Fault-Tolerant Extension of Hypercube for Secure and Reliable Group Communications

Maria Striki, John S. Baras,
Institute for Systems Research
University of Maryland, College Park
MD, 20742, USA

Kyriakos Manousakis
Telcordia Technologies Inc.,
One Telcordia Dr, Piscataway
NJ, 08854, USA

Abstract

Securing group communications in resource constrained, infrastructure-less Mobile Ad Hoc Networks (MANETs) has become one of the most challenging research directions in the areas of wireless networking and security. MANETs are emerging as the desired environment for an increasing number of commercial and military applications, addressing also an increasing number of users. Security on the other hand, is becoming an indispensable requirement of our modern life for all these applications. The inherent limitations of MANETs impose major difficulties on establishing a suitable secure group communications framework. We contribute to the latter by extending Hypercube, an existing key agreement (KA) scheme - all parties contribute equally to the group key - to tolerate multiple member failures with low cost. We achieve this by enhancing Hypercube with a novel adaptively proactive algorithm. Members are assumed to be already authenticated via some underlying mechanism and we only focus on the design and analysis of a fault-tolerant scheme. Our algorithm has been evaluated and compared with the existing approach. Through our analysis and simulations we demonstrate its superiority in terms of robustness and efficiency.

1. Introduction.

A MANET is a collection of wireless mobile nodes, communicating among themselves over possibly multi-hop paths, without the help of any infrastructure such as base stations or access points. As the development of multicast services such as secure conferencing, visual broadcasts, military command and control grows, the research on security for wireless multicasting becomes increasingly important. The role of key management (KM) is to ensure that only valid members have access to a valid group key at any time. It is essential to develop a secure, robust KM scheme for group communications in these environments. The characteristics of MANETs constitute the major challenge for the design of suitable KM schemes. We deal with dynamic, infrastructure-less networks of limited bandwidth, unreliable channels where topology is changing fast. Network nodes have limited capacity, computational or transmission power. Connections are temporary (mobility changes, battery drainage, poor physical protection) and unreliable. These constraints turn most of the existing protocols inefficient in MANETs. Along with the continuous requirement to design more efficient KM schemes than existing, the need for these schemes to handle successfully and tolerate network dynamics and failures with low impact in a network with large number of nodes is now equally important.

Hypercube provides an orderly way of KA among nodes, and achieves minimization of the total number of rounds (latency) required, but does not exploit topological proximity of nodes. It ends up connecting any two members, possibly through a considerable number of relays. A pre-agreed schedule is used based on members' attributes, like their "hashed" *IP*s, or *ID*s dynamically assigned beforehand. In particular, for the 2^d members to agree on a group key, d rounds of key exchanges are required. In every round, each member selects a different peer to perform a key exchange with. However, not all of them can be in relative proximity to each other. The dynamic network characteristics result in frequent membership and mobility changes, with considerable overhead. The existing scheme does not anticipate disruptions (dynamic changes, failures) that occur all too often in MANETs. Upon failure(s) at any point during KA, Hypercube must start over: the underlying routing is invoked again, a significant amount of relays become involved in the group key exchange, and considerable delay (reflected in the total number of rounds for the successful protocol termination) burdens the network. Extending Hypercube to tolerate failures with low overhead becomes critical for its own feasibility and for this of protocols that depend on it i.e. Octopus schemes [1, 17].

This paper introduces a novel algorithm towards a fault-tolerant Hypercube, and provides comparative performance evaluation with the existing approach [1, 5]. Section 2 gives an overview of related work on robust KM. Section 3 gives an overview of the original scheme, and section 4 discusses how we define fault-tolerance for our framework. In section 5 we introduce our own *R*-Proactive algorithm, and in section 6 we sketch a proof of how Fault-Tolerance is achieved. In section 7 we discuss the significance of the algorithm. In section 8 we present the analysis of the *R*-Proactive *vs.* the *Basic* scheme with respect to certain metrics of interest and some indicative results of the comparative performance evaluation conducted, and in section 9 we present the simulations set-up and our results. Finally, in section 10 we conclude the paper.

2. Related Work

Proposals related to secure group communications abound in the literature, and can be classified in two main categories: contributory and centralized. Most of them address wire-line

978-0-7695-3106-9/08 $25.00 © 2008 IEEE
DOI 10.1109/ICN.2008.95

networks and cannot operate as such in MANETs. Contributory schemes require equal contribution from all members towards the generation of the group key and are useful when no single member is trusted or has the capabilities to generate and distribute the group key alone to the rest.

From the perspective of **contributory** protocols, Becker et al. [1], derived lower bounds for contributory key generation systems for the gossip problem and proved them realistic for Diffie-Hellman (DH) based protocols. They used the basic DH distribution extended to groups from the work of Steiner et al.[2]. From the latter work, GDH.2 is the most efficient representative of group DH protocols: it minimizes the total number of message exchanges. TGDH by Kim et al. [9], is a hybrid, efficient protocol that blends binary key trees with DH key exchanges. Becker in [1], introduced **Hypercube**, that requires the minimum number of rounds. In [5], Asokan et al. added an algorithm by which a node that finds that its chosen partner in a given round is faulty, selects other partners until a non-faulty one is found. Becker introduced Octopus scheme as one that requires minimum number messages and then derived the 2^d-Octopus that combined Octopus with Hypercube to a very efficient scheme that works for arbitrary number of nodes.

Centralized protocols are based on a simple key distribution center. The simplest and most fundamental representative is GKMP [8], in which a group leader shares a secret key with each member and uses it to communicate the secret group key to the associated member. LKH [7], creates a hierarchy of keys for each member. Each group member is secretly given one of the keys at the bottom of the hierarchy and can decrypt the keys along its path from the leaf to the root. Evolution of the latter are: ELK [18], designed rather for a stationary network, and OFT [6] that minimizes the number of bits broadcast to members after a membership change.

There exist some more recent proposals for wireless ad-hoc networks. Even these, do not seem to scale well or handle successfully the network dynamics. Some of these approaches rely on public key cryptography, which is very expensive for resource constrained nodes, or on threshold cryptography [12, 13, 14, 19], which results in high communication overhead, and does not scale well. A different approach is based on probabilistic key pre-distribution [15, 16, 23], which is a very lightweight method, designed for sensor networks, requires some basic infrastructure to handle mobility and membership changes (revocations). Amir et al. [10, 11], focus on robust KA, and attempt to make GDH protocols fault-tolerant to asynchronous network events. However, their scheme is designed for the Internet, and requires an underlying reliable group communication service and ordering of messages, so that preservation of virtual semantics is guaranteed. In [22], it is shown that the consideration of the physical location of members is critical for developing energy-efficient KM schemes, and based on this observation a new energy-aware KM scheme is proposed. In [17], additional Octopus protocols to provide robust and efficient KM for group communications in MANETs are proposed. The primary focus of this work is the evaluation of the proposed schemes, in isolation from network functions that interact with them (e.g. routing). Also, factors that greatly affect their overall performance, i.e. their poor reaction to disruptions, are not studied.

3. Original Hypercube Scheme (Overview)

Even though the Original Hypercube is well documented in our references [1, 5], we give a more detailed and simplified description of its basic operation in this section, in order to make it easier for the reader to follow the next sections.

Notation_1: Let $B(x) = a^x$ be the blinding of value x (exponentiation of x under base a) and $\varphi(x) = x \bmod n$. For simplicity, we often **replace** expression $\boldsymbol{B(X)}$ **with** $\boldsymbol{BX}$.

High level description: 2^d parties agree upon a key within d simple rounds by performing pair-wise DH key exchanges (DHKEs) on the edges of a logical d-dimensional cube. Every party is involved in exactly one DHKE per round. Each party uses the intermediate secret key K_{i-1} generated at the end of its DHKE in round (i-1), to compute the intermediate secret key K_i for round i: each party processes K_{i-1} and sends its peer in the clear the value $B(\varphi(K_{i-1}))$. The 2^d parties are identified on the d-dimensional space $GF(2)^d$ and a basis $b_1, \ldots, b_d \in GF(2)^d$ is selected to be used for the direction of communications per round: all subgroups must be engaged into communication exchanges at every round and no subgroup must be selected by two or more different subgroups during the same round. In every round, the parties communicate on a maximum number of parallel edges (round i, direction b_i). Members that acquire a common key from the previous round use the same "logical" direction of communications during the following round.

DHKE: The execution of a pair-wise DHKE allows two nodes to securely agree on the same secret key by exchanging contributions in the clear in the presence of "eavesdroppers". DH relies cryptographically on the assumption that it is computationally infeasible to solve the discrete logarithms problem (DLP). A DHKE for parties v, t with secret values r_v, r_t includes the following steps: they both blind their secret values and exchange them. In the end, parties v and t obtain (r_v, a^{r_t}) and (r_t, a^{r_v}), respectively. Party v raises its secret to the received value and generates the next secret: $K_v = (a^{r_t})^{r_v} = a^{r_v r_t}$. Similarly, party t generates $K_t = (a^{r_v})^{r_t} = K_v$.

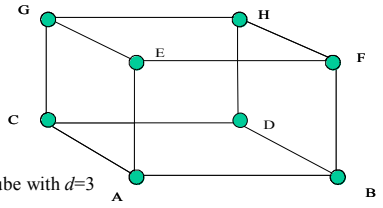

Fig 1: Hypercube with d=3

4. Requirements for Fault-Tolerant Hypercube

Our objective is to achieve fault-tolerance for Hypercube when it is run in the dynamic infrastructure-less environment of interest. We assume that the underlying routing is capable of establishing end to end paths, avoiding intermediate link failures. Under the same lines, we do not consider cases where the network is partitioned. We **mainly focus on member disruptions and failures** during group key establishment that frequently occur in our studied framework. The impact of such

failures either on the communication schedule or on the security properties of Hypercube is quite significant and has not been studied to date. Also, even a relatively low failure rate increases considerably the number of rounds it takes for the protocol to terminate. We wish to extend Hypercube to anticipate failures so that: *a*) the extra overhead and latency produced is maintained as low as possible, and *b*) the minimum amount of information gathered up to the point of failure is lost. Also, we want to preserve the following fundamental properties of the group key that escort secure KM protocols:

Forward (Backward) Secrecy: a passive adversary who knows a contiguous subset of old (new) group keys cannot discover subsequent (preceding) group keys.
Group key Secrecy: it is computationally infeasible for a passive adversary to discover any group key.

From this perspective, a member *k* that is no longer considered legitimate according to the group policy (misbehaving, evicted, faulty, disconnected) should not be able to compute the final group key, even if it still receives all the future blinded values exchanged. To better illustrate this with an example of eight nodes (scheme 1), assume that member *B* is evicted, and thus should be excluded from the final group key $K = a^{a^{ab}a^{cd}a^{a^{ef}a^{gh}}}$. Elements: $a^a, a^{a^{cd}}, a^{a^{a^{ef}a^{gh}}}$ are exchanged in the clear, and any member may get hold of them. It is then easy to see that *B* (contributed the initial secret value *b*) could compute *K* via successive exponentiations. We want to alter *K* in such a way that *B* is unable to reconstruct it, while most messages already exchanged are still useful (setting $K' = a^{a^{cd}a^{a^{ef}a^{gh}}}$ is a solution but excludes member *A* as well).

The extreme approach to exclude member *k* from the final group key is to start over Hypercube from scratch. A more improved version is to freeze Hypercube in round ***t*** during which the failure was observed, and restart it only among the "polluted parties", namely those that have so far interacted with member *k* in any of the previous rounds. That way, a 2^t-cube within the 2^d-cube is restored first, and then the normal execution resumes at round ***t***. This version, denoted as ***Basic***, still results in significant extra bandwidth and latency (rounds), particularly when failures occur near the final rounds.

5. R-Proactive Fault-Tolerant Hypercube

5.1. Properties and Objectives.

Hypercube was mainly used within the Octopus framework in previous work. Membership changes, failures and disruptions were always handled within Octopus subgroups, and the changes were just securely propagated through Hypercube. It was either assumed that no failures occur during the execution of Hypercube, because the participants were assumed to be powerful members with special capabilities (not true for any framework), or if failures were assumed, Hypercube would start from scratch. **R-Proactive** Hypercube is an adaptively proactive approach to anticipating and handling member failures at any time during group key establishment.

From the **security** perspective, our main concern is to maintain the property of Group Key Secrecy at the presence of disruptions. The properties of Forward/Backward Secrecy are ensured by other protocols that use them, i.e. Octopus that handles membership changes efficiently. Hence, we want to ensure that after a member is excluded from Hypercube at any time during group key establishment, it is unable to compute the group key. In terms of **efficiency**, we will explore ways to maintain the security of our scheme, without starting over the protocol from the 1st round whenever failures occur. Limiting the total number of rounds required for successful termination and reducing the associated costs are our main objectives. The less the number of rounds, the less the side-effects of routing on the standard communication are.

5.2 Overview.

R-Proactive consists of two stages: proactive (1st stage), and normal (2nd stage). The protocol still requires d rounds to terminate if no failures occur. The 1st stage extends until round R<d. The parameter *R* represents the level of "proactive-ness". It is dynamically adaptive and depends on the metrics we want to improve, and on the rate of failures observed: the higher the rate, the higher the value of *R* is set.

5.2.1. 1st Stage: Participants relay additional shares for the DHKEs that result in each member executing more than one DHKE with its peer per round, for the first ***R*** rounds. If one or multiple failures occur at this stage, Hypercube does not stall, and does not start over, but proceeds normally to the termination of this stage at round *R*. The multiple intermediate DH values computed by all members during this stage will be processed after round R. Members use the collected values to proactively compute multiple intermediate secret keys at this stage, each of which excludes one or more peers encountered during the previous rounds. Thus, if any of a member's peers fails, the member will select this key at round R from its available computed key pool that excludes the faulty peers from the key computations so far. Peers that fail before round R have not effect on R-Proactive other than determining which intermediate keys from the key pool formed at round R will be used and by which members. For peers that fail after round R, the protocol resumes from round *R* instead of 1, after the designated members select the proper keys from the key pool, and use them for the subsequent rounds.

5.2.2. 2nd stage: The algorithm switches to the original Hypercube. After members select the designated keys from the key pool, they use them for the subsequent rounds, and perform a single DHKE with their peers for the rest of the rounds.

5.3. Detailed Description.

In this section we describe in detail the R-Proactive algorithm for a general Hypercube of size *d* and show that it correctly terminates at the presence of failures. Because of the **symmetry** of the algorithm, it suffices to focus on DHKEs and computations of members that belong to any ***i*-cube** for each **round *i*.** For the communications schedule, we assign each member with a unique *id* from Z^*, i.e. (*A, B, C, D, E*,…)= (0,1,2,3,4,..). We select this base from the vector space so that

member j communicates with peer $k=\varphi(j \oplus 2^{i-1})$ at round i. So, peer A communicates with peer B at round 1, with C at round 2, with E at round 3, with I at round 4, etc.

Notation_1: Party A that initiates Hypercube with secret value a is denoted as $A(a)$. In this analysis, the modular reduction of a secret generated value x, namely $\varphi(x)$, prior to its blinding, $B(\varphi(x))$ is implicitly assumed, but not reflected to our equations, for ease of the notation. We denote the operation of raising each base value in the vector $X_n^T = (x_1, x_2, .., x_n)$ to each exponent included in the vector $Y_m^T = (y_1, y_2, \ldots, y_m)$ as:

$$X_n \times^{\wedge} Y_m^T = \begin{bmatrix} x_1^{y_1} & x_1^{y_2} & \ldots & x_1^{y_m} \\ x_2^{y_1} & x_2^{y_2} & \ldots & x_2^{y_m} \\ \ldots & \ldots & \ldots & \ldots \\ x_n^{y_1} & x_n^{y_2} & \ldots & x_n^{y_m} \end{bmatrix}.$$

5.4. Stage 1 (Proactive)

Round 1: Parties do one DHKE and store all values either generated or just received via their peer, in order to relay them all to the party they contact at the next round. At the end of this round parties $A - B$, and $C - D$ store the following values:

$A(a)$<->$B(b)$: $X_{AB}^T = (a^{ab}, a^a, a^b)$, $X_{1A}^T = (a^{ab}, a)$, $X_{1B}^T = (a^{ab}, b)$.

$C(c)$<->$D(d)$: $X_{CD}^T = (a^{cd}, a^c, a^d)$, $X_{1C}^T = (a^{cd}, c)$, $X_{1D}^T = (a^{cd}, d)$.

***Notation*2:** We denote the global vector including all initial **blinded keys (BKs)** of all nodes generated at the beginning of round 1 as $BX_{INIT}^T = (a^a, a^b, a^c, a^d, a^e, a^f, a^g, a^h, \ldots)$. Let vector $BX_{INIT}^T(i, j)$ include those elements of BX_{INIT}^T that become available to member j at the end of round i. For example, $BX_{INIT}^T(0, A) = (a^a)$, $BX_{INIT}^T(1, A) = (a^a, a^b)$, $BX_{INIT}^T(2, A) = (a^a, a^b, a^c, a^d)$, etc. as we will see shortly.

Round 2: A party sends the following values to its peer: *i*) the *BKs* computed at the end of the previous round and *ii*) the BKs computed and/or received at the beginning of the previous round. The following exchanges and computations take place for members A, B, C, D:

Message exchanges during round 2.

$C(a^{cd}, c)$ -> $A(a^{ab}, a)$: $XB_{CD}^T = (a^{a^{CD}}, a^c, a^d)$.

$A(a^{ab}, a)$ -> $C(a^{cd}, c)$: $XB_{AB}^T = (a^{a^{AB}}, a^a, a^b)$.

$D(a^{cd}, d)$ -> $B(a^{ab}, b)$: $XB_{CD}^T = (a^{a^{CD}}, a^c, a^d)$.

$B(a^{ab}, b)$ -> $D(a^{cd}, d)$: $XB_{AB}^T = (a^{a^{AB}}, a^a, a^b)$.

Computations during round 2.

***A* executes:** $X_{2A}^T = XB_{CD} \times^{\wedge} X_{1A}^T = (a^{a^{CD}}, a^c, a^d) \times^{\wedge} (a^{ab}, a) = (a^{a^{ab}a^{cd}}, a^{c \times a^{ab}}, a^{d \times a^{ab}}, a^{\alpha \times a^{cd}}, a^{ca}, a^{da})$.

***C* executes:** $X_{2C}^T = XB_{AB} \times^{\wedge} X_{1C}^T = (a^{a^{AB}}, a^a, a^b) \times^{\wedge} (a^{cd}, c) = (a^{a^{ab}a^{cd}}, a^{a \times a^{cd}}, a^{b \times a^{cd}}, a^{c \times a^{ab}}, a^{ca}, a^{bc})$

***B* executes:** $X_{2B}^T = XB_{CD} \times^{\wedge} X_{1B}^T = (a^{a^{CD}}, a^c, a^d) \times^{\wedge} (a^{ab}, b) = (a^{a^{ab}a^{cd}}, a^{c \times a^{ab}}, a^{d \times a^{ab}}, a^{b \times a^{cd}}, a^{cb}, a^{db})$.

***D* executes:** $X_{2D}^T = XB_{AB} \times^{\wedge} X_{1D}^T = (a^{a^{AB}}, a^a, a^b) \times^{\wedge} (a^{cd}, d) = (a^{a^{ab}a^{cd}}, a^{a \times a^{cd}}, a^{b \times a^{cd}}, a^{d \times a^{ab}}, a^{da}, a^{bd})$.

Similar is the process for parties E, F, G, and H.

***Notation*_3:** Let: $K = a^{a^{ab}a^{cd}}$, $L = a^{a^{ef}a^{gh}}$, $K(A) = a^{b \times a^{cd}}$, $K(B) = a^{\alpha \times a^{cd}}$, $K(C) = a^{d \times a^{ab}}$, ..., $L(E) = a^{f \times a^{gh}}$, $L(F) = a^{e \times a^{gh}}$, $K(BC) = a^{da}$, $K(AC) = a^{db}$, ..., $L(AG) = a^{fh}$, etc.

Round 3: A similar process is performed as in round 2, with the exception that only a limited number of the newly generated DH secrets are stored in the resulting vector X_3^T. Each member i blinds the values contained in X_{3i}^T to generate the vector XB_{3i}^T that will be communicated to its peer: $i \oplus 2^{j-1}$, where j represents the current round number. We denote vector SX^T as one containing a subset of the keys of vector X^T. Let $\tilde{K}_i^T$ denote the vector of values associated with the single value K, held by member i, and let $\tilde{L}_j^T$ denote the vector of values associated with the single value L, held by member j (K, L, as well as $K(AB)$, $L(GH)$ etc. computed at round 2).

Depending on the desired fault-tolerance level, we can control and limit the number of BKs communicated between peers in any given round. For example, the maximum number of BKs that can be sent from member E to A is provided below:

E->A: $XB_E^T = B(X_{2E}^T \cup X_{1E}^T \cup X_{0E}^T) = ((BL, BL(H), BL(G), BL(F), BL(FH), BL(FG)) \cup (a^{a^{ef}}, a^{a^{gh}}, a^g, a^h) \cup (a^e, a^f))$.

If member A combines all these BKs with its own secrets from the previous rounds, i.e. $X_{2A}^T \cup X_{1A}^T$, - and all the rest of members act similarly - , the newly generated BKs provide Hypercube with the capability to go around any subset of simultaneous failures instantly, without extra processing cost. The trade-off is a vastly growing number of keys, computed or received by each member during every round. However, we will show how members can tolerate any number of failures per round, proactively, using a relatively low number of keys per round. The exchanges and computations for a number of members in this round are demonstrated below:

Message Exchanges during round 3.

E* ->*A: $SXB_E^T = SB\, X_{2E}^T = (BL, BL(H), BL(G), BL(F), BL(FH), BL(FG), a^{a^{ef}} = BL(GH), a^{a^{gh}} = BL(EF))$.

A*->*E: $SXB_A^T = SB\,X_{2A}^T$ = (*BK*, *BK*(*D*), *BK*(*C*), *BK*(*B*), *BK*(*BD*), *BK*(*BC*), $a^{a^{ab}}$ = *BK*(*CD*), $a^{a^{cd}}$ = *BK*(*AB*)).

F*->*B: $SXB_F^T = SB\,X_{2F}^T$ = (*BL*, *BL*(*H*), *BL*(*G*), *BL*(*E*), *BL*(*EH*), *BL*(*EG*), $a^{a^{ef}}$ = *BL*(*GH*), $a^{a^{gh}}$ = *BL*(*EF*)).

B*->*F: $SXB_B^T = SB\,X_{2B}^T$ = (*BK*, *BK*(*D*), *BK*(*C*), *BK*(*A*), *BK*(*AD*), *BK*(*AC*), $a^{a^{ab}}$ = *BK*(*CD*), $a^{a^{cd}}$ = *BK*(*AB*)).

G*->*C: $SXB_G^T = SB\,X_{2G}^T$ = (*BL*, *BL*(*F*), *BL*(*E*), *BL*(*H*), *BL*(*FH*), *BL*(*EH*), $a^{a^{ef}}$ = *BL*(*GH*), $a^{a^{gh}}$ = *BL*(*EF*)).

C*->*G: $SXB_C^T = SB\,X_{2C}^T$ = (*BK*, *BK*(*B*), *BK*(*A*), *BK*(*D*), *BK*(*BD*), *BK*(*AD*), $a^{a^{ab}}$ = *BK*(*CD*), $a^{a^{cd}}$ = *BK*(*AB*)), etc.

The two peers may communicate to each other the vector of initial BKs available to them so far. For example, *A* may also send to *E* vector $BX_{INIT}^T(2, A)$ = (a^a, a^b, a^c, a^d) and vice versa. If more than one failure occurs after round *R*, then for each additional failure, the rest of members need to obtain the proper element from $BX_{INIT}^T(R, -)$. This vector can be constructed **proactively,** by having each member *k* send to its peer $BX_{INIT}^T(j-1, k)$ at every round *j*. This burdens the exchanges per member per round *j* with 2^{j-1} additional keys. The benefit is that in the event of member failures after round *R*, the rest of members already acquire the desired element from this vector and process it right away. The second option is to distribute this element to members **on a need to know basis**, after a failure. As we will see, there are always non-faulty members that hold this element.

Computations during round 3.

A executes: $X_{3A}^T = S(SXB_E \times\hat{}\ SX_{2A}^T) = S[S(BL \times\hat{}\ \tilde{K}_A^T), S(B\tilde{L}_E^T \times\hat{}\ K), S(B\tilde{L}_E^T \times\hat{}\ \tilde{K}_A^T)]$.

From these three vectors, member *A* only needs to include the following resulting keys in X_{3A}^T:

X_{3A}^T = (*BL*×^ *K*, *BL*×^ *K*(*B*), *BL*×^ *K*(*C*), *BL*×^ *K*(*D*), *BL*(*F*)×^ *K*, *BL*(*G*)×^ *K*, *BL*(*H*)×^ *K* , $a^{a^{ab}a^{ef}}$, $a^{a^{ab}a^{gh}}$).

Working in a similar fashion we get for member *E*:

X_{3E}^T = (*BK*×^ *L*, *BK*×^ *L*(*F*), *BK*×^ *L*(*G*), *BK*×^ *L*(*H*), *BK*(*B*)×^ *L*, *BK*(*C*)×^ *L*, *BK*(*D*)×^ *L*, $a^{a^{ab}a^{ef}}$, $a^{a^{cd}a^{ef}}$).

The following equality holds for any party *J*:

BL×^ *K*(*J*) = *BK*(*J*)×^ *L* (1).

This can be instantly proven since $(a^L)^{K(J)} = (a^{K(J)})^L$.

Considering (1), we see that vectors X_{3A}^T and X_{3E}^T differ only in the last element. The first element is actually the DH key produced at the end of the 3^d round as in the original scheme. If *d*=3, then all members agree upon this key at the end of the 3^d round, which contains equal contributions from all members. Observing the following six elements in both vectors, it can be seen that each one **excludes** the contribution of **only** one other member. The contributions (*a*, *e*) of members *A*, *E*, are naturally contained in all six elements (since if not, members *A* and *E* could always add their own secret contributions (*a*, *e*) themselves through one or more exponentiations to an element that contains neither of them). Members *A* and *E* interact at round 3 to compute secret keys that contain the contributions of all parties so far, or exclude the contribution of one only party (other than themselves) per element (secret key).

***Notation_4*:** Let *Y* = *BL*×^ *K* = $a^{a^{a^{ab}a^{cd}}a^{ef}a^{gh}}$, and then let *Y* (*B*) = *BL*×^ *K*(*B*) = $a^{a^{a\times a^{cd}}a^{ef}a^{gh}}$, etc.

We combine the first seven elements of vectors X_{3A}^T and X_{3E}^T in the vector $\tilde{Y}_{A,E}^T$, so that element $\boldsymbol{t} \in \tilde{Y}_{A,E}^T$ => $\boldsymbol{t}$ = *Y*(*j*), *j*∈ { ∅ , *B, C, D, F, G, H*}. So, we use the following notation for the computed keys of the 3^d round:

X_{3A}^T = ($\tilde{Y}_{A,E}^T$, $a^{a^{ab}a^{ef}}$, $a^{a^{ab}a^{gh}}$),

X_{3E}^T = ($\tilde{Y}_{A,E}^T$, $a^{a^{ab}a^{ef}}$, $a^{a^{cd}a^{ef}}$).

If we choose to construct the vector that contains the parties' initial blinded secrets $BX_{INIT}^T(R, -)$ proactively, then the vectors X_{3A}^T, X_{3E}^T contain a part of $BX_{INIT}^T(R, -)$. Hence:

{ $BX_{INIT}^T(3, A)$, $BX_{INIT}^T(3, E)$ }= $BX_{INIT}^T(2, A) + BX_{INIT}^T(2, E)$

As a consequence, the newly computed vectors for members *A*, *E* at the end of the 3^d round become:

X_{3A}^T = ($\tilde{Y}_{A,E}^T$, $BX_{INIT}^T(3, A)$, $a^{a^{ab}a^{ef}}$, $a^{a^{ab}a^{gh}}$),

X_{3E}^T = ($\tilde{Y}_{A,E}^T$, $BX_{INIT}^T(3, E)$, $a^{a^{ab}a^{ef}}$, $a^{a^{cd}a^{ef}}$).

By processing all keys for the rest of the members in a similar fashion, we get the following results for the 3^d round:

Reactive (on demand) Option

If *i*∈ {*A, B*}=> X_{3i}^T = ($\tilde{Y}_{i,i\otimes 4}^T$, $a^{a^{ab}a^{ef}}$, $a^{a^{ab}a^{gh}}$),

If *i*∈ {*C, D*}=> X_{3i}^T = ($\tilde{Y}_{i,i\otimes 4}^T$, $a^{a^{cd}a^{ef}}$, $a^{a^{cd}a^{gh}}$),

If *i*∈ {*E, F*}=> X_{3i}^T = ($\tilde{Y}_{i,i\otimes 4}^T$, $a^{a^{ab}a^{ef}}$, $a^{a^{cd}a^{ef}}$),

If *i*∈ {*G, H*}=> X_{3i}^T = ($\tilde{Y}_{i,i\otimes 4}^T$, $a^{a^{cd}a^{gh}}$, $a^{a^{ab}a^{gh}}$).

Scheme 2: Resulting DH keys for 8 parties at round *j*=3

At round *j* = 3, each party computes 2^j+1=9 keys as a result of 2^j+1 DHKEs with its peer (scheme 2). The last two elements in the vectors are used to handle certain cases of simultaneous failures instantly, at no extra cost. The same can be achieved with vector $\tilde{Y}_{i,i\otimes 4}^T$ alone as well, with little extra processing.

Proactive Option

At round j=3, each party computes 2^j+1=9 keys as a result of 2^j+1 DHKEs with its peer, as shown in scheme 3, and also includes in the secret vector generated at the end of the round, 2^j = 8 additional elements, that correspond to a subset of the blinded initial members' contributions contained in BX^T_{INIT}.

If $i \in \{A, B\} \Rightarrow X^T_{3i} = (\tilde{Y}^T_{i,i\otimes 4}, BX^T_{INIT}(3,i), a^{a^{ab}a^{ef}}, a^{a^{ab}a^{gh}})$

If $i \in \{C, D\} \Rightarrow X^T_{3i} = (\tilde{Y}^T_{i,i\otimes 4}, BX^T_{INIT}(3,i), a^{a^{cd}a^{ef}}, a^{a^{cd}a^{gh}})$

If $i \in \{E, F\} \Rightarrow X^T_{3i} = (\tilde{Y}^T_{i,i\otimes 4}, BX^T_{INIT}(3,i), a^{a^{ab}a^{ef}}, a^{a^{cd}a^{ef}})$

If $i \in \{G, H\} \Rightarrow X^T_{3i} = (\tilde{Y}^T_{i,i\otimes 4}, BX^T_{INIT}(3,i), a^{a^{cd}a^{gh}}, a^{a^{ab}a^{gh}})$

Scheme 3: Resulting DH keys for 8 parties at round j=3

Round 4-Round R: The process is similar to this of round 3 and is skipped here for lack of space.

5.5. Stage 2 (Normal) – Rounds [*R, d*].

Members switch to original Hypercube, using the designated values from the subsets (vectors X^T_R) generated at round R as secret values for the DHKE during round R+1. Every party is involved in exactly 1 DHKE per round. If no member failure occurs during stage 2, then all members agree on the same key at the end of round d. Upon failure(s), those members that have interacted directly or indirectly with the faulty one in any of the previous rounds, (i.e. have been "polluted"), must restart stage 2, by "going back" to round R and selecting (and processing) this element of X^T_R that correspond to the particular failure.

6. Fault Tolerance with R-Proactive Hypercube

6.1. Member(s) Failure prior to Round *R*.

Let us assume that member B fails during round k, while engaged in communication with party $T = B \oplus 2^{k-1}$. If T has received the blinded vector from B, it processes it exactly as indicated by R-Proactive, else it requests its own blinded vector from party $W=B \oplus 2^{k-2}$ (the party that paired with B at the previous round, i.e. if $T=F$ then $W=D$). Upon receiving a blinded vector either from B or W, member T processes it exactly as described in the R-Proactive algorithm. During the following rounds, all parties that would normally communicate with B, are now reconfigured to communicate with T (if T actually received the blinded vector from B, and B failed afterwards) or W (if B failed prior to sending its blinded vector to T, or if T fails as well) respectively, thereon. Should T and W also fail during the same round k or a subsequent one, the members that were supposed to contact them from this point and on, backtrack in rounds and are configured to communicate thereon with the party that was last in contact (most recent round) with any of these parties. This is the only effect that the failure of any member during stage 1 causes to the normal execution of R-Proactive, as we are going to show.

To illustrate this with an **example**, let $T = F$, and $W = D$ referring to the 3-cube example of eight nodes (scheme 1). Assume that B fails prior to contacting F. Then, F will get SBX^T_{2D} instead of SBX^T_{2B} from D. Member D has contacted B during the previous round, and generates a BK with elements equal to those that B has generated. This BK passes on to member F during the current round. Hence, the only difference observed is the following: in the end, member F will compute vector X^T_{3F}, which differs from what it would compute if B had not failed, only in that it includes $\tilde{Y}^T_{D,F}$ instead of $\tilde{Y}^T_{B,F}$. This means that: (*a*) F will have paired with D, which is correct and expected by the algorithm anyway, and (*b*) among all elements in vector $\tilde{Y}^T_{D,F}$ (held by F) there is one that excludes member B, (i.e. $L \times^{\wedge} K(B)$), but there is not an element excluding D if D also fails at some later point also. Similarly, all members that will be redirected to D instead of B, during the following rounds, will end up with legitimate final key vectors. Since at the final stage, no vector $\tilde{Y}^T_{B,F}$ is computed by any member, if B is the only "faulty" member, then it is excluded instantly from the final key, if the rest of the members pick the element $L \times^{\wedge} K(B)$ as their final key. Similarly, the failure of any additional member(s) i prior to round R, is (are) also reflected in the computations, and at the end of round R, no vector of type $\tilde{Y}^T_{i,j}$, for any of the remaining members j is created. Hence, all the remaining members contain an element in their vectors that excludes member B, one that excludes member i (i.e. $L \times^{\wedge} K(i)$ or $L(i) \times^{\wedge} K$), and in general, one that excludes one failed member at a time. All the elements that correspond to the **non-failed** members are multiplied to construct the **final key**. This way, faulty members are excluded from the final key altogether with a single action, and each of the remaining members can instantly compute the desired product, after round R is over as: $K = (L \times^{\wedge} K(B)) \times (L(i) \times^{\wedge} K)$.

6.2. One or Multiple Failures during round *R*.

Assume that B failed prior to round R. The rest of members select key $L \times^{\wedge} K(B)$ from vector $\tilde{Y}^T$. All members except for B, hold this key (B's failure was accommodated so that no vector $\tilde{Y}^T_{B,I}$ exists to deprive party I from $L \times^{\wedge} K(B)$). If member G fails also, its contribution must be excluded from the key that will be picked up after round R as well. Again, *G's* failure is accommodated in round R so that no vector $\tilde{Y}^T_{G,I}$ is computed by any party I. The key computed by all members except for B and G is: $K = (L \times^{\wedge} K(B)) \times (K \times^{\wedge} L(G))$, as before.

6.3. Member Failures after Round *R*.

Upon failure of any member(s), Hypercube is stalled at the current round k. The "polluted" members corresponding to

round R pick the proper new key(s) from $\tilde{Y}^T$ and use it (them) instantly, or process it (them) via modular exponentiations and multiplications as discussed next. Starting from round R+1, only the members that initially "polluted" the other half of the members at any given round (pollution propagation), in the sense that they communicated to those members values that included the contribution of the "faulty" member, need to update those parties with the new values (**Hypercube** is executed in **one direction** only), until round k is reached, and original Hypercube is executed once again from round k. The same process is repeated upon subsequent member failures.

Assume now that C fails. If the new key becomes $L\times^{\wedge} K(C)$, then member G that holds the vector $\tilde{Y}^T_{C,G}$ will not be able to compute the final group key, as element $L\times^{\wedge} K(C)$ is not included in vector $\tilde{Y}^T_{C,G}$. In this case, the final key at round R is not computed instantly as in the previous cases, but members need to execute one extra operation (ME) to compute it. The final key at round R now becomes: $K = a^{g(L\times^{\wedge} K(C))}$.

The BK a^g has been either already relayed to all members (if the proactive option is used for the propagation of the initial BKs vector), or it will be communicated to the members that request either from G or from members that have obtained this value from the first two rounds, if the reactive option has been adopted. Any member may send the required value $a^{L\times^{\wedge} K(C)}$ to G. Upon failure of another member e.g. D, the key selected by all the rest is: $K = (a^{g(L\times^{\wedge} K(C))})\times(a^{h(L\times^{\wedge} K(D))})$.

So far, we sketched how R-Proactive goes around all possible failure cases during execution, and how intermediate keys computed after each failure always exclude the "faulty" members without the need to start the execution from scratch.

7. Discussion

Considering that parties may not be directly connected, and may also be subject to abrupt dynamic changes, the impact of extra rounds in this framework is significant: routing increases the total communication, and the overall performance degrades (i.e. QoS deteriorates due to more collisions at the MAC layer, the bandwidth usage becomes higher and the same is the case for the consumption of network resources). Reducing the extra routing cost and number of rounds becomes our most important priority, even at the expense of extra computation and storage cost, which must still be kept low. R-Proactive is designed with these requirements in mind, and is also very flexible since we can adjust the level of "proactive-ness", by adaptively selecting the round R at which we switch from the *proactive* to the *normal* mode. This "proactive-ness" translates to the **trade-off** between the total number of **rounds** in the presence of failures, and the extra **communication** or **computation costs** incurred due to the multiple keys computed per round. R-Proactive can tolerate any number of failures. For a group of size O(1000), typical values for d (and consequently for R) do not exceed 13.

8. Analytical Evaluation of Basic vs. R-Proactive

8.1. Analysis and Comparative Analytical Evaluation

We now derive analytical formulae of the metrics of interest for both schemes. We also conduct a comparative performance evaluation, assuming that up to ***M* failures** occur during one execution ($M<2^d$). We evaluate the schemes for various distributions of these M failures over the d rounds of the typical Hypercube execution (examine worst, best, and average case scenarios that correspond to given distributions of M over the Hypercube rounds), and for multiple values of parameters R and d. The worst case scenario occurs if all M failures occur at the last round for both algorithms.

Notation5: Let K be the number of bits per message, C_E be the processing cost (bits) of a modular exponentiation (ME). By "RC" and "CM", we abbreviate the combined routing communication cost, and the computation cost respectively.

8.1.1. Basic: A failure before round j pollutes a 2^{j-1}-cube. It takes j-1 rounds to update it. During each round, only half of the parties are senders. All members do 2 MEs per round (compute current secret and blind it for the next round).
RC (no failures): $2^d \times d \times K$.

Round j failure: **RC**(j): $\sum_{i=1}^{J} 2^{J-1} \times K = 2^J \times K$, **CM**($j$): $2^{J+1} \times C_E$.

RC (with failures): $\sum_{i=1}^{d} (2^d - F(i)) \times K + \sum_{Failure=1}^{M(any\_round\_J)} 2^J \times K$.

8.1.2. *R*-Proactive
Stage 1: During this stage, each member blinds $2^{t-1}+2$ shares from round t-1, computes 2^t+2 new DH keys, and transmits 2^t BKs, under the proactive option at round t. The **total costs until round *R*, in the presence of failures** become:

Comp(*RProact1*):= $\sum_{i=1}^{R} (2^d - F(i)) \times (3 \times 2^{i-1} - 2) C_E$, and

Comm(*RProact1*): = $\sum_{i=1}^{R} (2^d - F(i)) \times (2^i - 1) K$,

$F(i)$: number of members that have failed up to round i. A failure at this stage affects insignificantly the previous costs.

Stage 2: Upon failure at round $j>R$, the "polluted" members run *Basic* from R, selecting another key from their vector pool.

RC: $\sum_{i=R+1}^{J} 2^{J-1} \times K = (2^J - 2^R) \times K$, **CM**: $((2^{J+1} - 2^{R+1}) + 2^R) \times C_E$.

R-Proactive succeeds in greatly reducing the total round number of Hypercube execution in the presence of failures, as well as the associated RC. The **novelty** is that Hypercube never has to stall for a failure that occurs at stage 1, and as for stage 2, the round number to remedy a failure is much less, since members only need to go back to round R. However, RC and CM are higher for R-Proactive, if measured in bits (the higher the level R, the higher the number of elements required to build the framework of stage 1). However, if we implement R-Proactive with ECDH, we achieve a substantial RC reduction. Nevertheless, the amount of rounds R-Proactive takes to

execute in the presence of failures is dramatically reduced compared to the original *Basic* scheme.

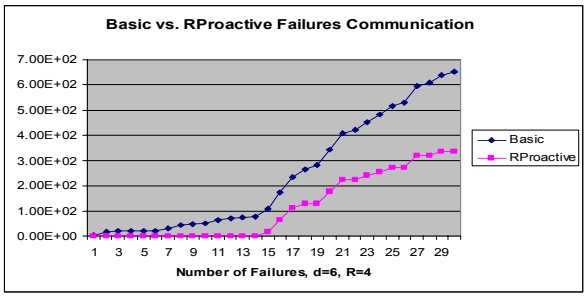

Fig 1: Total Communication due to Failures *Basic vs. R-Proactive*

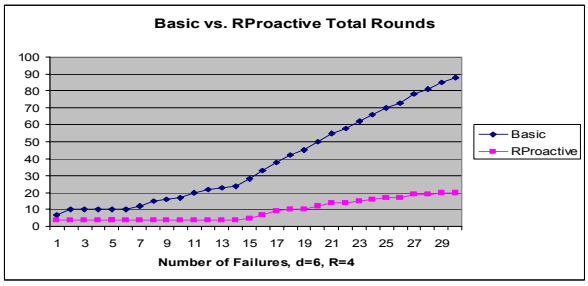

Fig 2: Total # Rounds-30 Failures: *Basic vs. RProactive*, d=6, R=4.

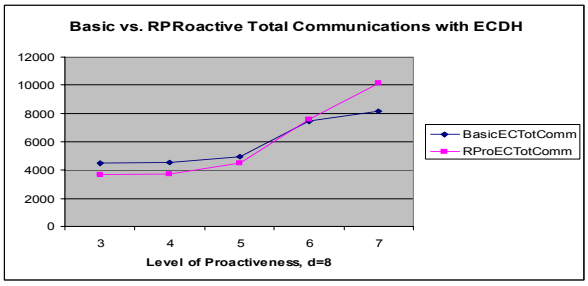

Fig 3: Total Communication *Basic vs. R-Proactive* implemented with ECDH, varying the Proactive-ness Level R

8.2. Elliptic Curve Crypto (ECC) Implementation

ECC is known to offer a security level comparable to that of other cryptographic systems of larger key sizes. If we substitute DHKEs with EC-DHKEs we achieve an obvious reduction in RC and storage costs (i.e. DH K=1024 *vs.* ECDH m=169 bit key sizes). We illustrate the benefit of ECs for R-Proactive with an arithmetic example. Consider an IP packet of 1500 bits, with a (head-tail) label of 128 bits, and available payload of 1372 bits. If we apply R-Proactive, setting R=4, a member must send a maximum of 18 elements to its peer during round R. If DH is used, then only 1 element fits in the IP packet, and hence 18 packets will be sent from a node to its peer. If ECDH is used, then $\lfloor {}^{1372}/_{169} \rfloor = 8$ elements fit in one IP packet, so partners need to exchange only 3 packets. So, R-Proactive becomes more powerful with the use of ECs [20, 21].

9. Basic *vs.* R-Proactive Simulation Results

We conducted simulations to compare RC from the execution of R-Proactive *vs.* Basic Hypercube over ad-hoc multi-hop networks. We use different graphs to generate the secure group. A number of nodes from this graph are randomly selected as subgroup leaders. We assume a generic Dijkstra routing protocol that finds the shortest paths between leaders. For our evaluation, we generated various random graphs of different sizes $S \in [100, 500]$ for each initial input of member number $n = 2^d$, and for the same graph we have varied parameter d: $2 < d \leq 8$. For the same graph and input, we have varied the group configuration, i.e., we have selected the n members at random in every repetition. For R-Proactive we additionally varied parameter R, where $1 < R \leq d$, for each graph and subgroup configuration, and for each d. Failures were simulated with the use of a probability failure parameter p, where $0.01 \leq p \leq 0.04$ for each member that is alive during a round. The probability of member failures is uniformly distributed in terms of rounds. For a given input $<S, d, R, p>$, we ran both algorithms with and without failures, for 100 different group configurations and we averaged the results.

We illustrate in the following graphs some indicative results produced by *Basic* and R-Proactive.

Fig1 shows a comparison in the RC overhead of the two schemes, for d=8, R=4, in the presence of failures, after the first group key establishment. The backbone framework for the schemes is not considered and we focus only on the steady state. Clearly, as the number of failures increase, R-Proactive presents an increasingly superior performance.

Fig2 shows a comparison in latency for d=6, R=4, by varying the number of failures before the protocols successfully terminate. R-Proactive presents an increasingly superior performance, and appears "impervious" to failures. This behavior compensates its relatively expensive framework.

Fig3 shows a comparison of the schemes in the total RC cost, when R-Proactive is implemented with ECs, for d=8, in the presence of failures. We see that both algorithms present comparable overall RC now.

Fig4 shows a comparison of the schemes for R=4, in terms of the total number of packets under increasing failure rates [0.01, 0.04] and group size [16,…,64]. When failure rates increase, i.e. p>0.03, the total number of packets decrease in *Basic*. This is because when the number of failed members is halved we re-adjust their schedule in *Basic* and decrease by one the round number. Still, R-Proactive is superior for almost all scenarios. It is not simple to do the re-adjustment discussed in *Basic*. Members must be assigned new *ids* and co-ordinate to apply a new schedule, which is impractical and expensive.

Fig 5 shows a comparison of the total packets transmission of *Basic vs. R-Proactive* (R in [1..7]) when no failures occur. Without failures, *Basic* always prevails, since *R-Proactive* consumes extra overhead to build the proactive framework and is not compensated if no failures occur, e.g., when R=7, d=7, the packet transmission overhead is offset a great deal.

Fig 6 shows a comparison of the schemes (for R in [3..7]), for a network of size S=200, and d=7, in terms of the total round number under failures, for two failure rates p=0.01, 0.02. R-Proactive achieves a significant reduction in the total number

of rounds, particularly as R increases. Again, we see that for p=0.02, the rounds for *Basic* are quite low. This is also due to the round re-adjustment. Even so, for R>4, *R-Proactive* achieves an even bigger reduction in the total round number.

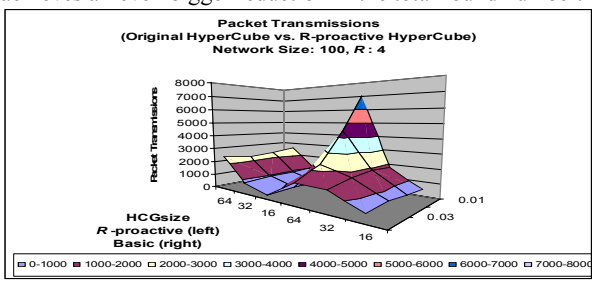

Fig 4: Total Packets Transmission Overhead for Basic *vs.* *R*-Proactive (R=4) w/ Failures, for failure rates [0.01,…,0.04]

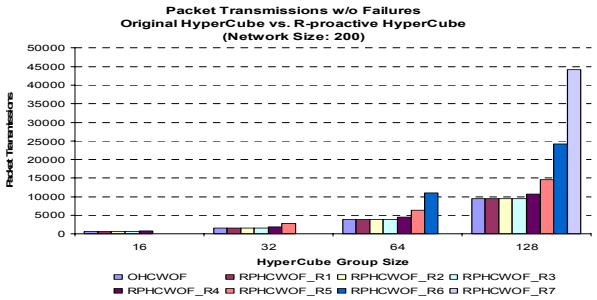

Fig 5: Total Packets Transmission OH for Basic and *R*-Proactive (R in [1..7]) Hypercube w/o failures, for d in [1..7]

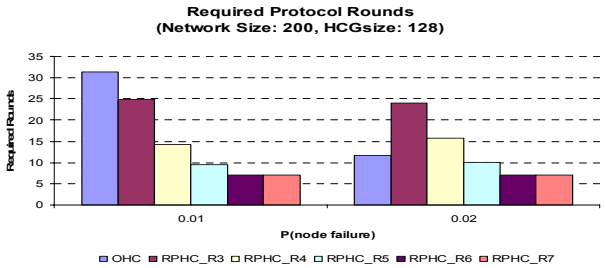

Fig 6: Total Round number for Basic and *R*-Proactive (R in [3..7]) w/ failures, for d =7, S=200, and p= 0.01, 0.02

10. Conclusion

This paper focuses on the design of a Fault-Tolerant Hypercube protocol in a MANET. We describe a hybrid proactive approach for extending Hypercube to tolerate failures suitable for the dynamic environment and for the application of interest. We present comparisons of two different schemes: the original Hypercube denoted as *Basic*, and our own Fault-Tolerant adaptively proactive extension, denoted as *R*-Proactive, in terms of communication, computation, storage cost, and total number of rounds. By exploring the features and evaluating the performance of the two algorithms, we show how we can achieve better performance with our scheme in the presence of failures. We also illustrate how the use of ECC favors our design and improves its overall performance.

11. References

[1] K. Becker, U. Wille, "Communication Complexity of Group Key Distribution," *Proc.5th ACM Conference on Computer & Communicatios Security*, pp. 1-6, San Francisco, CA, November 1998.

[2] M. Steiner, G. Tsudik, M. Waidner, "Diffie-Hellman Key Distribution Extended to Groups," *3rd ACM Conference on Computer & Communication Security*, pp. 31-37 ACM Press, 1996.

[3] M. Hietalahti. "Key Establishment in Ad-Hoc Networks," *M.S. Thesis*, Helsinki University of Technology, Dept. of Computer Science and Engineering, May 2001.

[4] A.Perrig, "Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication," *Int'l Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99),* pp. 192-202, July 1999.

[5] N.Asokan, P. Ginzboorg, "Key-Agreement in Ad-Hoc Networks," *Computer Communications*, Vol. 23, No. 18, pp. 1627-1637, 2000.

[6] D. McGrew, A.T. Sherman, "Key-Establishment in Large Dynamic Groups Using One-Way Function Trees," *IEEE Trans. On Software Engineering*, Vol 29, No. 5, pp. 444-458, 2003.

[7] H. Harney, E.Harder, "Logical Tree Hierarchy Protocol," *Internet Draft*, draft-harney-sparta-lkhp-sec-00.txt, Internet Eng. Task Force, March '99.

[8] H. Harney, C.Muckenhirn, "Group Key Management Protocol (GKMP) Specification/Architecture," *RFC 2093 and 2094*, Internet Engineering Task Force, July 1997.

[9] Y. Kim, A. Perrig, G. Tsudik, "Simple and Fault Tolerant Key Agreement for Dynamic Collaborative Groups," *Proc. 7th ACM Conference on Computer and Communication Security (CCS 2000),* pp. 235-244.

[10] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, G.Tsudik, "Exploring Robustness in Group Key Agreement", *Proc. of the 21th IEEE Int'l Conference on Distr. Computing Systems,* pp. 399-408, Phoenix, AZ, April 16-19, 2001.

[11] Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik, "Secure Group Communication Using Robust Contributory Key Agreement", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 15, no. 5, pp. 468-480, May '04.

[12] L.Zhou, Z.Haas, "Securing Adhoc Networks," *IEEE Network Magazine*, vol. 13, no.6, pp. 24-30, Nov/Dec 1999.

[13] J.Kong, P.Zerfos, H.Luo, S.Lu, L.Zhang, "Providing Robust and Ubiquitous Security Support for Wireless Ad-Hoc Networks," *Proc. 2001 IEEE Int'l Conf. on Network Protocols (ICNP 2001)*, pp. 251-260.

[14] S.Capkun, L.Buttyan, J.Hubaux, "Self-Organized Public Key Management for MANET," *IEEE Trans. on Mobile Computing*, Vol. 2, No. 1, pp. 52-64, Jan-Mar. 2003.

[15] L.Eschenauer, V.Gligor., "A Key Management Scheme for Distributed Sensor Networks," *Proc. 9th ACM Conference on Computer and Communication Security (CCS'02)*, pp. 41-47, Nov, 2002.

[16] H.Chan, A.Perrig, D.Song, "Random Key Predistribution Schemes for Sensor Networks," *Proc. 2003 IEEE Symposium on Security and Privacy*, pp. 197-213, May 2003.

[17] M.Striki, J.Baras "*Efficient Scalable Key Agreement Protocols for Secure Multicast Communication in MANETs*", Collaborative Technologies Alliance (CTA) Symposium, College Park MD, May 2003.

[18] A. Perrig, D.Song, D. Trygar, "ELK, a New Protocol for Efficient Large-Group Distribution," *Proc. 2001 IEEE Symposium on Security and Privacy*, pp. 247-262, Oakland, CA, May 2001.

[19] S.Yi, R.Kravets, "Key Management for Heterogeneous Ad hoc Wireless Networks," University of Illinois, Urbana-Champaign, CS dept., TR#UIUCDCS-R-2001-2241, UILU-ENG-2001-1748, July 2002.

[20] A. Hodjat, I.Verbauwhede, "The Energy Cost of Secrets in Ad-Hoc Networks", *IEEE Circuits and Systems (CAS) workshop on Wireless Communications and Networking*, Pasadena, CA, 2002.

[21] A.M.Fisiran, R.B.Lee, "Workload characterization of ECC and other network security algorithms for constrained environments*", IEEE Int'l Workload Characterization Workshop*, WWC-5, pp.127-137, Nov. 02.

[22] L.Lazos, R. Poovendran, "Energy-aware Secure Multicast Comm/tion in Ad-hoc networks Using Geographic Location Information", *IEEE Int'l Conf. of Acoustic Speech Signal Processing (ICASSP'03),* pp. 201-204, Hong Kong, China, April, 2003.

[23] S.Zhu, S.Setia, S.Xu, S.Jajodia, "GKMPAN: An Efficient Group Re-keying Scheme for Secure Multicast in Ad-hoc Networks", *IEEE Computer Society, MobiQuitous 2004,* pp. 45-51.