TECHNICAL RESEARCH REPORT

Fault-Tolerant Extension of Hypercube Algorithm for Efficient, Robust Group Communications in MANETs

by Maria Striki, John S. Baras

TR 2005-108

Fault-Tolerant Extension of Hypercube Algorithm for Efficient, Robust Group Communications in MANETs

Maria Striki, John S. Baras,
The Institute of Systems Research, Department of Electrical and Computer Engineering,
University of Maryland, College Park, MD, 20742

Abstract—**Securing multicast communications in 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. However, the limitations of the dynamic, infrastructure-less nature of MANETs impose major difficulties in establishing a secure framework suitable for group communications. The design of efficient key management (KM) schemes for MANET is of paramount importance, since the performance of the KM functions (key generation, entity authentication, key distribution/agreement) imposes an upper limit on the efficiency and scalability of the whole secure group communication system. In this work, we contribute towards efficient, robust and scalable, secure group communications for MANETs, by extending an existing key agreement (KA) scheme (where all parties contribute equally to group key generation) –Hypercube - to tolerate multiple member failures with low cost, through its integration with a novel adaptively proactive algorithm. We assume that the participating users have already been authenticated via some underlying mechanism and we focus on the design and analysis of a fault-tolerant Hypercube, with the aim to contribute to the robustness and efficiency of Octopus-based schemes (an efficient group of KA protocols for MANETs using Hypercube as backbone). We compare our algorithm with the existing approach, and we evaluate the results of our analysis. Through our analysis and simulation results we demonstrate how the new Hypercube algorithm enhances the robustness of the Octopus schemes maintaining their feasibility in MANETs at the same time.**

Index Terms—**Key Management, Key Agreement, Hypercube Protocol, Fault-Tolerance, Octopus Schemes, Elliptic Curves Cryptography**

I. INTRODUCTION

A Mobile Ad Hoc Network (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 wireless multicast services such as cable TV, secure audio and 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 or can be reached and updated with a valid group key at any time. It is essential to develop a secure, robust KM scheme for multicast communications in MANETs. However, the characteristics of MANETs constitute the major constraint and challenge for the design of suitable KM schemes. We are dealing with dynamic, infrastructure-less networks of limited bandwidth, unreliable channels where topology is changing fast. Nodes within the network may have limited capacity, computational and 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: among other requirements, they need to catch up with the rapidly changing network topology, and deal with failures at any time during group key establishment.

Along with the requirement to design secure KM schemes that achieve better performance than existing ones (either for wire-line or wireless networks), the need for the KM schemes to handle successfully and tolerate with low impact network dynamics and failures (robustness) in a network with large number of nodes (scalability) is now equally important.

In an attempt to meet all these objectives, two novel hybrid KA Octopus schemes have been previously introduced and evaluated [6,19] in addition to the original Octopus (O) [1]. Hierarchy is supported through the partition of a large key KA group to 2^d subgroups of smaller size. Initially, each subgroup agrees on its own subgroup key, and elects a subgroup leader to handle KA locally within its own subgroup. Then, the subgroup leaders alone, representing their subgroups, interact among themselves and use the previously generated subgroup keys to agree on a global group key via Hypercube. Finally, the leaders distribute securely the group key to their subgroup. The following features of Octopus schemes have motivated interest to explore and extend them: *a*) their hierarchical framework through which they can tolerate network dynamics and be more scalable, *b*) a subgroup leader handling a relatively small subgroup is a feasible option in MANETs, as opposed to a single leader in centralized schemes handling a very large group, *c*) disruptions are handled locally at first within the subgroup, and then reflected to the whole group via low cost re-keying over Hypercube.

Hypercube constitutes the backbone or the core of Octopus schemes. It provides an orderly way of key agreement among nodes, in such a way that minimization of the total number of rounds required for the agreement, and therefore of the total latency of the process, is achieved. The protocols applied within each subgroup, even though they are not fault-tolerant overall in general, they can be made to tolerate failures with low overhead. The latter is also greatly attributed to the limited subgroup size and topological proximity of the subgroup members [6, 20]. This is not the case however with the application of Hypercube, and that is where one of the main vulnerabilities of Octopus schemes lies. Topological proximity cannot be assumed for the Hypercube parties, which could end up connecting any two subgroups, even if they are located in different areas in the network, possibly through a considerable number of relay nodes. Obviously, the underlying routing is required for these nodes to agree on a group key, resulting in significant extra communication overhead every time it is invoked. The basic principles of Hypercube mechanism do not exploit topological proximity of members for the communication exchanges required towards the generation of the group key. Rather, a pre-agreed schedule is used, based on some attributes of subgroup leaders, like their "normalized/hashed" *IP* addresses, or some simple *ID*s that have been allocated dynamically beforehand to them in order to facilitate the generation of this schedule. Furthermore, for 2^d subgroup leaders to agree on a group key via Hypercube, *d* rounds of key exchanges are required. In every round each leader selects a different peer to perform a key exchange. Thus, since each leader is supposed to connect with at least *d* other leaders, it is expected that not all of them can be in relative proximity to each other anyway. The dynamic characteristics of the network, that bring it up against frequent membership and mobility changes, also attest to the later. The current Hypercube version, efficient as it may be w.r.t. KM messaging and number of rounds, does not anticipate disruptions (dynamic changes, failures etc.), that occur all too often in MANETs. Upon failure at any point during KA, Hypercube must start over (from the 1st round), burdening the network with significant overhead and latency. Extending Hypercube to tolerate failures with minimal overhead becomes critical for its own feasibility in MANETs, and for the feasibility of Octopus schemes in MANETs as well.

This paper introduces a novel algorithm towards a fault-tolerant Hypercube version included in Octopus-based schemes, and provides comparative performance evaluation with the existing "reactive" approach [1,5]. Section 2 gives an overview of related work on robust KM. Section 3 gives an overview of the original simple scheme, and section 4 discusses how we define fault-tolerance for our framework. In section 5 we introduce and describe our *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 *R*-Proactive *vs.* a *Basic* scheme w.r.t. 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.

II. Related Work

There exist several KM proposals for secure group communications in the literature. Most of them address wire-line networks and cannot operate as such in MANETs. From the perspective of **contributory** protocols (those in which equal member contributions are required for the generation of the group key), 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 their 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. [10], 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 to Hypercube 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. This kind of fault-tolerance, although useful, **is only a subset** of our own **requirements** in terms of **fault-tolerance**. Becker introduced Octopus scheme as one that requires minimum number of total 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 (non-contributory) protocols are based on a simple key distribution center (single point of failure, particularly in a MANET). The simplest and most fundamental representative is Group Key Management Protocol GKMP [9], 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. The original Octopus uses a GKMP version within each subgroup. Logical Key Hierarchy protocol LKH [8], 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: the Large-Group Key Distribution protocol ELK [21], designed rather for a stationary network, and OFT [7] that minimizes the number of bits broadcast to members after a membership change. The number of keys broadcast in this case, and the computational efforts of the members are logarithmic in the number of members.

There exist some more recent proposals for wireless ad-hoc networks. Even these schemes, 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 [14, 15, 16,22], which results in high communication overhead, does not scale well, and presents security vulnerabilities, mainly due to the mobility of nodes. A different approach is based on probabilistic key pre-distribution [17, 18], which is a very lightweight method, designed for sensor networks, but has serious security vulnerabilities, and requires some basic infrastructure to handle mobility and membership changes (revocations). Y.Amir et al. [12,13], 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 also requires an underlying reliable group communication service and ordering of messages, so that preservation of virtual semantics is guaranteed. Poovendran et al. [11], attempt to minimize the communication overhead w.r.t. to energy expenditure, for secure group communications. They utilize centralized tree key distribution schemes. The network topology is considered static, and there is no provision for adjusting the key tree structure to a dynamically changing network. It is shown that the optimal solution of their formulation does not scale with group size.

In [6,19,20], Octopus-based KA protocols have been designed to provide robust and efficient KM for group communications in MANETs. The primary focus of this work has been the analysis and performance evaluation of the proposed schemes, in isolation from network functions that interact with the protocols (e.g. underlying routing). Also, factors that greatly affect the overall performance of these schemes, like the poor reaction of Hypercube to disruptions, have been ignored. Although some of the protocols appear promising, their evaluation under these assumptions and simplifications is not totally "fair". Consideration of the above in the analysis and re-evaluation of those KA schemes, will shed more insight to their actual feasibility in MANETs, and will provide more realistic results.

III. 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 understand the sections that follow.

Notation_1: Let $B(x) = a^x$ be the blinding of value x (exponentiation of x under base a) and $\varphi(x) = x \bmod n$.

High level description: Hypercube is designed with the objective to minimize the total number of simple rounds required for the KA. In general, 2^d parties agree upon a key within d simple rounds by performing pair-wise Diffie-Hellman key exchanges (DHKEs) on the edges of a logical d-dimensional cube. Every member is involved in exactly one DHKE per round. 2-party DHKEs constitute the basic module of Hypercube. Each member uses the intermediate secret key K_{i-1} generated at the end of its 2-party DHKE in round (i-1), in order to compute through the current DHKE the intermediate secret key K_i for round i. For the execution of the 2-party DHKE at round i, each member processes K_{i-1} and sends its peer in the clear the value $B(\varphi(K_{i-1}))$.

Notation_2: Let $S_X(j)$ be the subset X of members that obtain the same intermediate secret key K_X after round j.

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

Hypercube Function: The 2^d participants are identified on the d-dimensional space $GF(2)^d$ and a basis $b_1,\dots, b_d \in GF(2)^d$ is selected to be used for the direction of communications per round. In every round, the parties communicate on a maximum number of parallel edges (round i, direction b_i). For instance, during round j, party i could interact with party $i \oplus 2^{j-1}$. During round i, each party v selects peer $v+b_i$ (different peer in every round) to perform a DHKE with. The selection of 2^{d-1} pairs per round, is based on a pre-agreed schedule established among parties before the start of Hypercube. As we will show next by induction, all parties share a common key at the end of the protocol.

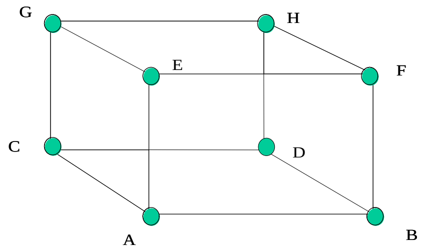
***1st round**:* All parties have agreed on the DH parameters that will be used (blinding operator a, modulo n). Every party $v \in GF(2)^d$ generates a random number r_v and performs a pair-wise DHKE with peer $t = v+b_1$ using the generated values r_v and r_t, respectively.

Any two members $A \in S_A(1)$, $B \in S_B(1)$, agree on the same key $K(S_{AB}(2))$ = DHKE($K(S_A(1))$, $K(S_B(1))$). Similarly, members $C \in S_C(1)$, $D \in S_D(1)$, agree through a DHKE on the same key $K(S_{CD}(2))$ = DHKE($K(S_C(1))$, $K(S_D(1))$). So, before round 2, members have teamed in subgroups of two, with a common subgroup key.

***2nd round**:* Assume that according to the selected vector base b_2 member A agrees on a key K_{AC} with member C and B agrees on another key K_{BD} with D. After the end of the DHKEs of the 2nd round, for member A, C we obtain the secret key: K_{AC} = DHKE($K(S_{AB}(2))$, $K(S_{CD}(2))$), for members B, D we obtain the key: K_{BD} = DHKE($K(S_{AB}(2))$, $K(S_{CD}(2))$).

Since $K_{AC} = K_{BD}$, all four members have computed the same secret key and belong to the same 2-cube subgroup, i.e. $S_X(4) = S_{ABCD}(4)$. If a similar schedule is applied for all the rest of members, 2^{d-2} subgroups with common intermediate secret keys of size four are generated. This is the end of the initial inductive step.

Round *i* (inductive step i)**:** Assume that after the end of round (i-1), $2^{d-(i-1)}$ subgroups $S_j(i-1)$, $1 \le j \le 2^{d-(i-1)}$) of size 2^{i-1} are formed. Assume that all members in $S_j(i-1)$ acquire a common intermediate subgroup key generated after the end of DHKEs at round (i-1). The new pairs for round i will be selected in such a way that: $\forall x \in S_j(i-1)$, a peer $y \in S_t(i-1)$, $1 \le j, t \le 2^{d-(i-1)}$), must be selected. At the end of round i, after the DHKEs between all pairs from the two different subgroups, a new subgroup of double size is generated, denoted as $S_b(i)$, $1 \le b \le 2^{d-(i-2)}$). Now, all members $z \in S_b(i)$ (or $z \in (S_j(i-1) \cup S_t(i-1))$ equivalently) obtain a new common subgroup key $K(S_b(i))$ = DHKE($K(S_j(i-1))$, $K(S_t(i-1))$). Therefore, at round i, members of one subgroup j must communicate each with a different peer from another common logical (i-1)-cube t. In other words, members that acquire a common key from the previous round should use the same “logical” direction of communications during the following round. In order to coordinate communication direction for all i-subgroups or i-cubes for all rounds i, so that all subgroups are engaged into communication exchanges at every round and no subgroup is selected simultaneously by two or more different subgroups during the same round, the need to select a basis in $GF(2)^d$, as stated earlier, arises. This step completes our induction, and hence we have shown that after round d all members agree on the same group key.



Scheme1: Hypercube with d=3

IV. 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. Member disruptions and failures during group key establishment in Hypercube occur often in our studied framework. As already discussed, the impact of such failures on Hypercube is quite significant, particularly because topological proximity cannot be assumed for its participating members: the underlying routing protocol invoked and relays required increase the overall overhead incurred to the network even further. In the original Hypercube, member failures during group key establishment have not been anticipated neither from the perspective of the schedule of communication among members nor from the perspective of the security of the key generation algorithm as such. Whenever a failure occurs, almost everything must be re-configured and Hypercube starts from scratch (1st round). Even a relatively low failure rate increases considerably the number of rounds it takes the protocol to terminate (and consequently its latency), and offsets the total overhead incurred in the network. We would like to extend Hypercube to anticipate and handle member failures efficiently, in such a way that a) the extra communication, computation cost and latency incurred in the network is maintained as low as possible, and *b*) the minimum amount of the information that has been gathered up to the point of failure is lost.

At the same time we want to preserve the following fundamental security properties with respect to the group key, that escort all secure KM protocols: *Forward*, *Backward* and *Group key secrecy*.

Forward Secrecy: a passive adversary who knows a contiguous subset of old group keys cannot discover subsequent group keys.
Backward Secrecy: a passive adversary who knows a contiguous subset of group keys cannot discover 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) at any round during group key establishment must become unable to compute the final group key, even if it still receives all the blinded values of the rest of the parties (resumes from failure, re-connects to the group).

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 $K = a^{a^{a^{ab}a^{cd}}a^{a^{ef}a^{gh}}}$. The elements: $a^{a}, a^{a^{cd}}, a^{a^{a^{ef}a^{gh}}}$ are exchanged in the clear among parties, and any member, including *B* may get hold of them. It is then easy to see that *B* that has 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 all the other members still contribute to it and most of the information already exchanged still remains useful (setting $K' = a^{a^{cd}a^{a^{ef}a^{gh}}}$ could be a solution - it excludes however 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 Hypercube 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 2^{d}-cube execution resumes at round ***t***. However, this reactive algorithm, denoted as ***Basic***, results still in significant extra communication cost and number of rounds, particularly in cases where the node failure occurs near the final execution rounds.R-Proactive Fault-Tolerant Hypercube.

A. Properties and Objectives.

So far, Hypercube has been mainly used within the Octopus framework in previous work. Membership changes (additions, evictions), failures and disruptions were always handled within the Octopus subgroups (1st, 3d steps in Octopus), 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 (subgroup leaders) were assumed to be powerful, trusted members with special capabilities (not generally true for the framework we consider), or if failures were assumed, Hypercube would start from scratch, or a version of Basic algorithm would be applied, with considerable overhead however (reactive approach). The R-Proactive Hypercube constitutes a hybrid adaptively proactive approach to anticipating and handling member failures in Hypercube 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 bound to the membership changes are ensured by Octopus, that handles membership changes very efficiently. Hence, we want to ensure that after a Hypercube member is excluded from the protocol for any reason at any time during group key establishment, it is unable to compute the group key, even if it knows a contiguous subset of intermediate Hypercube keys and it has become a passive adversary. We want to avoid the expensive solution of starting over Hypercube with new initial information, in order to maintain its security.

In terms of efficiency, we want to explore ways to maintain the security of our scheme, without the need to start over the protocol from the 1st round whenever failures occur. Starting over the protocol, results in considerable routing and communication overhead, as well as considerable latency. Limiting the total number of rounds required for Hypercube to terminate at the presence of failures and reducing the combined routing and communication costs are our main objectives, since we decrease both the latency of the protocol and the corresponding routing and communication costs further. The less the number of rounds, the less often the underlying routing must be invoked, and the less the side-effects of routing on the standard communication of Hypercube (i.e. number of relays involved, discovery of new routing paths and maintenance of existing ones).

B. Overview.

R-Proactive is a hybrid proactive algorithm that consists of two different stages, the proactive (1st stage), and the normal (2nd stage). The protocol still takes *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" incorporated. It is dynamically adaptive and depends on the metrics we want to improve, and on the rate of failures observed in the network. The higher the rate, the higher *R* should be set.

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 that are 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 other members they have interacted with during the previous rounds. Thus, if any of a member's peers fails, the member will select this key at round *R* from the available key pool it has computed, that excludes the faulty peers from the subgroup key computations so far. Peers that fail before round *R* have not effect on the *R*-Proactive protocol other than determining which intermediate keys from the key pool formed at round *R* will be used and by which members for the subsequent rounds. For peers that fail after round *R*, the protocol resumes from round *R* instead of round 1, after the designated members select the appropriate keys from the key pool, and use them for the subsequent rounds.

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.

Next, we describe our algorithm in detail, we show that *a*) it correctly terminates at the presence of failures (all participants until the final round and only they compute the same secure group key), and *b*) it satisfies the expected security and fault-tolerance properties.

C. Detailed Description.

In this section we are going to describe in detail the *R*-Proactive algorithm for a general Hypercube of size *d*. Because of the symmetry of members' participation to the algorithm (as already seen from the description of the original protocol), it suffices to focus on the DHKEs and computations of members that belong to a particular *i*-cube for each round *i*. For the schedule of communications, we allocate to each member a unique successive integer *id* from Z^*, i.e. (*A*, *B*, *C*, *D*, *E*,…) = (0,1,2,3,4,..). Furthermore, we select this base from the vector space $GF(2)^d$ s.t. member *j* communicates with peer $k = \varphi(j \oplus 2^{i-1})$ at round *i*. For example, member *A* communicates with peer *B* at round 1, with peer *C* at round 2, with *E* at round 3, with *I* at round 4, etc.

Notation_1: A party *A* that will initiate Hypercube on its part with the secret value *a* is denoted as *A*(*a*). In the following 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. Furthermore, we denote the operation of raising each base value included in the vector $X_n^T = (x_1, x_2, .., x_n)$ to each exponent value included in the vector $Y_m^T = (y_1, y_2, \ldots, y_m)$ as:

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

A. Stage 1 (Proactive)

Round 1.

Parties execute one DHKE and store all values either generated or just received via their peer party, in order to relay them all to the party they contact at the following round. At the end of this round parties *A* – *B*, and *C* – *D* for example 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 comprising **all blinded initial secrets** 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 communicates the following values to its peer: *i*) the blinded DH value computed at the end of the previous round (normal execution), and *ii*) the blinded secret values computed and/or received at the beginning of the previous round. The following exchanges and computations take place for members *A*, *B*, *C*, *D*:

A. Message exchanges during round 2.

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

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

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

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

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: For ease of the notation, we denote: $K = a^{a^{ab}a^{cd}}$, $L = a^{a^{ef}a^{gh}}$, $K(A) = a^{b \times a^{cd}}$ (i.e. the contribution of all members towards *K*, except for this of member *A*), $K(B) = a^{\alpha \times a^{cd}}$, $K(C) = a^{d \times a^{ab}}$, …., $L(E) = a^{f \times a^{gh}}$, $L(F) = a^{e \times a^{gh}}$ (i.e. the contribution of all members towards *L*, except for this of member *F*), $K(BC) = a^{da}$, $K(AC) = a^{db}$, and so on and so forth.

Round 3 - Round *R*.

Round3: 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 the stored X_{3i}^{T} to generate the vector XB_{3i}^{T} that will be communicated to the designated peer member, namely member $i \oplus 2^{j-1}$, where *j* represents the current round number. We denote vector SX^{T} as one containing a subset of those keys that are comprised in vector X^{T}. Also, 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 during round 2).

Depending on the level and nature of fault-tolerance we wish the extended Hypercube protocol to achieve, we can control and limit the number of blinded keys a member is going to communicate to its peer during any given round. The **maximum number of available blinded keys** that can be sent from member *E* to *A* for example 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 blinded keys 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 number of keys provides Hypercube with the capability to go around any subset of simultaneous member failures instantly, without any extra processing cost. The trade-off for this capability however, is a vastly growing number of keys (secret or blinded), computed or received by each member during every round, and the algorithm becomes impractical and infeasible after the first few rounds.

In this work, 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:

A. 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*:** $SXB_C^T = SB\,X_{2C}^T = (BK, BK(B), BK(A), BK(D), BK(BD), BK(AD), a^{a^{ab}} = BK(CD), a^{a^{cd}} = BK(AB))$, and so on and so forth.

Furthermore, the two peers may communicate to each other, the vector of blinded initial secrets available to them so far. For example, *A* may also communicate to *E* vector $BX_{INIT}^T(2, A) = (a^a, a^b, a^c, a^d)$ and vice versa. As we will see later, if more than one failure occurs after round *R*, then for each additional failure, the rest of the members need to obtain this among all elements of vector $BX_{INIT}^T(R,-)$ that *R*-Proactive designates. Vector $BX_{INIT}^T(R,-)$ may be constructed **proactively,** by having each member *k* communicating to its peer party the vector $BX_{INIT}^T(j-1, k)$ during every round *j*. This option burdens the communication exchanges per member per round *j* with 2^{j-1} additional keys. The great benefit of this option is that in the event of member failures after round *R*, all the rest of the members already acquire the desired element from $BX_{INIT}^T(R,-)$, and process it right away, issuing no extra latency to the network. On the other hand, the second option is to distribute the designated element to the Hypercube members **on a need to know basis**, after the failure of a member. As we will see, there are always members (among those that have not become "faulty") that do acquire the designated element and are able to distribute it to the rest. This option spares the communication of a considerable number of keys. Any of these two options may be adopted, depending on our network requirements and on which metrics we are mostly interested in improving. We consider both to evaluate our algorithm.

B. Computations during round 3.

After a party receives the vector of the blinded keys, it computes the DH values of the current round:

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

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

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

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

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

The following equality holds for any party *J*:

$BL \times^\wedge K(J) = BK(J) \times^\wedge L$ (1).

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

Considering (1), it is easy to 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 if the original, typical Hypercube is applied. 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* and *E* respectively, 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 either 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 $Y = BL \times^\wedge K = a^{a^{a^{ab}a^{cd}}a^{a^{ef}a^{gh}}}$, and let $Y(B) = BL \times^\wedge K(B) = a^{a^{a \times a^{cd}}a^{a^{ef}a^{gh}}}$, 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 $\boldsymbol{t} \in \tilde{Y}_{A,E}^T \Rightarrow \boldsymbol{t} = Y(j)$, $j \in \{\emptyset, B, C, D, F, G, H\}$. 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 on the other hand, we prefer to construct the vector that contains the participants initial blinded secrets $BX_{INIT}^T(R,-)$ proactively, then the vectors X_{3A}^T, X_{3E}^T contain a part of $BX_{INIT}^T(R,-)$ as well. Therefore:

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

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

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

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

Processing the blinded and secret keys for the rest of the members in a similar fashion, we get the following results for the rest of the parties during the 3^d round:

A. Reactive (on demand) Option

$$\begin{aligned}
&\text{If } i \in \{A, B\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, a^{a^{ab}a^{ef}}, a^{a^{ab}a^{gh}}),\\
&\text{If } i \in \{C, D\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, a^{a^{cd}a^{ef}}, a^{a^{cd}a^{gh}}),\\
&\text{If } i \in \{E, F\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, a^{a^{ab}a^{ef}}, a^{a^{cd}a^{ef}}),\\
&\text{If } i \in \{G, H\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, a^{a^{cd}a^{gh}}, a^{a^{ab}a^{gh}}).
\end{aligned}$$

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

To sum up, at round j = 3, each party computes 2^j+1=9 keys as a result of 2^j+1 DHKEs with its peer party, as shown in scheme 2. The last two elements in the vectors are somehow redundant. They 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 some extra processing however: one blinding and/or a multiplication as we will see next. Again, this extra functionality is another flexibility of the algorithm.

B. Proactive Option

$$\begin{aligned}
&\text{If } i \in \{A, B\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, BX_{INIT}^{T}(3,i), a^{a^{ab}a^{ef}}, a^{a^{ab}a^{gh}})\\
&\text{If } i \in \{C, D\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, BX_{INIT}^{T}(3,i), a^{a^{cd}a^{ef}}, a^{a^{cd}a^{gh}})\\
&\text{If } i \in \{E, F\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, BX_{INIT}^{T}(3,i), a^{a^{ab}a^{ef}}, a^{a^{cd}a^{ef}})\\
&\text{If } i \in \{G, H\} \Rightarrow X_{3i}^{T} = (\tilde{Y}_{i,i\otimes 4}^{T}, BX_{INIT}^{T}(3,i), a^{a^{cd}a^{gh}}, a^{a^{ab}a^{gh}})
\end{aligned}$$

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

To sum up, at round j=3, each party computes 2^j+1=9 keys as a result of 2^j+1 DHKEs with its peer party, 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 the vector BX_{INIT}^{T}. Therefore, vector X_{3i}^{T} comprises 2^{j+1}+1 = 17 elements in total, in this case. The last two elements serve the same purpose as this mentioned previously.

Round 4 – Round R:

The process is exactly similar to this of round 3, and we may skip its description. At the end of round j = 4 for example, vector X_{4i}^{T} contains:

a) $\tilde{Y}_{i,i\oplus 8}^{T}$ with cardinality $|\tilde{Y}_{i,i\oplus 8}^{T}|$ = (2^j+1-2)= 2^j-1, (one key that corresponds to the original Hypercube DH key, and 2^j-2 keys, each of which excludes the contribution of one only among all members , except for the two ones generating $\tilde{Y}_{i,i\oplus 8}^{T}$),

b) vector $BX_{INIT}^{T}(4,i)$ that contains a subset of the initial blinded members contributions, with cardinality 2^j, only in the proactive option, otherwise this vector is not a subset of X_{4i}^{T} for the reactive option, and

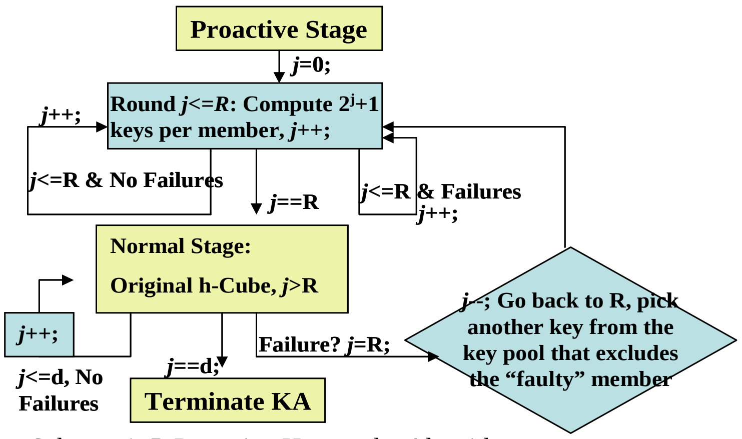
c) two additional DH keys that 1) discriminate member i from its peer member, and 2) contribute to the instant generation of a common key among all parties, at no extra cost, for some cases of simultaneous member failures.

B. Stage 2 (Normal) - Rounds [R, d].

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

V. Fault Tolerance with R-Proactive Hypercube

R-Proactive algorithm theoretically attains a very high level of fault-tolerance, i.e. number of (simultaneous) member failures at any point during execution.



Scheme 4: R-Proactive Hypercube Algorithm

We distinguish among the following scenarios and issues of Fault-Tolerance:

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

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 from party $W=B \oplus 2^{k-2}$ (the party that communicated with *B* at the previous round, i.e. if *T*=*F* then *W*=*D*) its own blinded vector. Upon receiving a blinded vector either from member *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 terms of rounds, and are configured to communicate thereon with the party that was last in contact (most recent round) with any of these parties. This, actually, 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 now.

To illustrate the latter with an example, let *T* = *F*, and *W* = *D* referring to the 3-cube placement example of eight nodes (scheme 1). Assume that *B* fails prior to communication with *F*. Then, *F* will get SBX^{T}_{2D} instead of SBX^{T}_{2B} from *D*. Member *D* has communicated with member *B* during the previous round, so that it generates a blinded key with elements equal to those that *B* has generated and would pass on to member *F* during the current round. Therefore, 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 otherwise (if *B* had not failed) only in that it includes $\tilde{Y}^{T}_{D,F}$ instead of $\tilde{Y}^{T}_{B,F}$. That means that member *F* will have paired with *D*, which is correct, and anticipated by the algorithm anyway. Also, it means that in the vector $\tilde{Y}^{T}_{D,F}$ (held by member *F*) there exists among all elements one that excludes member *B*, (i.e. $L \times^{\wedge} K(B)$), but there is not an element excluding *D* in the case that *D* also fails at some later point. Similarly, all members that will be redirected to *D* instead of *B*, during the following rounds, will end up with legitimate (according to *R*-Proactive) 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 member *B* 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, that is present in all other vectors, as discussed.

Similarly, the failure of any additional member(s) *i* prior to round *R*, is (are) also reflected to the computations, and in the end of round *R*, no vector of the form $\tilde{Y}^{T}_{i,j}$, for any of the remaining members *j* is created. Therefore, all the remaining members contain an element in their vectors that excludes member *B*, an element that excludes member *i* (i.e. $L \times^{\wedge} K(i)$ or $L(i) \times^{\wedge} K$), and in general, an element excluding one failed member at a time. All the elements that correspond to the 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 furthermore, each of the remaining members can instantly compute the desired product, as soon as round *R* is over. In the latter case, members compute:
$\boldsymbol{K} = (L \times^{\wedge} K(B)) \times (L(i) \times^{\wedge} K)$.

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

Assume that member *B* has failed prior to round *R*. The remaining members select the key $L \times^{\wedge} K(B)$ from their vector $\tilde{Y}^{T}$, as discussed. All members but *B* possess this key now (the failure of *B* has been accommodated so that no vector $\tilde{Y}^{T}_{B,I}$ exists to deprive party *I* from $L \times^{\wedge} K(B)$). Now, if member *G* fails as well, its contribution must also be excluded from the key that will be picked up after round *R*. Again, the failure of *G* is accommodated during round *R* so that no vector $\tilde{Y}^{T}_{G,I}$ is computed by another party *I*. In that case, the key computed from all members but *B* and *G* is:
$\boldsymbol{K} = (L \times^{\wedge} K(B)) \times (K \times^{\wedge} L(G))$, as in the previous case.

C. *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 appropriate new key(s) from $\tilde{Y}^{T}$ and use it (them) instantly, or process it (them) via modular exponentiations and multiplications as we will see right 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 the original Hypercube protocol is executed once again from round k. The same process is repeated upon subsequent member failures.

Assume now that member C fails in our example. If the new key becomes $L\times^{\wedge}K(C)$, then member G that holds the vector $\tilde{Y}_{C,G}^{T}$ will not be able to compute the final group key, since the element $L\times^{\wedge}K(C)$ is not included in the vector $\tilde{Y}_{C,G}^{T}$. 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 (modular exponentiation) to compute it. The final key at round R now becomes: $\boldsymbol{K} = a^{g(L\times^{\wedge}K(C))}$.

The blinded value a^g has been either already relayed to all members (if the proactive option is used for the computation and propagation of the initial vector of blinded keys), or it will be communicated to the members that request through unicast messages (2^R-4 in the worst case) or broadcasts from member G or from members that have obtained this value from the first two rounds, if the reactive option has been adopted. Furthermore, any member (probably one that is in topological proximity to member G) can communicate the requested value $a^{L\times^{\wedge}K(C)}$ to G.

Upon failure of another member e.g. D, the designated key selected by all the remaining members this time, is: $\boldsymbol{K} = (a^{g(L\times^{\wedge}K(C))})\times(a^{h(L\times^{\wedge}K(D))})$.

Through this analysis, we have explored all possible cases of failure during the execution of the Hypercube. We have discussed and sketched how R-Proactive algorithm handles and remedies all the above cases of failure. We have shown that the intermediate keys computed after each failure, always exclude the "faulty" members without need to start over the execution of Hypercube protocol from scratch.

VI. Discussion

The R-Proactive algorithm provides a hybrid proactive approach towards handling failures at any round of the Hypercube execution. Considering that the Hypercube parties in our case are most probably not directly connected, and are also subject to abrupt dynamic changes, it is easy to see that the impact of extra rounds in this framework is very significant: the underlying routing protocol that is invoked, increases the total communication cost of the KM framework, excessive relay nodes may be required, 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 node/network resources). Therefore, reducing the extra communication cost and the total number of rounds in the extended Fault-Tolerant scheme becomes our most important priority, even at the expense of extra computation and storage cost, which still need to be kept low however.

The R-Proactive algorithm 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 mode" to the "normal mode" of execution. This "proactive-ness" translates to the **trade-off** between the number of the **total rounds** in the presence of failures (or equally the overall **latency, and extra routing costs**) on one hand, and the extra **KM Communication and Computation costs** incurred to the network, due to the multiple keys that must be computed per round, on the other hand.

The R-Proactive algorithm can tolerate any number of failures, and this is a very useful feature for any framework upon which it is applied. As far as its application on the Octopus-based schemes, the probability of a large number of failures during a typical Hypercube execution varies depending on the characteristics of the selected subgroup leaders. Assuming that robust, powerful leaders, with extra capabilities, can be guaranteed at all times for any given subgroup is not very realistic in a dynamic infrastructure-less network. It is quite likely that simple, resource-constrained members undertake more often than not the role of subgroup leaders, increasing the probability of failures significantly, and consequently the probability that the Hypercube is stalled multiple times. In this case, an algorithm such as R-Proactive that handles failures with low cost, and more particularly, keeps the overall number of rounds in the presence of failures very low, becomes if paramount importance.

Moreover, in this work we assume that we are dealing with a KM group of size O(100) - O(1000) at maximum. In the case that we are interested in applying the Hypercube protocol over all group members, typical values for the Hypercube parameter d do not exceed the number 15 (since 2^{15} = 32768). In the case that we wish to apply Hypercube over the subgroup leaders of a KM group, typical values for d are even lower, and we assume they do not exceed the number 10. The value of d sets an upper limit to the value R of R-Proactive.

VII. Analytical Evaluation of Basic *vs.* R-Proactive Hypercube Schemes

i) *Analysis and Comparative Analytical Evaluation:*

In this section, we derive and present a number of significant analytical formulae of the metrics of interest for both *Basic* and the *R*-Proactive algorithms (i.e. overall Communication, Computation, Storage Costs and number of rounds). We also conduct a comparative performance evaluation of the two schemes, 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 the worst, best, average, and random case scenarios that correspond to particular 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 take place during the last round for both algorithms.

Notation: Let K be the number of bits per message, and let parameter C_E represent the computation cost of a modular exponentiation in bits approximately.

A. Basic.

A failure before round j pollutes a 2^{j-1}-cube. It takes j-1 rounds to update it. During each round, half of the parties are only senders and half only receive the updated values. All members however, carry out two exponentiations per round, one to compute the current secret, and one to blind it for the subsequent round. Hence, upon a failure during round j, the total KM **Communication cost** (in bits) is:

Comm(*Basic*):=$\sum_{i=1}^{J} 2^{J-1} \times K = 2^J \times K.$

Similarly, the **Computation cost (in bits)** upon a member failure during round j is:

Comp(*Basic*):=$2^{J+1}\times C_E,$

The **Storage cost** per member, if the member stores the blinded values computed at all d rounds becomes:

Store(*Basic*):=$d\times K.$

The total communication cost without failures is: **CommNF**(*Basic*):=$2^d\times d\times K.$

In the presence of failures, the **total communication cost**, until the protocol terminates by successfully computing the final key becomes:

CommF(*Basic*)=$\sum_{i=1}^{d}(2^d - F(i))\times K + \sum_{Failure=1}^{M(any\_round\_J)} 2^J \times K.$

B. 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 blinded keys, under the proactive option assumption, at round t. The total computation and communication 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*): = $\sum_{i=1}^{R}(2^d - F(i))\times(2^i - 1)K$,

Here the parameter $\boldsymbol{F}(i)$ represents the number of members that have failed up to round $\boldsymbol{i}$. It has to be noted that failure of a member during this 1st stage affects insignificantly the previous costs. The maximum per member storage cost becomes:

Store(*RProact1*):= $2^{R+1}\times K.$

Stage 2: Upon failure at round $j>R$, the "polluted" members execute *Basic* starting from round R, after selecting another key from their vector pool (1-2 exponentiations per member are required at max. to process the new key). The KM **Communication cost** is:

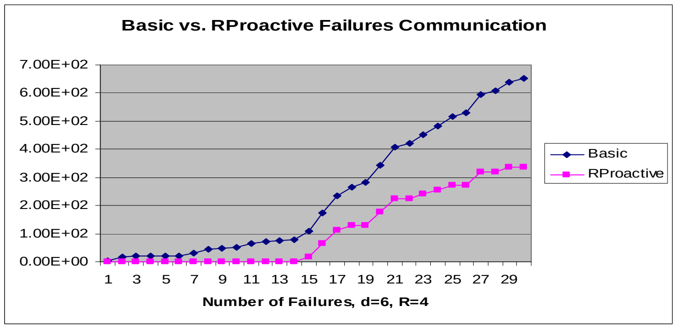
Comm(*RProact2*):= $\sum_{i=R+1}^{J} 2^{J-1}\times K = (2^J - 2^R)\times K,$

and the KM **Computation cost** working in a similar fashion becomes for the worst case scenario:

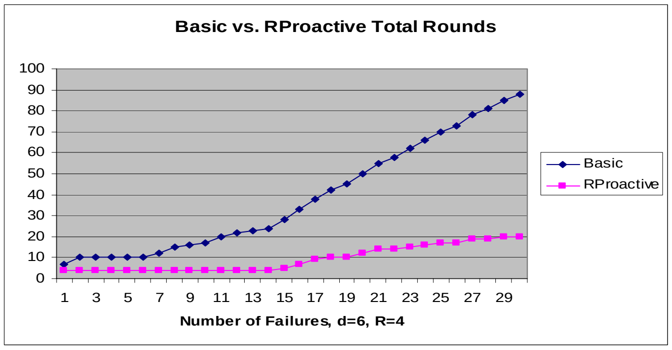
Comp(*RProact2*):=$(2^{J+1}-2^{R+1})\times C_E + 2^R\times C_E$.

Single Failure	Basic	*R*-Proactive , Round *j*
Stage1 comm/tion	$2^J\times K$	$\sum_{i=1}^{R}(2^d - F(i))\times(2^i-1)K$
Stage1 Comp/tion	$2^{J+1}\times C_E$	$\sum_{i=1}^{R}(2^d - F(i))\times(3\times 2^{i-1}-2)C_E$
Stage2 comm/tion	$2^J\times K$	$(2^J - 2^R)\times K$
Stage2 Comp/tion	$2^{J+1}\times C_E$	$(2^{J+1}-2^{R+1})\times C_E + 2^R\times C_E$
Storage	$d\times K$	$2^{R+1}\times K$

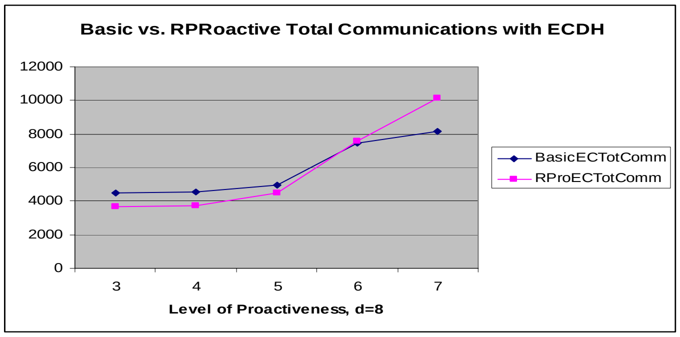
Table 1: Indicative Complexities of *R*-Proactive *vs.* Basic



Graph1:Communication due to Failures *Basic vs. R-Proactive*



Graph2: Total Number of Rounds for 30 Failures: *Basic vs. RProactive* for d=6, R=4



Graph3: Total Communication *Basic vs. R-Proactive* implemented with ECDH, varying the Proactive-ness Level *R*

From these results, it is obvious that the *R*-Proactive algorithm succeeds in reducing substantially the total number of rounds of the Hypercube execution in the presence of failures, as well as the communication cost due to failures. The total number of rounds of the protocol can be translated to the total latency, since the original Hypercube stalls, and waits for the appropriate number of rounds, to go around a failure. The **novelty** in our algorithm is that Hypercube never has to stall for a failure that occurs during the 1st stage, and as far as the 2nd stage, the number of rounds to remedy the failure is much less, since members only need to go back to round *R*. However, the total communication and computation cost is higher in general for *R*-Proactive, if the total number of transmitted bits is considered. This happens because the higher the proactive-ness level *R*, the higher the number of elements that must be exchanged among the communicating peers to construct the framework of the 1st stage. However, as we will see next, if we implement Hypercube (*Basic*, *R*-Proactive) with EC-DH, we achieve a substantial reduction in the total communication cost in our scheme, and it becomes even lower than the corresponding costs of *Basic* for most cases (if the costs are measured w.r.t. the number of packets transmitted). Nevertheless, the amount of rounds *R*-Proactive algorithm takes to execute (successfully terminate) in the presence of failures is dramatically reduced compared to the original *Basic* scheme.

ii) *Elliptic Curves Cryptography (ECC) Implementation of Hypercube and Performance Improvement*

ECC is known to offer a security level comparable to that of other cryptographic systems of larger key sizes. If we substitute the DHKEs with EC-DHKEs we can achieve an obvious reduction in the Communication and Storage Costs (i.e. DH K=1024 bits *vs.* ECDH m=169 bits key sizes). Detailed description of the EC-based *R*-Proactive algorithm can be found in [20], where we also show that the overall computation cost is reduced as well. We illustrate the significance of using 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 the *R*-Proactive algorithm, setting R=4, a member must transmit 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 therefore 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, meaning that partners need to exchange only 3 packets. Hence, *R*-Proactive becomes more powerful through the use of ECs [20, 23, 24].

VIII. BASIC *VS.* *R*-PROACTIVE SIMULATION RESULTS

i) *Simulation Set-Up and Discussion:*

We have conducted a simulation analysis in order to compare the routing cost and the combined routing/communication costs incurred to the network from the execution of *R*-Proactive vs. Basic Hypercube to gain a realistic view of the actual performance of both schemes over an ad-hoc multi-hop network. We use different graphs to generate the secure group and analyze the performance of the two algorithms. A number of nodes from this graph are randomly selected as subgroup leaders. Through the underlying routing, each leader obtains the routing path(s) to the rest of the leaders it is originally scheduled to communicate with. We assume a generic Dijkstra-based underlying routing protocol that finds the shortest paths between leaders (w.r.t. number of hops).

For our evaluation, we have generated various random graphs of different network sizes $S \in [100, 500]$ for each initial input of the number of members $n = 2^d$, and for the same graph we have varied the Hypercube parameter d, s.t. $2 < d \leq 8$. For the same graph and the same input, we have varied the group configuration, i.e., we have selected the n members in a different (random) manner in every repetition. For the *R*-Proactive we have additionally varied the “proactive-ness” parameter *R* s.t.

1 <$R \leq d$, for each graph and subgroup configuration, and for each given d. Members' failures have been simulated by the use of a probability failure parameter p, where $0.01 \leq p \leq 0.04$ for each member that is alive during every round. The probability of member' failures is uniformly distributed w.r.t. the Hypercube rounds.

For each given input <S, d, R, p>, we have run both algorithms with and without failures assumption, for 100 different Hypercube group configurations within the network graph and we have averaged the results. We have evaluated *a*) the total routing cost w.r.t. number of hops, *b*) the total number of rounds required until successful termination, and *c*) the combined overhead incurred from the exchanged key generation messages over all network nodes involved (members and their relays), for both *R*-Proactive and Basic schemes.

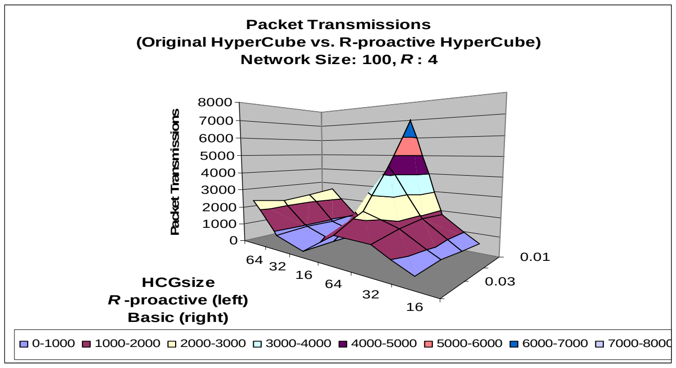
ii) Simulation Results:

Below we illustrate in the following graphs some indicative results on the combined communication/ routing overhead and number of rounds produced by both *Basic* and *R*-Proactive protocols. From careful inspection of our simulation results, we can see that they actually confirm our analytical results for all cases.

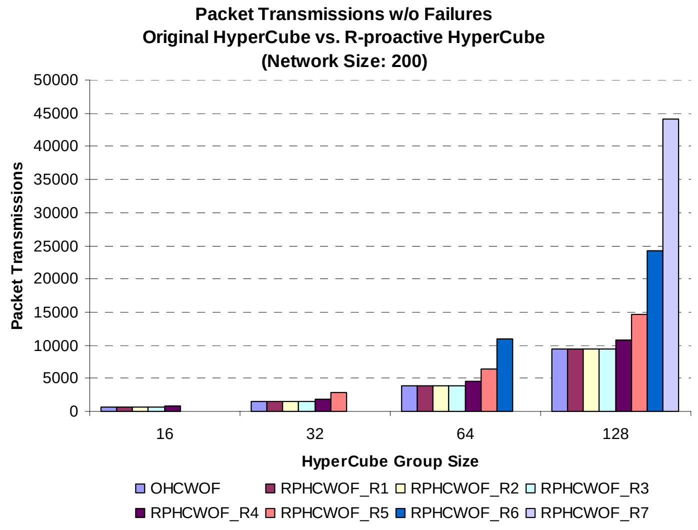
Graph4, shows in 3-*D* a comparison of *Basic* and *R-Proactive* for R=4, w.r.t. the total number of packets transmitted under increasing failure rates [0.01,…, 0.04] and increasing group size [16,…,64]. We can see that R-Proactive prevails for almost all cases. Notice that at some point when the failure rate increases significantly, i.e. p>0.03, the total packet transmissions in *Basic* decreases. This occurs because whenever the number of failed members is halved, we re-adjust the schedule of members in *Basic* and decrease the current number of rounds required for termination of the original scheme without failures by one. Still, it can be seen that *R*-Proactive demonstrates a superior behavior for almost all the scenarios reflected in this graph. Furthermore, in a real case scenario, it is not so simple to carry out the re-adjustment we suggest in Basic for the Hypercube members. They must be allocated new *ids* and co-ordinate to propagate this information and apply a new transmission schedule, which is not simple in practice.

Graph 5, shows a comparison w.r.t. the total packets transmission of *Basic* and *R-Proactive* (for R in [1..7]) when no failures are assumed in the network. We can see that without failures always the Basic scheme prevails, which is expected, because *R-Proactive* consumes extra overhead to build the proactive framework and is not compensated if no member failures occur. For example, when R=7, d=7, the packet transmission overhead is offset a great deal.

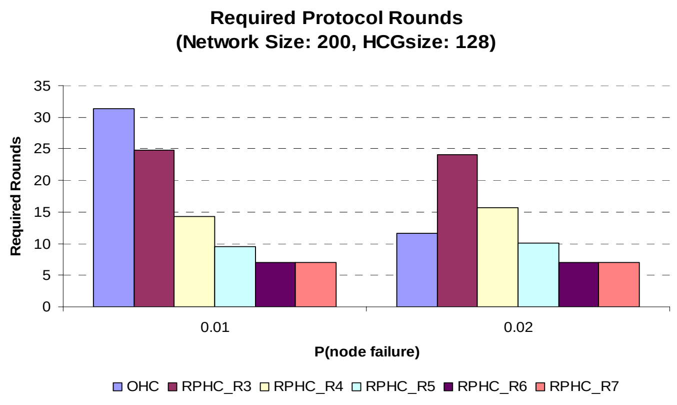
Graph 6 shows a comparison of *Basic vs. R-Proactive* (for R in [3..7]), for a network of size S=200, and d=7, w.r.t. the total number of rounds under the presence of failures, for two failure rates p = 0.01, 0.02. We can see that *R*-Proactive achieves a significant reduction in the total number of rounds, particularly as R increases, and this is a significant achievement of this algorithm. Again, we see that for p=0.02, the number of rounds for *Basic* is quite low. As in the case of graph 4, this is due to the round re-adjustment we have done on the scheme. Even in this case, for R>4, *R-Proactive* achieves an even great reduction in the total round number.



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



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



Graph 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

IX. Conclusion

This paper focuses on the design of a Fault-Tolerant Hypercube protocol when it used as the core scheme to connect subgroup leaders for Octopus-based schemes in a MANET. We describe a hybrid proactive approach for extending Hypercube protocol to tolerate failures suitable for the dynamic environment and for the application of interest. We present comparisons of two different schemes: the original Hypercube as presented in [1,5] denoted as *Basic*, and our own Fault-Tolerant adaptively proactive extension, denoted as *R*-Proactive, w.r.t. 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. Furthermore, we illustrate how the incorporation of ECC favors our proactive design and improves its overall performance.

References

[1] Klaus Becker, Uta Wille. Communication Complexity of Group Key Distribution. Proc.5th ACM Conference on Computer & Comm/tions Security, pages 1-6, San Francisco, CA, November 1998.

[2] Steiner M., Tsudik G., Waidner M., Diffie-Hellman. Key Distribution Extended to Groups. 3rd ACM Conference on Computer & Communication Security, ACM Press, 1996.31-37

[3] Maarit Hietalachti. Key Establishment in Ad-Hoc Networks. Helsinki University of Technology. Laboratory for Theoretical Computer Science.May'01.

[4] A.Perrig. Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication. Int'l Workshop on Cryptographic Techniques E-Commerce CryptTEC'99.

[5] N.Asokan and Philip Ginzboorg. Key-Agreement in Ad-Hoc Networks. Elsevier Preprint.2000.

[6] M.Striki, J.Baras. Efficient Scalable Key Agreement Protocols for Secure Multicast Comm/tion in MANETs. CSHCN TR 2002.

[7] David McGrew. Alan T.Sherman. Key-Establishment in Large Dynamic Groups Using One-Way Function Trees. May 1998.

[8] H. Harney, E.Harder. Logical Tree Hierarchy protocol. Internet Draft, Internet Eng. Task Force, April 1999.

[9] H. Harney, C.Muckenhirn. Group Key Management Protocol (GKMP). Specification/Architecture, Int. Eng. Task Force. July 1997.

[10]Y. Kim, A. Perrig, G. Tsudik Simple & Fault Tolerant Key Agreement for Dynamic Collaborative Groups.

[11]L.Lazos, R.Poovendran. Cross-Layer Design for Energy Efficient Secure Multicast Communications in Ad Hoc Networks. Proceedings of the ICC 2004 Conference, Paris, France, June 2004.

[12]Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik. Exploring Robustness in Group Key Agreement. Procs of the 21th IEEE Int'l Conference on Distributed Computing Systems, Phoenix, Arizona, April 16-19, 2001, pp 399-408.

[13]Y.Amir, Y.Kim, C.Rotaru, J.Schultz, J.Stanton, G.Tsudik. Secure Group Communication Using Robust Contributory Key Agreement. IEEE TPDS, vol. 15, no. 5, pp. 468-480, May 2004.

[14]L.Zhou, Z.Haas. "Securing adhoc networks", IEEE Network Magazine, vol. 13:, no.6, pp. 24-30, Nov/Dec '99

[15] J.Kong, P.Zerfos, H.Luo, S.Lu and L.Zhang. "Providing robust and Ubiquitous security support for MANET," IEEE ICNP 2001, 2001

[16]S.Capkun, L.Buttyan, J.Hubaux," Self-Organized Public-Key Management for MANETs," IEEE Transactions on Mobile Computing (TMC) 2002

[17]L.Eschenauer, V.Gligor., "A Key Management Scheme for distributed sensor networks"., ACM CCS'02, pages 41-47, Nov, 2002

[18] H.Chan, A.Perrig, D.Song, "Random Key Predistribution Schemes for Sensor Networks," IEEE Symposium on Security and Privacy, California, USA, May 2003.

[19] M.Striki, J.Baras. "Key Distribution Protocols for Secure Multicast Communication Survivable in MANETs". Proceedings of the IEEE MILCOM 2003, Boston MA, October 2003.

[20] M.Striki, J.Baras "Fault-Tolerant Extension of Hypercube for Efficient, Robust Group Communications in MANETs with Octopus Schemes", CSHCN TR 2005.

[21] A. Perrig, D. Song, J.D. Tygar. ELK, a new Protocol for Efficient Large-Group Distribution. Proc. of IEEE Security and Privacy Symposium, 2001, May 2001.

[22] S.Yi, R.Kravets, "Key management for heterogeneous ad hoc wireless networks," University of Illinois at Urbana-Champaign, Department of Commputer Science Technical Report #UIUCDCS-R-2001-2241, UILU-ENG-2001-1748, July 2002.

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

[24] A.M.Fisiran, R.B.Lee, "Workload Characterization of Elliptic curve Cryptography and other Network Security Algorithms for Constrained Environments", IEEE Int'l Workshop on Workload Characterization, WWC-5, 2002.