the communication costs of μTGDH are in general higher than those of μSTR. Hence, the idea is to combine both protocols into one framework to achieve the optimal trade-off between computation and communication efficiency.

Tree TFAN uses the binary tree structure described in Figure 1 with some strict specifications. An example of a TFAN tree is shown in Figure 8. The nodes CN_i, $0 \leq i \leq h-1$, are called *cover-nodes*, the path from CN_{h-1} to CN_0 is called the *cover-path*. Each cover-node CN_i has two child nodes: the left child node is the cover-node CN_{i+1}, and the right child node is a root node of a subtree. These root nodes are called *sub-roots*, whereas the subtrees are called *cs-trees* (short for *Cover-Sub-Trees*) and are denoted by CT_i. The only exception is that the left child node of CN_{h-1} is also a cs-tree CT_h. The structure of the cs-trees depends on two parameters: the maximal allowed height (q) and the art (art) of cs-trees. The parameter art determines the cs-tree's art: S for a μSTR tree or T for a μTGDH tree. The cover-node CN_i is associated with the secret key k_{CN_i} and the public key bk_{CN_i}. The secret and public keys associated with a node $\langle l, v\rangle$ in a cs-tree CT_i are denoted by $k^i_{\langle l,v\rangle}$ and $bk^i_{\langle l,v\rangle}$, respectively.

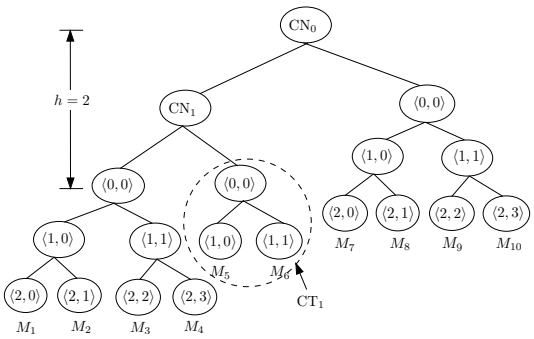

Figure 8. TFAN tree ($q = 2$, $art = T$)

Setup Given q and art, assume that n members wish to form a group. Every member broadcasts its public key with some other information required by the protocol. Members are then sorted according to some common criteria (e.g., sorted order of their public keys), w.l.o.g $M_1, \ldots, M_n$, and assigned to the leaf nodes of a TFAN tree. The idea behind the assignment of members to the leaf nodes is to divide the sorted list into $\lceil \frac{n}{b} \rceil$ blocks with a block length $b = q + 1$ if $art = S$, and $b = 2^q$ if $art = T$. Each block except for the last contains b members, and the last block the remaining members. The height of the TFAN tree is then $h = \lceil \frac{n}{b} \rceil - 1$. Figure 9 describes the setup protocol. Figure 10 gives an example of a TFAN tree with $art = T$[2] and $q = 2$. Every

1. Every member $M_i, i \in \{1, \cdots, n\}$ broadcasts its public key.
2. Members assigned to the same cs-tree CT_j execute the setup protocol of μSTR if $art = S$, or of μTGDH if $art = T$, and compute the public key $bk^j_{\langle 0,0\rangle}$. The sponsor of CT_j broadcasts $bk^j_{\langle 0,0\rangle}$.
3. Members assigned to CT_h and CT_{h-1} compute the secret keys in the cover-path. The rightmost member in CT_h computes and broadcasts the public keys in the cover-path.
4. Members assigned to CT_j, $j < h-1$ compute the secret keys $k_{CN_j}, \ldots, k_{CN_0}$.

Figure 9. TFAN Setup

member broadcasts its setup request and the public key. Assume that there are 14 members, e.g., M_1, M_2, $\cdots$, M_{14}. Since any cs-tree may contain at most 4 members, the following assignment to the cs-trees is applied: CT_3 consists of M_1, M_2, M_3, and M_4, CT_2 of M_5, M_6, M_7, and M_8, CT_1 of M_9, M_{10}, M_{11}, and M_{12}, and CT_0 of M_{13} and M_{14}. Members assigned to the same cs-tree execute the μTGDH

2 Due to space limitations we provide examples for TFAN trees with $art = T$. Note that examples with $art = S$ are similar.

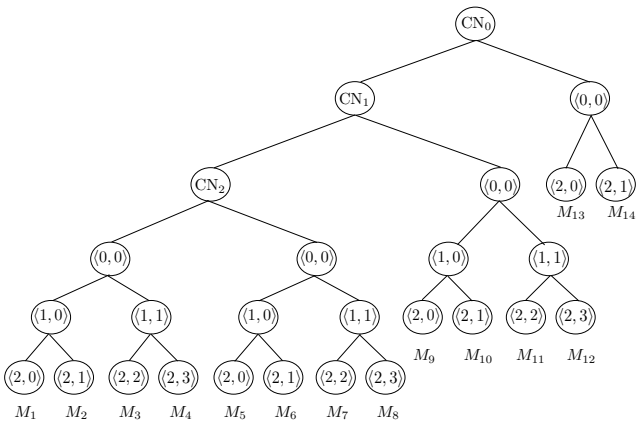

Figure 10. TFAN Setup ($q = 2$, $art = T$)

setup protocol. The sponsor of the third round is M_4. Since all members in CT_3 and CT_2 know all public keys of nodes in their co-paths, they compute secret keys of nodes in their paths. The sponsor M_4 additionally computes and broadcasts bk_{CN_2} and bk_{CN_1}. Upon receiving this message, the members in CT_1 compute k_{CN_1} and k_{CN_0}, and the members in CT_0 compute k_{CN_0}. The group key is k_{CN_0}.

Join Assume there is a group of n members $\{M_1, \cdots, M_n\}$, and a new member M_{n+1} wishes to join. M_{n+1} broadcasts a *join* request with its public key. The new member is inserted into the shallowest not fully filled cs-tree. The join event is then processed according to the join protocol of μSTR if $art = S$ or of μTGDH if $art = T$ with the extension that the sponsor computes all secret and public keys in its path. If all cs-trees are fully filled then a new cs-tree with the new member as its unique member and a new cover-node are created. The new cover-node becomes the new root of the TFAN tree with the old root as its left child and the sub-root of the new cs-tree as its right child. Figure 11 describes the detailed process of the join protocol. Consider an example with μTGDH cs-trees in Figure 12. Since CT_1 is the first not fully filled cs-tree, the new member M_{11} joins in CT_1. According to the μTGDH join protocol, M_5 is the sponsor. It changes its key pair

1. The new member M_{n+1} broadcasts its public key.
2. Every member M_i, $1 \leq i \leq n$, updates the tree structure by inserting M_{n+1} and removes all secret and public keys in the path of the sponsor M_s. The sponsor M_s changes its secret key, computes the secret and public keys in its path, and broadcasts the updated public keys together with the tree structure.
3. Every member M_i, $1 \leq i \leq n+1$ updates the tree structure and computes the changed secret keys in its path including the group key k_{CN_0}.

Figure 11. TFAN Join

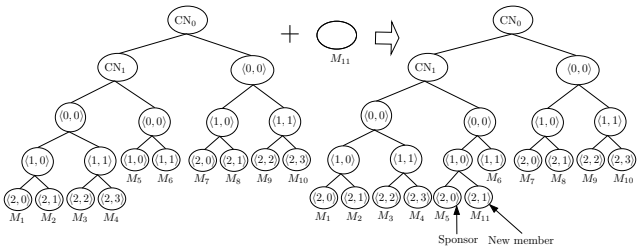

Figure 12. A TFAN Join ($q = 2$, $art = T$)

$(k^1_{\langle 2,0\rangle}, bk^1_{\langle 2,0\rangle})$ and computes the key pairs $(k^1_{\langle 1,0\rangle}, bk^1_{\langle 1,0\rangle})$, $(k^1_{\langle 0,0\rangle}, bk^1_{\langle 0,0\rangle})$, (k_{CN_1}, bk_{CN_1}), and the group key k_{CN_0}. M_5 then broadcasts the public keys $bk^1_{\langle 2,0\rangle}$, $bk^1_{\langle 1,0\rangle}$, $bk^1_{\langle 0,0\rangle}$, and bk_{CN_1} so that other members are also able to update the tree and compute the group key.

Leave Assume there is a group of n members, and M_l in the cs-tree CT_j leaves the group. If M_l is the only member in CT_j then the shallowest rightmost leaf node of CT_{j+1} if $j < h$, or of CT_{h-1} if $j = h$ is chosen as the sponsor M_s. Figure 13 describes the leave protocol. An example of

1. Every remaining member M_i updates the tree by removing the leaving member's node and its parent node, and removes all secret and public keys of nodes in the sponsor's M_s path. Additionally, M_s changes its secret key, computes the secret and public keys in its path, and broadcasts the updated public keys.
2. Every remaining member M_i computes the changed secret keys in its path.

Figure 13. TFAN Leave

a TFAN leave event is shown in Figure 14. M_8 in the cs-tree CT_1 is the leaving member. According to the μTGDH leave protocol members remove the leaf node of M_8 and its parent node $\langle 1,1\rangle$ from the tree, and M_7 is chosen as the sponsor. M_7 updates its secret key $k^1_{\langle 1,1\rangle}$ and computes the secret keys $k^1_{\langle 0,0\rangle}$, k_{CN_1}, and k_{CN_0}, and the public keys $bk^1_{\langle 1,1\rangle}$, $bk^1_{\langle 0,0\rangle}$, and bk_{CN_1}. It broadcasts then $bk^1_{\langle 1,1\rangle}$, $bk^1_{\langle 0,0\rangle}$ and bk_{CN_1} so that other members are also able to compute the group key.

Merge Assume m trees, denoted $T_i, i \in [1, m]$ have to be merged. Figure 15 describes the merge protocol. W.l.o.g. the trees are sorted from the highest to the lowest, i.e., $h_{T_1} \geq h_{T_2} \geq \ldots \geq h_{T_m}$. If two trees are of the same

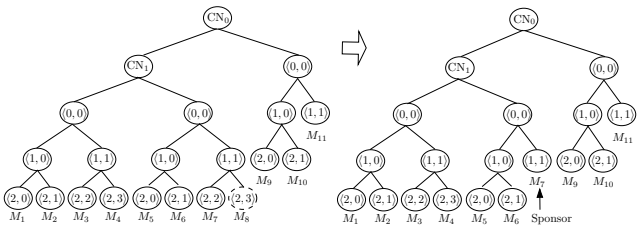

Figure 14. A TFAN Leave ($q = 2$, $art = T$)

height, some other criteria must be applied, e.g., a lexicographical order of the public keys of the respective sponsors. Considering the structure of a TFAN tree of height h only members assigned to the cs-trees CT_h and CT_{h-1} know public keys of all sub-roots. Hence, the sponsor M_{s_i} for the tree T_i is chosen as a member assigned to the rightmost leaf node in the cs-tree CT_h of the T_i. Note that a TFAN tree can be considered as a μSTR tree where whole cs-trees are used instead of sole leaf nodes. The idea behind the merge protocol of TFAN is to use the merge protocol of μSTR keeping the structure of the cs-trees in every merging tree T_i unchanged. The shallowest rightmost leaf node in T_1 becomes the sponsor M_s for the second round. The secret and public keys in the path of M_s are removed from the tree. Additionally, M_s updates its secret key and computes all secret and public keys in its path. The newly computed public keys are then broadcasted so that every other member is able to compute the secret keys in the own path. An example of the merge protocol with μTGDH cs-trees is given in Figure 16. Two groups T_1 of height 3 and T_2 of height 2 are merged. T_2 is appended on top of T_1. A new cover-node CN_1 is created with the root of T_1 as the left child and the root of CT_1 in T_2 as the right child. The sponsor M_8 updates own secret key, computes all secret keys $k^2_{\langle 0,0\rangle}$, k_{CN_2}, k_{CN_1}, and k_{CN_0}, and pubic keys $bk^2_{\langle 1,1\rangle}$, $bk^2_{\langle 0,0\rangle}$, bk_{CN_2}, and bk_{CN_1}, and broadcasts the newly computed public keys allowing other members to compute the group key.

1. Every sponsor $M_{s_i}, i \in [1, m]$ broadcasts the tree structure and public keys of all sub-roots of the tree T_i.
2. Every member updates the tree by merging all trees and removes all secret and public keys in the path of the sponsor M_s. M_s changes its secret key, computes all secret and pulbic keys in its path, and broadcasts the updated public keys.

Figure 15. TFAN Merge

Partition Consider a group of n members, and p of them are partitioned from the group. Group members that are chosen to compute and broadcast public keys within their cs-trees are called *cs-sponsors*. The group member that is

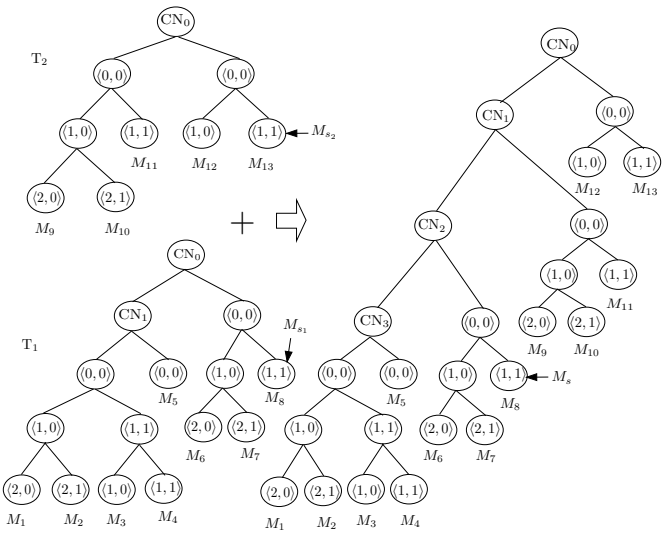

Figure 16. A TFAN Merge ($q = 2$, $art = T$)

chosen to compute and broadcast the public keys of the cover-nodes is called the *sponsor*. If all members in one cs-tree are partitioned then this cs-tree is removed from the TFAN tree so that no cs-sponsors are required. Otherwise the partition of members within each cs-tree is processed using the partition protocol[3] of μSTR (if $art = S$) or μTGDH (if $art = T$). The sponsor is chosen as described in the following. If all members of the deepest modified cs-tree leave the group then the sponsor is the rightmost member in the cs-tree directly below the removed cs-tree, or if no such cs-tree exists then in the greatest numbered remaining cs-tree above. In other case, if there are remaining members in the deepest modified cs-tree, the rightmost cs-sponsor in this cs-tree is chosen as the sponsor. Figure 17 describes the partition protocol. An example with μTGDH cs-trees is given in Figure 18. Since all members in CT_1 (i.e., M_8, and M_9) leave the group, this cs-tree and its parent node CN_1 are removed from the TFAN tree. According to the partition or leave protocol in μTGDH, M_5, M_{10} and M_{13} are the cs-sponsors for the partition of M_6, M_{11} and M_{12}, respectively. M_5 is also the sponsor. It updates its secret key, computes secret and public keys of all nodes in its path, and broadcasts $bk^1_{\langle 1,0\rangle}$, $bk^1_{\langle 0,0\rangle}$, and bk_{CN_1}. M_{10} and M_{13} broadcast their public keys. M_{13} computes then $k^0_{\langle 0,0\rangle}$, k_{CN_0}, computes and broadcasts $bk^0_{\langle 0,0\rangle}$, allowing other members to compute the group key k_{CN_0}.

Refresh Assume there is a group of n members assigned to a TFAN tree of height h. The member M_r, called the refresher, wishes to update its secret key. For this purpose the refresher changes own secret and public keys, com-

3 If there is only one member in the cs-tree that is partitioned then members of this cs-tree use the corresponding leave protocol instead.

1. Every remaining member updates the tree by removing the partitioned members and their parent nodes. The sponsor changes its secret and public key. Members in the remaining modified cs-trees perform the leave or partition protocol of μSTR (if $art = S$) or μTGDH (if $art = T$) with the following differences: The public keys of all cs-sponsors except for the sponsor are not removed. The non-sponsor cs-sponsor does not change its secret key. Additionally, the rightmost cs-sponsor in each cs-tree computes and broadcasts the public key of the corresponding sub-root.
2. The sponsor M_s computes the missing secret and public keys of the cover-nodes in its path, and broadcasts their public keys.
3. Every remaining member computes the missing secret keys of nodes in its path.

Figure 17. TFAN Partition

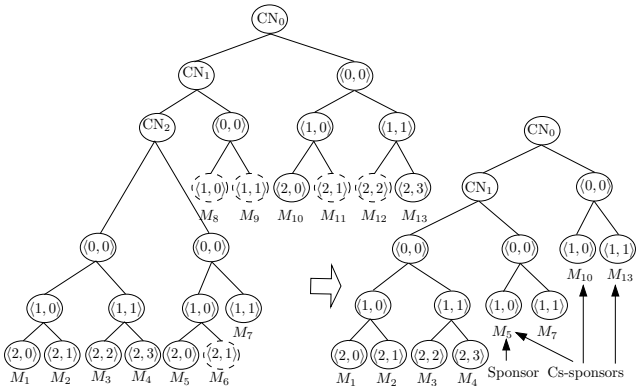

Figure 18. A TFAN Partition ($q = 2$, $art = T$)

putes changed secret and public keys of nodes in its path, and broadcasts the updated public keys. Upon receiving this message every other member removes known secret keys of nodes in M_r's path, updates the corresponding public keys in the own co-path, and recomputes the removed secret keys including the new group key.

9. Security Analysis

Our framework combines μSTR and μTGDH protocols using strictly defined cover-path as part of the binary key tree (TFAN tree). However, this structural modification of the key tree leaves the actual computation process of the group key unchanged, i.e., it still relies on the tree-based ECDH key exchange method applied in μSTR and μTGDH. Therefore, the security analysis of μSTR and μTGDH from [10] is also valid for our framework. For completeness we briefly discuss how TFAN fulfills the security requirements of Section 5. The *computational group key secrecy* of TFAN relies on the hardness of the ECCDH problem. The *decisional group key secrecy* of TFAN relies on the hardness of the ECDDH problem. Thus, a passive adversary A although being able to intercept TFAN messages sent over a public channel and obtain public keys can neither compute nor distinguish the group key. Therefore, A can discover the group key only if learns at least one secret key. Due to the hardness of the ECDL problem the adversary is not able to reveal these values from their public counterparts. For *backward secrecy* we show that any A being a joining member is not able to obtain any of the previous used group keys. Assume, A becomes a new member of the group and is located in the Cover-Sub-Tree CT_a of the TFAN tree at node $\langle l_a, v_a \rangle$ for some $l_a \in \{0, \ldots, q\}$. As a new member A is able to compute all secret keys in its path up to the root of the TFAN tree, i.e., $k^a_{\langle l,v \rangle}$ for all $l < l_a$ and $v = \lfloor \frac{v_a}{2^{l_a - l}} \rfloor$, and k_{CN_i} for all $i \leq a$. The sponsor of the additive event is either located at node $\langle l_a, v_a - 1 \rangle$ if CT_a was not fully filled or is the shallowest rightmost node in the Cover-Sub-Tree CT_{a+1}. It changes own secret key and causes the change of all secret keys in its path up-to the root of the TFAN tree. Hence, in both cases A can only compute the changed secret keys, and is therefore not able to compute the previously used group key. Thus, backward secrecy is provided. Analogously, for *forward secrecy* we have to show that any A being a leaving member in the Cover-Sub-Tree CT_a at node $\langle l_a, v_a \rangle$ for some $l_a \in \{0, \ldots, q\}$ is not able to obtain any subsequently used group key. A knows all secret keys $k^a_{\langle l,v \rangle}$ for all $l < l_a$ and $v = \lfloor \frac{v_a}{2^{l_a - l}} \rfloor$, and k_{CN_i} for all $i \leq a$ that are valid during its group membership. If A was not the only member in CT_a then the sponsor is located at node $\langle l_a - 1, \lfloor \frac{v_a}{2} \rfloor \rangle$ of the updated TFAN tree, otherwise it is the shallowest rightmost member in CT_{a+1}. In any case, since the sponsor changes own secret keys and causes the change of all secret keys in its path up to the root all secret keys that A knows get changed, and therefore A is not able to compute the updated group key. Thus, forward secrecy is provided. As combination of backward and forward secrecy we follow that TFAN provides *key independence*. Updated group keys are independent due to a random change of sponsor's contribution. Since both protocols, μSTR and μTGDH, fulfill the verifiable trust requirement our framework fulfills it too. Indeed, whenever a sponsor M_s located in the cs-tree CT_s at node $\langle l_s, v_s \rangle$ broadcasts a message containing the updated public keys, there is at least one other member that is able to verify the correctness of the sponsor's message. In case of TFAN ($art = S$) the verification can be done by all members in CT_s located at nodes $\langle l, v \rangle$ for all $l > l_s$. In case of TFAN ($art = T$) it can be done by a member located at the sponsor's sibling node. Additionally in both cases the changed public keys of cover-nodes CN_i for all $i \leq s$ can be verified by mem-

bers in every cs-tree CT_j with $j > i$.

10. Complexity Analysis

In this section we analyze the memory, communication, and computation costs of μSTR, μTGDH, and TFAN. The number of current group members, merging members, merging groups, and leaving members are denoted by: n, m, k, and p, respectively. Additionally, the height of the current and updated tree are denoted by h and $\widehat{h}$, respectively. The sponsor is denoted by $\langle l_s, v_s \rangle$ (or $\langle l_{s_i}, v_{s_i} \rangle$ if several sponsors exist). The sum of the heights of all non-highest trees in the merge protocol is denoted by α. We focus on the number of stored secret and public keys, the number of rounds, the total number of broadcast messages, the cumulative broadcast message size [4], and the serial number of multiplications [5]. We consider here random μTGDH trees and half fully filled TFAN trees, i.e., the number of members in each cs-tree is $\lceil \frac{q+1}{2} \rceil$ for $art = S$ and 2^{q-1} for $art = T$[6]. The serial number of multiplications can be further reduced if the sponsor computes the group key after it broadcasts the updated public keys. This optimization is considered in our analysis.

Memory costs Table 1 summarizes the memory costs of the protocols. Considering TFAN, it is clear that TFAN

Suite	Type of costs	Secret Keys	Public keys
	average	$\frac{n+3}{2}$	$2n-2$
μSTR	$\langle l, 1 \rangle$	$l+1$	$l+1$
	average	$\frac{n+3}{2}$	$\frac{n+1}{2}$
μTGDH	$\langle l, v \rangle$	$l+1$	$l+1$
	average	$\lceil \log_2 n \rceil + 1$	$\lceil \log_2 n \rceil + 1$
TFAN	$\langle l, 1 \rangle$ in CT_i	$i+l+3$	$i+l+3$
(art=S)	average	$\frac{q+\lceil \frac{n}{q+1} \rceil+5}{2}$	$\frac{q+\lceil \frac{n}{q+1} \rceil+5}{2}$
TFAN	$\langle l, v \rangle$ in CT_i	$i+l+2$	$i+l+2$
(art=T)	average	$\frac{2q+\lceil \frac{n}{2^q} \rceil+3}{2}$	$\frac{2q+\lceil \frac{n}{2^q} \rceil+3}{2}$

Table 1. Memory costs

($art = T$) consumes less storage space than TFAN ($art = S$). If we increase q, the member needs to store more keys and public keys in the cs-tree, however the number of keys and public keys outside the cs-tree is decreased. It is obvious that the required memory space for TFAN ($art = S$) is lower than for μSTR, and of TFAN ($art = T$) is higher than for μTGDH. Hence as a whole, all protocol suites can be sorted from the least to the highest according to their memory consumption as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR.

Communication and Computation Costs *Setup* Table 2 summarizes the computation and communication costs. μSTR and TFAN ($art = S$) are the most communication efficient protocol suites. Only 2 or 3 rounds are needed to form a new group from all individual group members. TFAN ($art = T$) is less efficient, i.e., $q + 2$ rounds are required. μTGDH is most inefficient with the number of rounds scaling logarithmically in the number of group members. According to the number of rounds the protocols can be sorted as follows: μSTR $<$ TFAN ($art = S$) $<$ TFAN ($art = T$) $<$ μTGDH. We consider now the number of messages and the total message size. With respect to the number of broadcast messages all protocols can be sorted from the least to the highest as follows: μSTR $<$ TFAN ($art = S$) $<$ TFAN ($art = T$) $<$ μTGDH. Considering the total message size, we get the following relation: μTGDH $<$ TFAN ($art = T$) $<$ μSTR $<$ TFAN ($art = S$). With respect to the number of serial multiplications μTGDH is the most computation efficient scaling logarithmically in the number of members, whereas μSTR requires a linear number of serial multiplications relative to the group size. The computation costs of TFAN in both cases depend on the cs-tree height q. In general TFAN ($art = T$) is more efficient than TFAN ($art = S$), which is in turn more efficient than μSTR. Hence all protocol suites can be sorted with respect to the number of required serial multiplications as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR. Considering the communication and computation costs TFAN ($art = T$) is most suitable to handle the setup event.

Join All protocol suites require 2 rounds and 2 messages. The total message size of STR is constant, only 3 public keys. In μTGDH it scales logarithmically in the number of group members. Since the shallowest cs-tree is usually not fully filled, the new member is joined in this cs-tree. In this case, the message size of TFAN ($art = S$) is 4, and of TFAN ($art = T$) is $q + 1$. Hence according to the required communication costs, the protocol suites can be sorted as follows: μSTR $<$ TFAN ($art = S$) $<$ TFAN ($art = T$) $<$ μTGDH. Under the assumption that the first cs-tree in TFAN is not fully filled, μTGDH is most expensive in terms of computation, its number of serial multiplications scales logarithmically in the number of group members. The number of serial multiplications in TFAN ($art = T$) is $3q + 1$, and in TFAN ($art = T$) is 7. To the contrary, μSTR re-

4 The sum of all broadcast message size of all members.

5 For serial costs operations that are performed by members in parallel are counted as one operation. In this case the highest costs are considered. The total costs are represented by the sum of all members costs in a given round (or protocol).

6 Hence the height is chosen as follows: $h \approx \lceil \log_2 n \rceil + 1$ for μTGDH, $h = \frac{n}{\lceil \frac{q+1}{2} \rceil}$ for TFAN ($art = S$), and $h = \lceil \frac{n}{2^{q-1}} \rceil$ for TFAN ($art = T$).

Protocol Suite	Protocol	Rounds	Beasts	Beast size	Serial muls
μSTR	Setup	2	$n+1$	$2n-2$	$3n-4$
	Join	2	2	3	4
	Leave	1	1	$\frac{n}{2}$	$\frac{3n}{2}-2$
	merge	2	$m+1$	$2n+k+1$	$3k+1$
	Partition	1	1	$\frac{n-p}{2}+1$	$\frac{3(n-p)}{2}+1$
	Refresh	1	1	$\frac{n}{2}+1$	$\frac{3n}{2}+1$
μTGDH	Setup	h	$n+h^2-h$	h^2-h+n	$2h-1$
	Join	2	2	$h+1$	$3h-2$
	Leave	1	1	h	$3h-2$
	Merge	$\lceil \log_2 m \rceil + 1$	$2m$	$h+\alpha+m\hat{h}$	$3(h+\hat{h})-4$
	Partition	$\min(\lceil \log_2 p \rceil + 1, \hat{h})$	$2min(2p, \lceil \frac{n}{2} \rceil)$	$(\hat{h}+1)\cdot\min(2p, \lceil \frac{n}{2} \rceil)$	$3h-2$
	Refresh	1	1	h	$3h-2$
TFAN ($art=S$)	Setup	3	$n+h+2$	$n+qh+q+h-1$	$2q+h-1+max(q,2h-2)$
	Join	2	2	4	7
	Leave	1	1	$\frac{q+h}{2}$	$\frac{3(q+h)}{2}+4$
	Merge	2	$m+1$	$h+2\alpha+m+2$	$3\alpha+4$
	Partition	2	$\hat{h}+1$	-	$6q$
	Refresh	1	1	$\frac{q+h}{2}+3$	$\frac{3(q+h}{2}+4$
TFAN ($art=T$)	Setup	$q+2$	$n+1+(q^2-q+1)(h+1)$	$n+h-1+(q^2+q+1)(h+1)$	$2q+3h-3$
	Join	2	2	$q+1$	$3q+1$
	Leave	1	1	$q+\frac{h}{2}+2$	$3q+3\frac{h}{2}+4$
	Merge	2	$m+1$	$h+2\alpha+m+q$	$3q+\alpha+1$
	Partition	$\frac{q}{2}+1$	$\frac{qq}{2}+1$	-	$6q$
	Refresh	1	1	$q+\frac{h}{2}+2$	$3q+\frac{3h}{2}+4$

Table 2. Computation and communication costs

quires only 4 multiplications. Hence according to the required computation costs all protocol suites can be sorted as follows: μSTR $<$ TFAN ($art = S$) $<$ TFAN ($art = T$) $<$ μTGDH. Considering the communication and computation costs μSTR is most suitable to handle the join event.

Leave, Refresh All protocols require 1 round and 1 broadcast message. μTGDH has the least cumulative message size, whereas μSTR the highest. TFAN ($art = S$) consumes more bandwidth than TFAN ($art = T$). In general the cumulative message size required by TFAN is between the sizes required by μSTR and μTGDH. According to the required computation costs, all protocol suites can be sorted as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR. With respect to the cumulative message size and serial multiplications all protocol suites can be sorted as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR. Obviously, μTGDH is most suitable to handle leave and refresh events.

Merge First we analyse the communication costs. Note that the number of rounds in μTGDH scales logarithmically in the number of merging groups, whereas all other protocols are more efficient keeping this number constant. According to the number of rounds all protocol suites can be sorted as follows: μSTR = TFAN ($art = T$) = TFAN ($art = S$) $<$ μTGDH. Furthermore, μTGDH requires double as much broadcast messages as in other protocol suites. Since the cumulative broadcast message size depends on the parameters *art* and q, it is difficult to give the exact relation between TFAN and other protocols. Considering the average cumulative message size all protocols can be sorted according to the message size as follows: TFAN ($art = T$) $\approx$ TFAN ($art = S$) $<$ μSTR and μTGDH, where the relation between μSTR and μTGDH depends on the number of merging groups and the tree structures. With respect to the computation costs all protocols can be sorted as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR. Hence, TFAN ($art = T$) is most suitable to handle the merge event.

Partition The μSTR and TFAN ($art = S$) protocol suites are most communication efficient, i.e., they require only one or two rounds. The partition is the most expensive operation in μTGDH, requiring a number of rounds bounded by the minimum of either the updated tree's height or $\log_2 p + 1$. The number of rounds in TFAN ($art = T$) is bounded by $\frac{q}{2}+1$. According to the number of rounds all protocol suites can be sorted as follows: μSTR $<$ TFAN ($art = S$) $<$ TFAN ($art = T$) $<$ μTGDH. With respect to the computation costs μTGDH requires a logarithmic number of multiplications, while μSTR scales linearly in the group size. TFAN is comparatively efficient, and less serial multiplications are needed if $art = T$. According to the computation costs the protocols can sorted as follows: μTGDH $<$ TFAN ($art = T$) $<$ TFAN ($art = S$) $<$ μSTR. Considering the communication and computation costs TFAN is most suitable to handle the partition event.

Discussion According to the analysis in [10] the μSTR protocol is most suitable in networks where high network delays dominate. However, its computation costs are most expensive. While μTGDH is most suitable in networks with clients having limited computation and storage space, and with low network delays. However, the clients in mobile ad hoc networks, in general, have limited processor power, storage capability, and energy supply. Additionally, the network delays dominate in such networks too. Considering the above described comparisons between the protocol suites in terms of computation and communication costs we remark that TFAN protocol suite is located in the mid-

dle of comparison relations for each kind of the subprotocol (i.e., setup, join, leave, merge, partition) whereas μSTR is mostly efficient in terms of the communication costs, and μTGDH is mostly efficient in terms of computation costs. Therefore, we follow that TFAN has the optimal trade-off between both kinds of costs.

11. Experimental Results

To compare the performance of μSTR, μTGDH, and TFAN protocol suites in practice we have implemented and simulated all six subprotocols for each suite. In this section we compare the measured computation and communication costs. The protocol suites are implemented with J2ME CDC [16], and simulated with the virtual machine *CDC HotSpot Implementation* [16] on the laptop with Centrino 1.7 GHz processor and 480 MB memory. We simulate the communication process and measure the total computation delay beginning with the time the event occurs and ending with the time the group key is computed.

Test method We use the EC curve P-192 defined in [13], whose security level corresponds to $|p| = 2720$ bits in $\mathbb{Z}_p^*$ according to [8]. The open source BouncyCastle API [17] for J2ME is used to implement the tree-based elliptic curve Diffie-Hellman key exchange method. We remark that cryptographic operations in the BouncyCastle API are not optimized, i.e., the computation delay we have measured can be significantly reduced if optimized implementation of the underlying cryptographic operations is used. Note that the measurements still allow to make a fair comparison of the considered protocol suites. In order to achieve a fair comparison of the plain protocol costs we do not measure the costs for message authentication. Finally, we have used randomly generated μTGDH trees and half fully filled TFAN trees in the simulation.

We use the following scenarios to measure the computation delay and the communication costs. For setup, join, leave and refresh, the number of current group members is $n = 4k$, $1 \leq k \leq 16$. In case of partition the group size before the event is $n = 16$ and 64. In case of merge the group size after the event is $n = 16$ and 64. In the following we denote TFAN with art and m by TFAN(art, m), e.g., TFAN(T, 2) and TFAN(S, 3), and consider here only the measurements of the computation delay and cumulative message size, since the number of rounds and messages is already given in Table 2. We have simulated TFAN with different combinations of art and q, and found out that for groups of 24 up to 64 members TFAN $(T, 3)$ and TFAN(S, 7) perform at best. The smaller is the value for q, the tighter is the approximation of the costs of TFAN by those of μSTR, whereas the greater is the value for q, the tighter is the approximation of the costs of TFAN by those of μSTR if $art = S$, and of μTGDH if $art = T$. In Figures 19 to 26 the y axis in the left graph denotes the computation delay in seconds, and in the right graph denotes the cumulative message size in kilobytes (KB).

Setup The costs are measured for the time period between the reception of all setup requests and the computation of the group key by all members. Figure 19 compares the computation delay and the total message size for the setup protocols of the suites. The x axis shows the group size. Note that the experimental results are similar to the estimated theoretical results from Section 10. Considering the computation delay μTGDH is most efficient, μSTR is least efficient, and TFAN costs are in the middle. We remark that if the parameter art is kept constant then the lower q is the higher computation delay occurs.

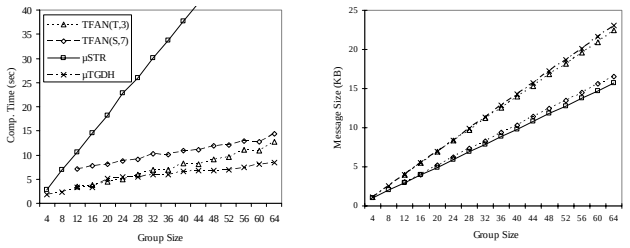

Figure 19. Cost Comparison for Setup

Join The costs are measured for the time period between the reception of the new member's request and the computation of the group key by all members (including the joined member). Figure 20 compares the computation delay and the total message size for the join protocols of the suites. The x axis shows the group size before the event. Note that the experimental results are similar to the estimated theoretical results from Section 10. However, in all protocol suites the message size scales linearly in the group size. This is because in every join protocol the sponsor broadcasts the whole tree structure.

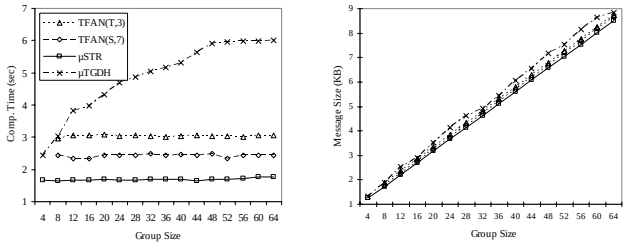

Figure 20. Cost Comparison for Join

Leave The costs are measured for the time period between the reception of the leave notification from the underlying

group communication system and the computation of the changed group key by all members. The leaving members are chosen from the group for each simulation step at random and the average costs are computed. Figure 21 com-

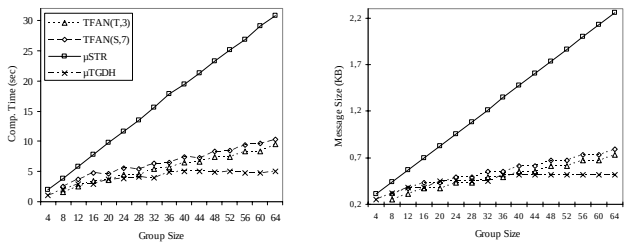

Figure 21. Cost Comparison for Leave

pares the computation delay and the total message size for the leave protocols of the suites. The x axis denotes the number of group members before the event. Note that the experimental results are similar to the estimated theoretical results from Section 10.

Merge We have measured the costs within the time period the group key is computed by all group members after all

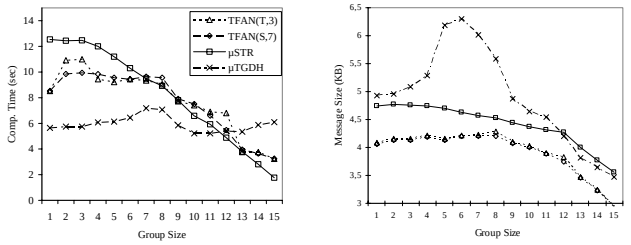

Figure 22. Cost Comparison for Merge (16)

members have been notified about the event. The number of

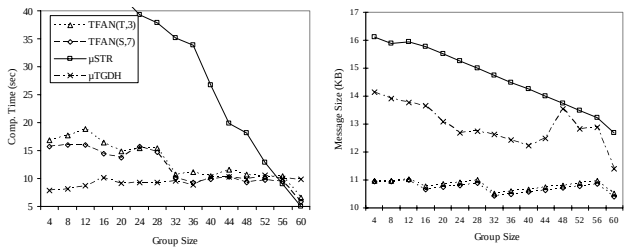

Figure 23. Cost Comparison for Merge (64)

resulting group members is 16 and 64. We assume the maximum number of merging groups is 5. Note that in practice merging of two groups is the most frequent case. Figures 22 and 23 compare the computation delay and the total message size for the merge protocols of the suites in case where the resulting group size is 16 and 64 members, respectively. In all graphs the x axis denotes the group size before the events. Note that the experimental results are similar to the estimated theoretical results from Section 10.

Partition We have measured the costs from the time point the partition event has occured to the time point when the new group key is computed by all members. The partitioned

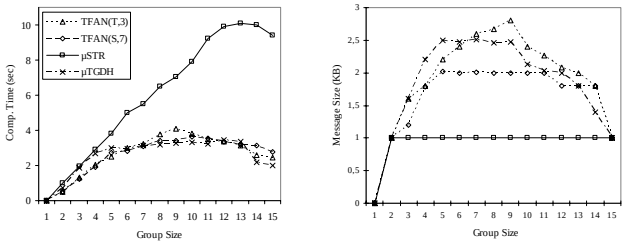

Figure 24. Cost Comparison for Partition (16)

members are chosen randomly from the group. Figures 24 and 25 compare the computation delay and the total message size for the partition protocols of the suites in case where the group size before partition is 16 and 64 members, respectively. In all graphs the x axis denotes the group size before the event. Note that the experimental results are

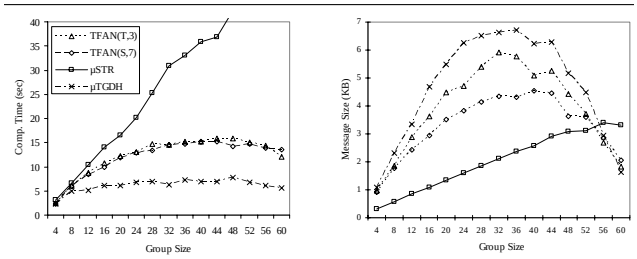

Figure 25. Cost Comparison for Partition (64)

similar to the estimated theoretical results from Section 10. That is the protocol suites can be sorted as follows μSTR $<$ TFAN($art = S$) $<$ TFAN($art = T$) $<$ μTGDH.

Refresh We have measured the costs from the time point the refresher begins to update its secret key to the time point the new group key is computed by all group members. The refresher is chosen from the group at random and the average costs of all members are computed for each simulation run. Figure 26 compares the computation delay and the total message size for the refresh protocols of the suites. The x axis denotes the group size. Note that the experi-

mental results are similar to the estimated theoretical results from Section 10 except for the difference that the costs of TFAN($T, 3$) and TFAN($S, 7$) are almost identical. This is because the average distance from the refresher's leaf node

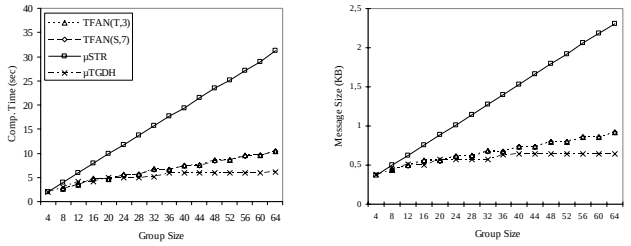

Figure 26. Cost Comparison for Refresh

to the root node of the cs-tree in both protocols are almost identical.

Discussion The above experiment results validate our theoretical analysis, i.e., TFAN provides an optimal trade-off between communication and computation costs compared to μSTR and μTGDH protocol suites. Considering our simulation[7] we suggest to use TFAN with the following parameters for the specified group size n, i.e., the group size before join, leave, and partition, but after merge: if $n < 24$ then TFAN($T, 2$) and TFAN($S, 7$) perform at best, whereas if $24 \leq n \leq 100$[8] then TFAN($T, 3$) and again TFAN($S, 7$) are most suitable.

12. Conclusion

In this paper we have proposed TFAN, a secure framework for the group key agreement in mobile ad-hoc networks which provides an optimal trade-off between computation, communication and memory costs as substantiated by the theoretical analysis and the experimental results obtained from the simulation. TFAN combines the communication efficiency of the μSTR protocol and the computation efficiency of the μTGDH protocol.

References

[1] N. Asokan and P. Ginzboorg. Key-agreement in ad-hoc networks. *Computer Communications*, 23(17):1627–1637, 2000.

[2] M. Burmester and Y. Desmedt. A secure and efficient conference key distribution system. In *Advances in Cryptology (EUROCRYPT '94),Lecture Notes in Computer Science*, volume 950, pages 275–286. Springer-Verlag Berlin, May 1994.

[3] S. Capkun, L. Buttyn, and J.-P. Hubaux. Self-organized public-key management for mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 2(1):52–64, 2003.

[4] A. E. Emmanuel Bresson, Olivier Chevassut and D. Pointcheval. Mutual authentication and group key agreement for low-power mobile devices. In *Proceedings of the 5th IFIP-TC6 International Conference on Mobile and Wireless Communications Networks (october 27 - 29, 2003, Singapore)*, pages 59–62. World Scientific Publishing, 2003.

[5] A. Joux and K. Nguyen. Separating decision diffie-hellman from diffie-hellman in cryptographic groups. Cryptology ePrint Archive, Report 2001/003, 2001. `http://eprint.iacr.org/`.

[6] Y. Kim, A. Perrig, and G. Tsudik. Communication-efficient group key agreement. In *Information Systems Security, Proc. of the 17th International Information Security Conference, IFIP SEC'01*, 2001.

[7] Y. Kim, A. Perrig, and G. Tsudik. Tree-based group key agreement. *ACM Transactions on Information and System Security*, 7(1):60–96, 2004.

[8] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001.

[9] J. Luo, P. T. Eugster, and J.-P. Hubaux. Route driven gossip: Probabilistic reliable multicast in ad hoc networks. In *INFOCOM*, 2003.

[10] M. Manulis. Contributory Group Key Agreement Protocols, Revisited for Mobile Ad-hoc Groups. In *Proceedings of 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS 2005), International Workshop on Wireless and Sensor Networks Security (WSNS 2005) (to appear)*. IEEE Computer Society, 2005.

[11] M. Manulis. Key agreement for heterogeneous mobile ad-hoc groups. In *Proceedings of 11th International Conference on Parallel and Distributed Systems (ICPADS 2005) Volume 2 International Workshop on Security in Networks and Distributed Systems (SNDS 2005)*, pages 290–294. IEEE Computer Society, 2005.

[12] U. M. Maurer and S. Wolf. The Diffie-Hellman protocol. *Designs, Codes and Cryptography*, 19:147–171, 2000.

[13] N. I. of Standards and Technology. Fips pubs 186-2: Digital signature standard (dss). Jan 2000.

[14] Standards for Efficient Cryptography Group (SEC). Sec 1: Elliptic curve cryptography, `http://www.sec.org`, September 2000.

[15] M. Steiner, G. Tsudik, and M. Waidner. Key agreement in dynamic peer groups. *IEEE Transactions on Parallel and Distributed Systems*, 11(8), 2000.

[16] SUN Microsystem. Connected Device Configuration (CDC) of J2ME; JSR 36, JSR 218. `http://java.sun.com/products/cdc/index.jsp`.

7 We have additionally tested TFAN with the following parameters for (art, q): $(T, 2)$, $(S, 3)$, $(T, 4)$, $(S, 15)$, $(T, 5)$, $(S, 31)$, $(T, 6)$, and $(S, 63)$.

8 In our analysis presented in this paper for convenience the maximal group size is chosen as 64. However, the performance of the implemented protocols remains acceptable for groups of up to 100 members.

[17] The Legion of the Bouncy Castle. Bouncy Castle Crypto APIs. http://www.bouncycastle.org/.