情報保護學會論文誌
第11券 第4號, 2001. 8

(2,4)-트리를 이용한 그룹키 관리*

조 태 남**, 이 상 호**

Group Key Management using (2,4)-Tree

Tae-Nam Cho**, Sang-Ho Lee**

요 약

통신 기술의 발달로 최근 원격화상회의나 다중 사용자 게임 등 그룹 중심의 응용이나 서비스가 증가하고 있다. 그러한 정보에 대한 접근제어의 수단으로서 그룹 구성원들이 하나의 그룹키를 공유하여 비밀통신을 하는데, 규모가 크고 동적인 그룹인 경우에는 효율적인 그룹키 갱신 방법이 매우 중요하다. 본 논문에서는 구성원의 가입 및 탈퇴시 발생하는 키갱신의 효율성을 높이기 위하여 높이 균형 트리인 (2,4)-트리를 이용하여 키-트리를 구성하였다. 특히, CBT (Core Based Tree)를 이용하여 네트웍 구성 정보를 유지하도록 하고 이를 키-트리 구성에 반영하도록 함으로써, 네트웍 장애 및 복구로 발생하는 일부 그룹의 분할과 병합 시에도 효율적으로 키갱신을 수행할 수 있는 방안을 제시하였다.

ABSTRACT

Recently, with the explosive growth of communication technologies, group oriented services such as teleconference and multi-player game are increasing. Access control to information is handled by secret communications with group keys shared among members, and efficient updating of group keys is vital to such secret communications of large and dynamic groups. In this paper, we employ (2,4)-tree as a key tree, which is one of height balanced trees, to reduce the number of key updates caused by join or leave of members. Especially, we use CBT(Core Based Tree) to gather network configurations of group members and reflect this information to key tree structure to update group keys efficiently when splitting or merging of subgroups occurs by network failure or recovery.

keyword : *group key management, key distribution, multicast, CBT, security*

Ⅰ. 서 론

최근 통신 기술의 발달에 힘입어 원격회의나 실시간 정보 서비스, 유료 영상 서비스, 대화형 분산 시뮬레이션, 네트웍을 통한 공동작업 등 그룹통신에 기초한 응용과 서비스들이 다양화되고 있다. 각 응용에 따라 구성원간의 통신 형태와 구성원의 변화가 다양하기 때문에, 세부적으로는 필요한 기능이나 안전성의 종류도 다르게 된다. 그러나 그룹통신을 하는 대부분의 응용에서 공통적으로 필요한 사항은 그룹의 구성원들만이 정보를 공유하거나 메시지를 송수신할 수 있으며, 그 외의 사람들은 공유할 수 없도록 하는 것이다. 구성원들의 비밀 통신은 구성원들만이 알 수 있는 하나의 비밀키를 공유하도록 하고, 이를 이용하여 메시지를 암호화하여 전송하고, 수신된 메시지를 구성원만이 복호화할 수 있도록 함으로써 성취할 수 있다. 구성원간에 공유된 비밀키를 그룹키라 하는데, 일반적으로 대칭키를 사용한다.

* 본 연구는 이화여자대학교 교내 연구비 지원으로 수행하였음.
** 이화여자대학교 컴퓨터학과 컴퓨터이론 및 보안 연구실{(tncho,shlee)@ewha.ac.kr}

그룹키 공유 방법은 제3의 신뢰기관(TTP : Trusted Third Party)이나 그룹 제어자(group controller)가 키를 생성하여 구성원에게 전달하는 키분배 방법(Key Transport)[1,2,6,9,16]과 각 구성원들이 제공하는 비밀값들을 이용하여 그룹키를 형성하고 공유하는 키합의 방법(Key Agreement)이 있다.[7,11,13,14] 전자의 경우는 방법이 비교적 간단하고 구성원들의 작업량이 적은 방법이며, 후자의 경우는 전적으로 신뢰받는 신뢰기관을 제거함으로써 구성원들이 키 구성에 참여하고자 하는 요구를 반영하고 통신 병목 현상과 공격의 집중화를 배제한 방법이다. 또한 구성원의 변동이 없거나 거의 없는 경우에는 초기 그룹키 공유방법이 중요하지만, 변동이 많은 경우에는 초기 공유방법보다 구성원의 변동시에 효율적으로 그룹키를 관리해 주는 프로토콜이 필요하다. 새로운 구성원이 추가될 경우에는 새로운 그룹키를 생성하여 기존의 구성원 및 새 구성원이 공유할 수 있도록 함으로써, 새 구성원이 가입 이전의 통신에 대해 정보는 얻을 수 없으면서 이후의 통신에 대해서는 공유할 수 있도록 한다. 만일 구성원이 탈퇴를 하거나 규약 위반 등에 의해 퇴출될 경우에도 탈퇴·퇴출된 구성원을 제외한 나머지 구성원들만이 알 수 있는 그룹키를 다시 생성하여 공유하여야 한다. 그 외에도 네트워 장애 등으로 인한 그룹의 분할이나 장애 복구로 인한 부그룹의 병합에 대해서도 안전하고 효율적으로 그룹키를 갱신할 수 있는 방법이 필요하다. 본 논문에서는 키-서버의 키 관리를 위한 저장소가 충분하고 규모가 큰 동적인 그룹에서, 구성원의 가입이나 탈퇴 뿐 아니라 네트워 고장으로 인한 그룹의 분할 및 병합으로 인한 키 갱신이 효율적으로 이루어질 수 있는 방안을 제시한다.

Ⅱ. 그룹키 관리의 필요성

그룹키 사용의 목적은 허용된(합법적) 그룹의 구성원만이 정보를 얻도록 하기 위한 것으로서, 구성원이 아니면 그룹키를 알아내지 못하도록 하는 것은 안전성을 위한 기본 조건이라 할 수 있다. 이 외에도 forward secrecy와 backward secrecy는 구성원의 가입과 탈퇴가 허용되는 동적인 그룹에서 제공해야 할 매우 중요한 요소이다.[12] 일반적으로 구성원이 가입하는 경우에는 비교적 간단하게 처리된다. 구성원이 탈퇴할 경우에는 나머지 구성원들이 새로운 키를 공유하도록 하여 탈퇴자가 이후의 키를 알 수 없도록 해야 하는데, 탈퇴자는 이미 현재의 키들을 알고 있기 때문에 조금 더 복잡하다. 그룹키가 변경되어야 하는 또 다른 경우는 네트워의 고장으로 인한 그룹의 분할과 복구로 인한 그룹의 통합이다. 이 경우에는 한 구성원의 변동시 보다 더 복잡하게 되는데, 분할의 경우가 더욱 복잡하여 효율적인 방법은 아직 제시된 바가 없다.

본 논문에서는 키-서버가 존재하는 키 분배 프로토콜에서의 키-트리를 다루며, 다음과 같이 표기를 정의한다.

- n : 그룹의 크기(구성원의 수)
- m_i : i 번째 구성원
- k_i : 구성원 m_i와 키-서버간의 공유키
- $k_{i\sim j}$: 구성원 m_i, $i\le x\le j$들과 키-서버와의 공유키
- $k_{i,j}$: 구성원 m_i, m_j와 키-서버와의 공유키
- $\{M\}_{key_1,key_2}$: 메시지 M을 key_1과 key_2로 각각 암호화한 암호문

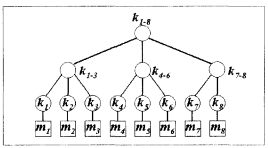
Ⅲ. 기존 연구

3.1 키-트리(Key Trees)

키-트리는 키-그래프(key graph)의 일종으로서 다음과 같은 노드로 구성되는 트리이다.[15]

- 구성원-노드 : 단말 노드로서 그룹 구성원을 표현한다.
- 키-노드 : 내부 노드로서 하나의 키에 대응된다. 이 키는 이 노드를 루트로 하는 부트리의 모든 구성원이 공유한다.
- 루트 : 그룹키를 나타낸다.

[그림 1]은 구성원이 m_1, m_2, ..., m_8인 그룹에 대한 키-트리의 예이다. 각 구성원은 해당하는 단말 노드로부터 루트까지의 경로상에 있는 키를 소유하고, 하나의 키-노드는 그 노드를 루트로 하는 부트리의 모든 구성원들이 공유한다. 즉, m_4는 k_4, k_{4-6}, k_{1-8}을 소유하고, k_{1-6}은 m_3, m_5, m_6들이 공유한다. 한 구성원이 소유하는 키의 개수 m은 최대 트리의 높이 h와 같다. 키-트리의 차수가 d일 때, 트리가 균형 잡혀



[그림 1] 키-트리의 예

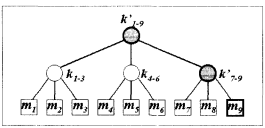
있을 경우에는 $m = h = O(\log_d n)$이며, 균형이 잡혀 있지 않다면 최악의 경우 $m = h = O(n)$이다.

본 논문에서는 [그림 1]의 k_1, k_2, ..., k_8과 같이 각 사용자와 키-서버만이 공유하는 키는 키-트리에서 표현을 생략한다.

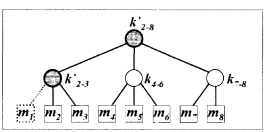
3.2 키-트리를 이용한 그룹키 관리

키-트리의 모든 키-노드는 그 키-노드를 루트로 하는 부트리의 구성원에게 새로운 키를 전송할 때, 새로운 키의 암호화/복호화에 사용된다. 차수가 2인 이진트리를 이용할 경우, 생성되는 키의 수는 $n+(n-1) = 2n-1$개가 되며, 각 구성원은 약 $\log_2 n$ 개의 키를 소유하게 된다. 성형 구조(star shaped)[7]에서는 키-서버와 각 구성원들간에 공유키를 하나씩 보유하고 이를 이용하여 키-서버가 그룹키를 분배한다. 따라서 각 구성원은 2개(서버와의 공유키, 그룹키)의 키만 보유하고 서버는 총 $n+1$개의 키를 생성한다. 키-트리를 이용한 방식은 성형 구조를 이용한 방식보다 많은 키가 생성될 뿐 아니라 각 구성원이 소유해야 하는 키의 수도 많다. 그러나 성형 구조에서는 키 갱신 시에도 그룹 구축 시와 동일한 방법으로 키를 분배하여야 하기 때문에 둘 다 구성원 수에 비례하는 $O(n)$의 작업이 필요하게 된다.

[그림 1]과 같이 키-트리가 구성되어 있을 때, m_9가 가입한다면 k_1 ~ k_8과 k'_{1-9}과 k'_{7-9}로 변경되어야 한다([그림 2] 참조). k'_{1-9}는 m_9와 기존의 모든 구성원들에게 분배되어야 하고, k'_{7-9}는 m_7, m_8 및 m_9에게 전달되어야 한다. m_9에게는 k'_{1-9}, k'_{7-9}를 서버와의 공유키인 k_9로 암호화하여 보낸다. 나머지 구성원에게는 갱신된 키인 k'_{1-9}과 k'_{7-9}를 기존의 키인 k_{1-8}과 k_{7-8}로 각각 암호화한다. 이를 하나의 메시지인 $\{\{k'_{7-9}\}_{k_{1-8}}, \{k'_{1-9}\}_{k_{7-8}}\}$로 구성하여 멀티캐스팅한다.



[그림 2] 구성원 m_9의 가입



[그림 3] 구성원 m_1의 탈퇴

[그림 1]에서 m_1이 탈퇴하는 경우에는 m_1이 알고 있는 모든 키를 변경하여야 하며([그림 3] 참조), 변경된 키는 m_1이 알지 못하는 키로 암호화해야 하므로 $\{\{k'_{2-3}\}_{k_{4-6}}, \{k'_{2-8}\}_{k_{7-8}}, k_{7-8}\}$을 보내어 순차적으로 복호화하여 새로운 키를 얻을 수 있도록 한다.[15]

예에서 본 바와 같이, 구성원이 가입하거나 탈퇴하는 경우에 변경된 구성원으로부터 루트까지의 경로 상에 있는 키들을 변경해야 하므로, $O(\log n)$의 키 변경이 필요하다.

키 갱신 시에 변경되는 키의 개수는 변경을 유발한 구성원으로부터 루트에 이르는 경로 상의 모든 키-노드의 수로서 트리의 높이에 비례한다. 만약 구성원의 불규칙적인 가입과 탈퇴로 키-트리의 균형이 깨지고 사향트리(skewed tree)가 된다면, 키 갱신의 작업이 $O(n)$이 되므로 키-트리의 사용은 의미가 없어진다. 즉, 트리의 높이가 $\log n$에 비례한다는 전제 하에 키 갱신의 효율성이 보장되므로 키-트리의 높이를 $O(\log n)$에 유지하는 문제는 효율적인 키 관리에 있어서 매우 중요하다. 그러나 대부분의 그룹키 관리 프로토콜들은 트리의 균형을 가정하여 사용하고 있다.[7,9,11,13,14,16]

다음 절에서 구성원의 가입과 탈퇴에 의한 키-트리의 불균형을 조정하기 위하여 기존에 제안된 두 가지 방법에 대해 소개한다.

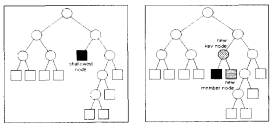
3.3 경험적(Heuristic) 방법에 의한 재균형(re-balancing)

IRTF draft로 제출된 제안서[9]에서 2진 키-트리를 경험적으로 재균형하는 방법들을 제시하였다.

첫 번째 방법에서는 트리에서 가장 깊이가 깊은 노드인 d-노드(deepest node)와 가장 낮은 노드인 s-노드(shallowest node)를 기억하고 있다. 가입하는 구성원은 [그림 4(a)]와 [그림 4(b)]에서와 같이 s-노드에 첨가한다. 한 구성원 m이 탈퇴하여 해당 구성원-노드를 제거할 때는 d-노드와의 깊이 차가 임계값을 넘는지 검사한다. 만일 임계치를 넘지 않으면, m을 제거하고, 그렇지 않으면, d-노드를 제거하고 이를 m의 위치에 재삽입한다. 구성원 제거의 예를 [그림 5(a)]와 [그림 5(b)]에 나타내었다.

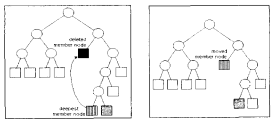
또 다른 방법들로는 구성원이 변경될 때 곧 바로 키-트리에 반영시키지 않고 일정한 주기가 지나거나, 특정 조건을 만족할 때 일괄적으로 반영시키는 방법을 제시하였다.

그러나 s-노드와 d-노드에 대한 정보 유지가 어렵고, [그림 4]와 [그림 5]에서 보는 바와 같이 키-트리가 사향트리가 되는 근본적인 문제를 해결하지 못하였기 때문에 트리의 높이를 $O(\log n)$으로 보장하지 못하며, 최악의 경우에는 $O(n)$의 높이를 갖는다.



(a) 구성원의 삽입 위치 (b) 가입 구성원 삽입 후의 키-트리

[그림 4]



(a) 구성원 탈퇴로 인한 키-트리 조정 (b) 구성원 탈퇴 후 조정된 키-트리

[그림 5]

3.4 AVL-트리를 이용한 방법에 의한 재균형

이 방법은 O. Redeh 등에 의하여 제안된 것으로서, 구성원들을 단말 노드로 하고 공유키들을 내부 노드로 하는 AVL-$트리$를 키-트리로 사용한다.[14] 구성원의 가입은 기존의 그룹과 구성원이 하나인 그룹과의 병합으로 생각할 수 있다. 구성원이 탈퇴할 경우에는 갱신되어야 할 키-노드들을 제거하고, 이로 인해 생기는 $O(\log n)$개의 AVL-키트리들을 병합하면서 새로운 키-노드들을 생성한다.

n명의 구성원을 가지는 초기 그룹에 대한 그룹키를 생성하기 위한 키-트리는 재귀적으로 구성된다. 구성원의 수 n이 2 이상이면 $n/2$명의 구성원으로 구성된 좌우 AVL-키트리를 구성하고, 두 부트리의 부모노드인 키-노드를 생성하는 방식이다.

두 AVL-키트리를 병합한 결과도 AVL-키트리이며 높이는 $O(\log n)$에 비례하므로, 구성원의 가입과 탈퇴에 따른 키-트리 갱신의 작업량 및 키-트리의 높이도 항상 $O(\log n)$을 보장한다. 그러나 이 방법에서는 네트워의 장애로 인하여 다수의 구성원이 탈퇴하는 경우에 대해서는 효율적인 해결 방법이 없고 각 구성원의 탈퇴 절차를 반복하여야 한다.

Ⅳ. $(2,4)$-$트리$를 이용한 키관리

키-트리의 균형은 구성원이 동적으로 변하고 규모가 큰 그룹에서 효율적으로 그룹키를 관리하기 위해 필수적으로 필요한 조건이다. 키-트리의 균형을 위한 경험적 방법은 처리가 복잡하고 구성원 변경 시 s-노드와 d-노드의 유지 및 갱신이 어려울 뿐 아니라, 높이가 $O(\log n)$임을 보장할 수 없기 때문에 키 갱신의 복잡도도 $O(\log n)$을 보장하지 못한다. AVL-$트리$를 이용한 방법에서는 효율적인 방법으로 키-트리의 높이를 $O(\log n)$으로 유지한다. 그러나 AVL-$트리$를 이용한 방법의 효율성은 키-트리의 구성원-노드들이 순서화되어 있지 않는 데서 얻는다. 즉, 새로운 구성원의 가입으로 키-트리에 추가될 경우, 추가되는 구성원-노드의 정보에 따라 트리의 삽입위치가 결정되는 것이 아니라 트리의 균형을 유지하기 위한 임의의 위치에 추가된다. 따라서 네트워 고장으로 인하여 m명의 구성원이 탈퇴를 할 경우, 최악의 경우 m번의 탈퇴 절차를 거쳐 $O(m \cdot \log n)$번의 키 갱신이 필요하게 된다. 그 이유는 AVL-키트리에 구성원들의 지역성(locality)이 반영되지 않아서, m명의 구성원이 트리 전체에 산재해 있을 수 있기

때문이다.

본 장에서는 구성원들의 지역성을 반영하여 네트웍 장애로 인한 그룹의 분할 및 병합에도 효율적으로 키-트리의 균형을 유지할 수 있도록 (2,4)-*트리*를 이용한 새로운 키 관리 방법을 제시한다. 또한 키-트리에 지역성을 반영하기 위한 방법으로서 *CBT*를 이용하여 네트웍의 구성 정보를 수집하는 방법을 제시하고, 그 예를 보일 것이다.

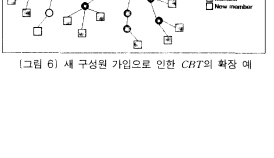
4.1 *CBT*를 이용한 그룹 구성 정보 유지

4.1.1 *CBT*

CBT(Core-Based Tree)는 멀티캐스팅(multi-casting)을 위해 제안된 것이다. 멀티캐스팅을 위한 라우팅 기법들은 소스마다 하나의 배달 트리(delivery tree)를 유지하면서 최적의 경로로 데이터를 멀티캐스팅한다.[4,15] *CBT*-기반 멀티캐스팅 기법은 가장 최근에 제안된 방법으로서, 그룹 가입 및 탈퇴가 소스 주도(source-initiate)로 이루어지는 것이 아니라 사용자 주도(user-initiate)로 이루어지며, 하나의 그룹에 유일한 배달 트리를 유지하기 때문에 소스와 무관한 그룹키를 생성하고 관리하는데 적용될 수 있다.[2] 이러한 특성 때문에 최근 *CBT*를 이용한 그룹키 관리 기법들이 연구되고 있다.[1,6,10]

*CBT*는 루트인 코어(core)라고 하는 하나의 특별한 라우터를 가지는데, 여기서는 코어를 키-서버로 설정한다. 가입하고자 하는 구성원은 로컬 라우터에 이를 보고하고,[15] 이 라우터는 코어로 가입 요청을 전달한다. 로컬 라우터로부터 코어로 찾아가는 경로 상에서 이미 *CBT*에 포함되어 있는 라우터를 만나게 될 때까지의 경로가 배달 트리의 일부로 확장된다. (그림 6)에 *CBT*의 확장 예를 나타내었다.

각 라우터는 그룹별로 구성원들이 연결되어 있는 인터페이스들을 기억함으로써 멀티캐스트 데이터의 전달 정보를 유지한다.

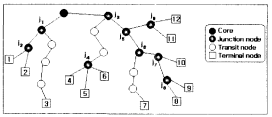


(그림 6) 새 구성원 가입으로 인한 *CBT*의 확장 예

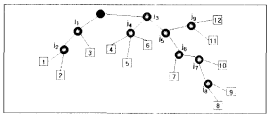
4.1.2 *CBT*를 이용한 *RT* 구성

구성원들의 상대적인 위치를 파악하고 이를 키-트리에 반영하기 위하여 키-서버는 *RT*(Reduced-tree)를 유지한다. *RT*는 *CBT*로부터 유도되며 다음과 같이 정의된다.

- *transit*-노드 : *CBT*에서 자식 노드가 하나인 라우터
- *junction*-노드 : *CBT*에서 자식 노드가 두 개 이상인 라우터
- *non-member*-노드 : *CBT*에 속하지 않은 라우터
- *구성원*-노드: 그룹에 속한 구성원
- *RT* : *CBT*에서 *transit*-노드들을 제거하여 코어와 *junction*-노드 및 *구성원*-노드만으로 구성한 트리



(a) 그룹의 *CBT*



(b) *RT*

(그림 7)

*RT*의 각 노드는 대응되는 라우터나 구성원 id와 자식 노드 및 부모 노드에 대한 링크를 가진다.

어떤 그룹에 대한 *CBT*가 (그림 7(a))와 같을 때, 그에 대한 *RT*는 (그림 7(b))와 같다.

CBT 라우터는 그룹별 라우팅 정보로부터 자신이 *transit*-노드인지 *junction*-노드인지 혹은 *non-member*-노드인지 알 수 있다. 여기서는 편의상 자식 노드의 수가 *num_child*에 저장되어 있고, *non-member*-/*transit*-/*junction*-노드인지의 구분은 *my_state*에 저장되어 있다고 가정한다.

RT 정보는 새로운 구성원이 가입하면서 *CBT*를

확장할 때, 가입 요청 메시지를 통하여 키-서버인 코어로 전달하도록 한다.

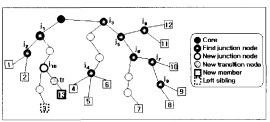
이제 키-트리의 갱신을 유발시키는 네 가지 이벤트인 한 구성원의 가입이나 탈퇴, 네트웍 고장으로 인한 그룹의 분할과 병합에 대하여 어떻게 *RT*를 유지하는지 보이고자 한다.

본 논문에서 사용하는 가입/탈퇴 요청은 송신자 및 메시지 인증을 포함하지 않고 있으나, 이는 라우터 및 호스트(구성원)의 공개키/개인키를 이용하여 서명함으로써 얻을 수 있다.

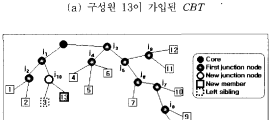
(1) 구성원의 가입

한 사용자가 그룹에 가입하게 되면, 새 구성원으로 인하여 확장되는 *CBT*의 경로상에 있는 라우터들은 *non-member*-노드에서 *transit*-노드가 되거나 *transit*-노드에서 *junction*-노드로 상태변화가 일어날 수 있다. 예로, [그림 7]에서 새로운 구성원 m_{13}이 가입할 경우 *CBT*는 [그림 8(a)]와 같이 확장될 것이고, 그에 대한 *RT*는 [그림 8(b)]와 같이 수정되어야 한다.

라우터들의 상태 변경은 새 구성원으로부터 코어로 가입 메시지를 전달하면서 이루어진다. 이 중, *transit*-노드에서 *junction*-노드로 변경되는 라우터에 대한 정보와 처음으로 만나는 *junction*-노드에 대한 정보는 코어로 전달되어 *RT* 수정에 사용된다. [그림 8(a)]에서 *tr*로 표시된 라우터는 *non-member*-노드에서 *transit*-노드로 상태변화를 일으킨 라우터이고, j_{10}은



(a) 구성원 13이 가입된 *CBT*



(b) 구성원 13이 가입된 *RT*

[그림 8]

transit-노드에서 *junction*-노드가 된 라우터이며 첫 번째로 만나는 기존의 *junction*-노드는 j_1이다. m_{13}의 가입으로 인하여 *RT*에서는 j_1의 자식 노드였던 m_3 대신 새로운 *junction*-노드인 j_{10}이 추가되고, m_3와 m_{13}은 j_{10}에 연결되어야 한다. 그러므로 m_{13}이 j_1의 어느 인터페이스를 통해 연결되는지, m_3와 m_{13}은 j_{10}의 어느 인터페이스를 통해 연결되는지에 대한 정보를 코어로 전달해야 한다.

가입 신청 메시지인 *msg*는 [그림 9]와 같이 구성된다.

g_id	*m_id*	*first_jct*		*new_jct*		
		id	*new_inf*	*id*	*new_inf*	*old_inf*

[그림 9] 가입 메시지 구성

- *g_id* : 가입하고자 하는 그룹의 id
- *m_id* : 가입하고자 하는 사용자(새 구성원 혹은 호스트) id
- *first_jct* : 새 구성원으로부터 코어까지의 경로에서 처음으로 만나는 기존의 *junction*-노드
  - *id* : 라우터 id, 초기치는 null
  - *new_inf* : 새로운 구성원이 연결된 인터페이스 번호
- *new_jct* : *transit*-노드에서 *junction*-노드가 되는 라우터에 대한 정보
  - *id* : 라우터 id, 초기치는 null
  - *new_inf* : 새 구성원이 연결된 인터페이스 번호
  - *old_inf* : 기존의 구성원들이 연결되어 있는 인터페이스 번호

[그림 8(a)]에서, m_3와 m_{13}이 j_1의 i_1번째 인터페이스를 통해 연결되어 있고, j_{10}의 i_2와 i_3번째 인터페이스를 통해 각각 연결되어 있다고 가정하자. 그러면 *msg.first_jct.id*는 j_1, *msg. first_jct.new_inf*는 i_1, *msg.new_jct.id*는 j_{10}이고, *msg.new_jct. ld_inf*는 i_2이며, *msg. new_jct.new_inf*는 i_3이다.

새 구성원의 가입 요청에 대한 처리 단계를 알고리즘 1에 나타내었다. 가입하려는 사용자는 *g_id*와 *m_id* 만을 설정하여 로컬 라우터로 보내고, 로컬 라우터로부터 코어까지의 경로에서 만나는 라우터들은 수신한 *msg*에 대하여 알고리즘 1을 수행하여, 자신의 상태를 변경시키고 코어로 보낼 정보를 메시지에 저장한다.

```
switch (my_state) {
case (junction) :
   if (receive message from new interface)
      num_child = num_child + 1;
   if (msg.first_jct.id==null) {
      msg.first_jct.id ← router id;
      msg.first_jct.new_inf
            ← incoming interface number;
   }
   break;
case (transit) :
   if (receive message from new interface) {
      num_child = num_child + 1;
      my_state ← junction;
      msg.new_jct.id ← router id;
      msg.new_jct.new_inf
            ← incoming interface number;
      msg.new_jct.old_inf
            ← old interface number;
   }
   break;
default : /* non_member */
         num_child ← 1;
         my_state ← transit;
}
pass message to core;
```

〔알고리즘 1〕 새 구성원 가입 요청 처리

키-서버는 가입 요청 메시지에 있는 정보를 이용하여 RT를 수정한다. 〔그림 8〕의 예에서, j_1의 i_1번째(*msg.first_jct.new_inf*) 인터페이스에 연결되어 있던 m_3 자리에 j_{10}(*msg.new_jct.id*)을 삽입하고, m_3와 m_{13}은 j_{10}의 i_2번째(*msg.new_jct.old_inf*)와 i_3번째(*msg.new_jct.new_inf*) 인터페이스의 자식 노드로 각각 연결한다. 키-트리에서 새 구성원이 삽입될 위치(*left_node* 오른편에 삽입)를 계산한다. 예에서 삽입위치는 m_3가 된다.

가입 요청 메시지에 대한 키-서버의 RT는 알고리즘 2를 따라 수정된다.

(2) 구성원의 탈퇴

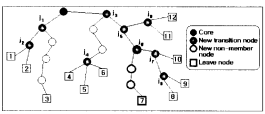
한 구성원이 그룹에서 탈퇴하면, 이 구성원으로부터 코어로 이르는 경로상의 라우터들은 *junction*-노드에서 *transit*-노드로, *transit*-노드에서 *non-member*-노드로 상태 변화가 일어날 수 있다. 이러한 상태변화를 반영하기 위하여 각 라우터는 탈퇴 요청을 한 구성원이 제거되더라도 자신이 *CBT* 노드로 남게 되는

```
create a new node new_mem;
new_mem.id ← msg.m_id;
if (msg.new_jct.id != null) {
   create a new node new_jct;
   new_jct.id ← msg.new_jct.id;
   msg.new_jct.old_inf th child of new_jct
      ← msg.first_jct.new_inf th child
         of msg.first_jct.id;
   msg.first_jct.new_inf th child of
      msg.first_jct.id ← new_jct;
   msg.new_jct.new_inf th child of new_jct
         ← new_mem;
}
else {
   new_inf th child of first_jct.id
         ← new_mem;
}
left_node ← left member of new_mem;
```

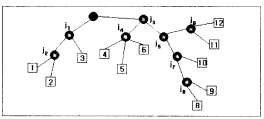
〔알고리즘 2〕 새 구성원 가입에 따른 RT의 수정

지 여부를 자기 부모 노드로 알려준다.

예로 〔그림 7〕에서 m_7이 탈퇴한다면, 〔그림 10(a)〕와 같이 j_6은 *junction*-노드에서 *transit*-노드로 변경되고, j_6과 m_7 사이의 라우터들은 *non-member*-노드로 변경된다. 이를 반영한 RT는 〔그림 10(b)〕와 같다.



(a) m_7 탈퇴로 인한 *CBT* 변경



(b) m_7 탈퇴로 인한 RT 변경

〔그림 10〕

그룹 탈퇴 신청 메시지인 *msg*는 〔그림 11〕과 같이 구성된다.

g_id	m_id	no_child

〔그림 11〕 탈퇴 메시지 구성

- *g_id* : 탈퇴하고자 하는 그룹의 id
- *m_id* : 탈퇴하고자 하는 구성원(호스트) id
- *no_child* : 라우터에 더 이상 연결된 구성원이 없음을 표시

가입하려는 구성원은 *g_id*와 *m_id* 만을 설정하여 로컬 라우터로 보낸다. 로컬 라우터는 자신에게 연결된 네트워크에 더 이상의 구성원이 없으면 *no_child*를 "yes"로 설정하고 그렇지 않으면 "no"로 설정한다. 이를 위해 로컬 라우터로부터 코어까지의 경로에서 만나는 라우터들의 처리절차를 알고리즘 3에 나타내었다.

키-서버는 *RT*에서 탈퇴하는 구성원 노드를 삭제하고 이에 따라 *transit*-노드가 되거나 *non-member*-노드가 되는 노드들을 삭제한다. 코어는 이미 *RT*에 대한 정보를 모두 가지고 있으므로 수정에 필요한 정보를 별도로 수집하지 않는다.

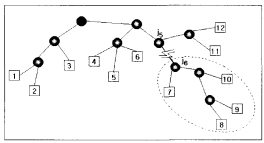
〔그림 10(a)〕의 예에서 *transit*-노드로 변경되어 *RT*로부터 제거되는 j_6이 j_5의 i_1번째 인터페이스에 연결되어 있었다면, j_6에 연결되어 있던 j_7을 j_5의 i_1번째 인터페이스의 자식노드로 변경한다.

```
if (msg.no_child == yes) {
  num_child ← num_child - 1;
  switch (num_child) {
    case 0: state ← non_member;
            no_child ← yes;
    case 1: state ← transit;
            no_child ← no;
    default: no_child ← no;
  }
}
pass message to core;
```

〔알고리즘 3〕 구성원 탈퇴 요청 처리

(3) 그룹 분할 및 병합

네트워크 장애로 인하여 일부 구성원들이 그룹 통신을 할 수 없을 때, 그 구성원들을 *RT*와 키-트리로부터 제거하고, 새로운 키들을 나머지 구성원들에게 분배하여야 한다.



〔그림 12〕 네트워크 장애로 인한 그룹 분할

〔그림 12〕에서 보는 바와 같이 j_5와 j_6 사이 링크의 장애로 구성원 m_7, m_8, m_9, m_{10}을 그룹으로부터 제거할 때, 해당 링크를 *RT*에서 비활성화 시키고 나중에 장애가 복구되면 다시 활성화시킨다. 이에 대응되는 키-트리의 변경은 다음 절에서 설명한다.

4.2 (2,4)-트리를 이용한 키관리

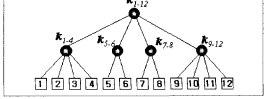
*(2,4)-트리*는 높이 균형 트리의 하나로서 임의의 위치에서 노드의 삽입과 제거가 일어나더라도 $O(\log n)$의 갱신시간과 트리 높이를 보장한다.[8]

*(2,4)-트리*를 이용한 키-트리 *HKT*(Height Balanced Key Tree)는 다음과 같은 성질을 만족하는 키-트리이다.

- 모든 구성원-노드는 동일한 깊이에 있다.
- 모든 키-노드의 자식은 2개 이상이고 4개 이하이다.

*HKT*의 구성원-노드에는 대응되는 구성원 id와 서버와의 공유키가 저장되어 있다. *HKT*는 일반 *(2,4)-트리*와 달리 루트로부터 단말노드를 찾아가는 탐색트리(search tree)가 아니기 때문에, 루트를 제외한 모든 노드는 자식 노드가 아니라 부모 노드로의 링크를 가지고 있다.

〔그림 7〕의 예에 해당하는 *HKT*를 나타내면 〔그림 13〕과 같다(구성원의 가입 순서에 따라 키-트리의 구성은



〔그림 13〕 그림 7에 대한 키-트리

달라질 수 있다).

(1) 구성원의 가입

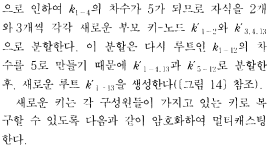
관리 철체에 따라 새 구성원의 가입이 승인되면, 키-서버와 새 구성원은 안전한 채널을 통하여 하나의 키를 공유한다. 새 구성원을 *HKT*에 반영하기 위해서는, 새 구성원의 id와 서버와의 공유키를 저장하는 구성원-노드를 생성하고 이를 *RT* 수정 시에 선정된 왼쪽 형제 구성원-노드 *left_node*의 오른편 형제 구성원-노드로 삽입한다. 만일 이 노드의 삽입으로 인하여 부모 키-노드의 차수가 5가 되면, 부모 키-노드를 2개로 분할하여 자식 구성원-노드를 각각 2개와 3개로 나누고, 그렇지 않으면 삽입이 종료된다. 노드의 분할이 일어난 경우에는 부모 키-노드에 대하여 동일한 과정을 반복한다.[8] 삽입이 종료되면, 새로운 구성원-노드로부터 루트까지의 키-노드들과, 분할된 키-노드들을 갱신하여 구성원들에게 배포한다.

예로, [그림 13]의 *HKT*에 새로운 구성원 m_{13} 이 가입되는 경우에, m_{13}은 알고리즘 2에서 *left_node*로 선정된 m_7의 오른쪽 형제로 삽입된다. 그러나 삽입으로 인하여 k_{1-4}의 차수가 5가 되므로 자식을 2개와 3개씩 각각 새로운 부모 키-노드 k'_{1-2}와 $k'_{3,4,13}$으로 분할한다. 이 분할은 다시 루트인 k_{1-12}의 차수를 5로 만들기 때문에 $k'_{1-4,13}$과 k'_{5-12}로 분할한 후, 새로운 루트 k'_{1-13}을 생성한다([그림 14] 참조).

새로운 키는 각 구성원들이 가지고 있는 키로 복구할 수 있도록 다음과 같이 암호화하여 멀티캐스팅한다.

$\{\{k'_{1-13}\}_{k_{1-4},k_{5-12}}, \{k'_{3,4,13}\}_{k_3,k_4,k_{13}}, \{k'_{1-4,13}\}_{k_{1-4},k'_{5-12}}$

$\{k'_{5-12}\}_{k_5,\ldots,k_7,\ldots,k_{9-12}}, \{k'_{1-13}\}_{k'_{1-4,13},k'_{5-12}}\}$



[그림 14] 구성원 13이 가입된 후의 *HKT*

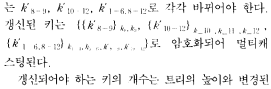
갱신되는 키의 개수는 트리의 높이와 분할되는 키-노드의 개수의 합이다. 키-노드의 분할은 최악의 경우에 각 레벨에서 한번씩 일어나므로, 키 갱신의 개수는 $O(\log n)$이다.

(2) 구성원의 탈퇴

구성원이 탈퇴하면 *HKT*에서 그에 대응되는 구성원-노드 l을 제거하고 부모 키-노드 v로의 링크를 제거한다. 만일 v의 차수가 2이상이면, 노드의 제거는 완료된다. 만일 v의 차수가 1이고 부모노드의 오른쪽(혹은 왼쪽) 노드 y의 차수가 2보다 크면, y의 최좌측 자식 노드를 v의 최우측 노드로 옮긴다. 만일 v의 차수가 1이고 y의 차수도 2이면, v의 자식을 y의 좌좌측 자식으로 옮겨주고 v를 제거한다. v을 제거한 경우에는 v의 부모노드의 차수가 감소되므로 위와 같은 과정을 반복한다.[8]

[그림 13]에서 구성원 m_7이 탈퇴할 경우에 k_{7-8}의 차수가 1이 되므로 오른쪽 형제의 최좌측 노드인 m_9을 가져와 k_{7-8}의 최우측 자식노드로 만든다([그림 15] 참조). 이에 따라 k_{7-8}, k_{9-12}, k_{1-12}는 k'_{8-9}, k'_{10-12}, $k'_{1-6,8-12}$로 각각 바뀌어야 한다. 갱신된 키는 $\{\{k'_{8-9}\}_{k_8,k_9}, \{k'_{10-12}\}_{k_{10},k_{11},k_{12}}, \{k'_{1-6,8-12}\}_{k_1,\ldots,k_6,\ldots,k'_{8-9},\ldots}\}$로 암호화되어 멀티캐스팅된다.

갱신되어야 하는 키의 개수는 트리의 높이와 변경된 자식의 수의 합이다. 키-노드의 병합이 일어난 경우에는 추가의 키 갱신이 필요하지 않으며, 오른편(왼편) 키-노드의 자식을 가져올 때는 한 개의 키-노드 갱신이 추가되나 트리의 수정작업이 더 이상 루트로 전파되지 않는다. 따라서 추가의 키 변경은 최대 1개이고 구성원 탈퇴에 따른 키 갱신의 개수의 합은 $O(\log n)$ 이다.



[그림 15] m_7이 탈퇴한 후의 *HKT*

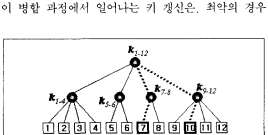
(3) 그룹 분할 및 병합

네트웍 장애로 인하여 그룹에서 제거되어야 하는 구성원들은 *HKT*에서 모두 인접해 있다. 이것은 네트웍 정보를 *RT*에 유지하고 이에 따라 *HKT*를 구

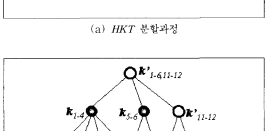
성하기 때문이다. 키-트리를 구성하는 구성원들을 왼쪽부터 순서화했을 때, $M=\{m_{i_1}, m_{i_2}, \ldots, m_{i_r}\}$라 하고, 네트웍 장애로 구성원 $M_1=\{m_j, \ldots, m_{i_r}\}$이 제거된다고 하자. *HKT*에서, 루트로부터 m_{i_1}까지의 경로상의 노드를 제거하면 m_j의 왼쪽 노드들로 구성된 트리들의 집합 F_1과 m_j의 오른쪽 노드들로 구성된 트리들의 집합 F_2가 생성된다. 만약 F_2에 M_1의 구성원이 있다면, 반드시 m_{i_r}도 포함하므로 m_{i_r}가 속한 트리에서 루트로부터 m_{i_r}까지의 경로상의 노드들을 제거하고 M_1의 노드들로만 구성된 트리를 제거한 나머지를 F_3라 하자. 그러면 F_1과 F_2의 트리의 수는 $O(\log n)$이며 이들의 병합도 $O(\log n)$의 시간이 걸린다.[5]

[그림 12]의 예에서와 같이 $M_1=\{m_7, m_8, m_9, m_{10}\}$일 경우, *HKT*의 분할은 [그림 16(a)]와 같으며, 갱신된 *HKT*의 결과는 [그림 16(b)]와 같다.

네트웍 복구로 제거되었던 구성원들이 재가입하게 될 때에는, 분할되었던 부트리들을 F_3라 할 때 이들을 병합하고, F_3가 삽입될 위치를 중심으로 *HKT*를 F_1과 F_2로 분할한 후, F_1, F_3, F_2를 하나씩 병합한다. 높이(T_1)≤높이(T_2)인 두 개의 트리 T_1과 T_2를 병합하는 방법은 T_2를 T_2의 레벨과 같은 T_1의 최우측 노드의 오른쪽 형제로 삽입하는 것이다. 이 병합 과정에서 일어나는 키 갱신은, 최악의 경우

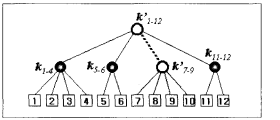


(a) *HKT* 분할과정



(b) 네트웍 장애로 인한 *HKT* 갱신

[그림 16]



[그림 17] 네트웍 복구로 인한 키-트리 갱신

m_{i_1}와 m_{i_r}을 제거하는데 필요한 키 갱신의 수를 합한 것과 같으므로 $O(\log n)$만큼의 키 갱신이 필요하다.[5] 분할되었던 그룹이 병합되어 갱신된 *HKT*의 예가 [그림 17]에 나타나 있다.

4.3 효율성 분석

키-서버는 기존의 방법에 비해 *RT*를 별도로 유지하여야 한다. *RT*는 실제 네트웍의 구성을 간소화한 것으로서, 최악의 경우는 모든 구성원이 각각 다른 *junction*-노드에 연결된 경우로서 $2n$개의 노드를 위한 저장소가 필요하다. 또한 *RT*에 추가·삭제된 노드가 *HKT*에 추가·삭제되기 위해서는 *RT*의 각 구성원 노드가 *HKT*에서의 대응 노드로 링크되어 있어야 한다. 링크를 위한 저장소도 $O(n)$만큼 필요하므로 총 $O(n)$의 추가 저장소를 필요로 하게 된다. 키-머서가 *RT*와 키-트리를 갱신하기 위한 시간은 최악의 경우에 각각 $O(n)$과 $O(\log n)$을 필요로 한다. 또한 *RT* 유지를 위해, 구성원의 가입 시에 [그림 9]에 있는 부가의 정보가 키-서버로 전달되어야 한다. 그룹 식별자 *g_id*, 구성원 식별자 *m_id*와 라우터의 식별자가 ip 주소와 같이 32비트라 할 때, 가입 요청 메시지는 약 16바이트 증가한다. 분배되는 키의 크기를 512비트라 가정하고 구성원의 수가 1000이상이 되는 규모가 큰 그룹에서, *RT*유지를 위한 데이터 전송량의 증가는 제안된 방식으로 인해 절약되는 키 전송량과 비해서 작은 비용이라 할 수 있다.

본 논문에서 제시한 방법은 구성원 하나의 가입과 탈퇴 뿐 아니라 네트웍 장애로 인한 그룹 분할 및 병합에 따라 갱신되는 키의 수가 모두 $O(\log n)$을 보장한다. [표 1]에서 최악의 경우에 대하여 기존의 키-트리를 이용한 방법들과 본 논문에서 제시한 방법을 비교하였다.

[16]에서는 구성원의 가입과 탈퇴의 수가 동일하다는 가정 하에, 키-서버가 수행하는 키의 암호화

[표 1] 갱신되는 키 개수

사건 / 방식	가입	탈퇴	분할	병합
일반 키-트리를 이용한 방식	$O(n)$	$O(n)$	-	-
정형적 방법에 의한 방식	$O(n)$	$O(n)$	$O(n)$	$O(m+\log n)$
AVL-트리를 이용한 방식	$O(\log n)$	$O(\log n)$	$O(m \cdot \log n)$	$O(m+\log n)$
제안된 방식	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

수와, 구성원이 수행하는 키 갱신 메시지의 복호화 수의 평균을 분석하였다. 이 분석에 따르면, 키-트리의 차수가 d이고 높이가 h일 때, 키-서버와 각 구성원은 각각 $(d+2)(h-1)/2$와 $d/(d-1)$번의 암호화와 복호화를 수행하며 $d=4$일 때 키-서버의 계산량이 최소화된다. 본 논문에서 제안한 방법에서는 트리의 각 레벨에서 최대 2개의 키가 갱신될 수 있으므로, 서버의 암·복호화 수는 $(d+2)(h-1)$가 된다. 기존의 키-트리는 대부분 $d=2$인 키-트리를 사용하고 본 논문에서 제안하는 키-트리는 $2 \le d \le 4$ 이므로, 트리의 모든 노드의 차수가 2일 때 서버의 계산량은 기존의 키-트리와 같으며 차수가 4일 경우 최적이 된다. 전술한 분석에 의하여 구성원의 제산량도 이진 키-트리를 사용할 경우보다 효율적임을 알 수 있다.

갱신되는 키의 수는 키-서버가 생성해야 하는 랜덤 수를 의미하며, 키-서버의 암호화 수는 전송되는 메시지의 크기를 말한다. 따라서 제안한 방식은 갱신되는 키의 수, 서버의 랜덤 수 생성과 암호화 수, 사용자의 복호화 수 및 통신량에 있어서 기존의 키-트리 방식보다 효율적이다.

4.4 안전성 분석

동적인 특성을 가지는 그룹에 대한 키 관리에서 제공해야 하는 보안 특성은 다음과 같다.[12]

- group key secrecy : 그룹키는 그룹 구성원들만이 공유하여야 한다.
- forward secrecy : 과거 그룹키들 일부를 안다고 해도, 이로부터 그 이후의 그룹키는 알아낼 수 없다.
- backward secrecy : 과거 그룹키들 일부를 안다고 해도, 이로부터 그 이전의 그룹키는 알아낼 수 없다.

group key secrecy는 그룹키의 특성상 제공해야 하는 기본적인 성질이다. forward secrecy는 합법적인 구성원이 탈퇴(혹은 그룹 분할)할 경우, 탈퇴(혹은 분할)한 구성원들이 이전에 공유하고 사용하던 그룹키로부터 탈퇴(혹은 분할) 이후의 그룹키를 생성할 수 없도록 하기 위한 조건이며, backward secrecy는 새로운 구성원이 합법적으로 가입(혹은 그룹 병합)하여 그룹키를 공유하게 되더라도 가입(혹은 병합) 이전의 그룹키는 알아낼 수 없도록 하기 위한 조건이다. 동적 그룹에서 확장성의 문제는 구성원 변화에 대한 효율적인 키 갱신의 문제라고 할 수 있다.

제안한 방식은 구성원 변화에 대하여 기존의 키-트리를 이용한 방식들과 동일한 방법으로 키 갱신을 수행한다. 즉, 구성원이 가입하거나 그룹이 병합될 경우, 새로운 구성원(들)이 소유하게 되는 키-트리 상의 모든 키들을 변경함으로써 새 구성원(들)이 기존의 키를 알아낼 수 없도록 한다. 이와 마찬가지로, 구성원이 탈퇴하거나 그룹의 분할이 일어날 경우, 이 구성원(들)이 다른 구성원들과 공유하던 모든 키를 갱신하고, 나머지 구성원들만이 공유하고 있는 키로 생성된 키들을 암호화하여 전송하기 때문에 탈퇴나 분할 이후의 키를 알아낼 수 없다. 따라서 합법적인 구성원만이 현재의 그룹키를 공유할 수 있게 되므로, 제안한 방식은 세 가지 보안 조건을 모두 만족한다.

V. 결 론

그룹통신을 안전하게 유지하기 위한 방편으로서 그룹 구성원만이 그룹키를 공유하고 이를 이용하여 비밀 통신을 한다. 이 공유키를 구성원에게 안전하게 배포하고 관리하는 것은 안전한 그룹통신에서의 핵심이라 할 수 있다. 구성원이 동적으로 유지되어야 하는 큰 규모의 그룹을 효율적으로 관리하기 위해서 키-트리를 사용한다. 본 논문에서는 키-트리를 이용한 그룹키 관리의 효율성을 보장하기 위하여 비교적 구현이 간단한 *(2,4)-트리*를 이용하였다. *(2,4)-트리*의 이용은 구성원의 가입과 탈퇴에 따른 키 갱신을 $O(\log n)$에 수행할 수 있도록 보장할 뿐

아니라, $O(\log n)$의 복잡도로 임의의 위치에 부트리를 삽입하거나 분리할 수 있는 장점을 가지고 있다. 만약, 키-트리의 구조에 실제 네트웍의 구성을 반영한다면, 네트웍 장애로 인한 구성원 집합의 탈퇴에도 효율적으로 키 갱신을 할 수 있다. 본 논문에서는 *CBT*를 이용하여 키-서버가 네트웍 구성정보를 수집하고 유지하여 이를 키-트리 구성에 반영함으로써 네트웍 장애 시에도 $O(\log n)$의 복잡도로 키 갱신을 할 수 있는 방법을 제시하였다.

참 고 문 헌

[1] A. Ballardie, "Scalable Multicast Key Distribution," *IETF* RFC 1949, 1996.

[2] A. Ballardie, "Core Based Trees(CBT) Multicast Routing Architecture," *IETF* RFC 2201, 1997.

[3] M. Burmester and Y. Desmedt, "A Secure and Efficient Conference Key Distribution System," EUROCRYPT, 1994.

[4] CISCO, "Internet Protocol(IP) Multicast," http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/ipmulti.htm.

[5] W. Fenner, "Internet Group Management Protocol, Version 2," IETF RFC 2236, 1997.

[6] 김봉한, 이채광, "그룹통신을 위한 멀티캐스트 키 분배 프로토콜 설계 및 검증," *통신정보보호학회*논문지 제10권, 제2호, 2000.

[7] Y. Kim, A. Perrig, G. Tsudik, "Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups," *ACM* CCS 2000, 2000.

[8] K. Mehlorn, *Data Structures and Algorithms 1 : Sorting and Searching*, Springer-Verlarg, 1984.

[9] M. J. Moyer, J. R. Rao, P. Rohatgi, "Maintaining Balanced Key Trees for Secure Multicast," IRTF Internet Draft, June 1999.

[10] K. Matsuura, Y. Zheng, and H. Imai, "Compact and Flexible Resolution of CBT Multicast Key-Distribution," Proc. of the 2nd International Conference on Worldwide Computing and Its Applications, 1998.

[11] A. Perrig, "Efficient Collaborative Key Management Protocols for Secure Autonomous Group Communication," International Workshop on Cryptographic Techniques and E-Commerce, 1999.

[12] A. Perrig, D. Song, J. D. Tygar, "ELK, a New Protocol for Efficient Large-Group Key Distribution," 2001 IEEE Symposium on Security and Privacy, 2001.

[13] O. Rodeh, K. P. Birman, D. Dolev, "Optimized Group Rekey for Group Communication Systems," Network and Distributed Systems Security 2000, 2000.

[14] O. Rodeh, K. P. Birman, D. Dolev, "Using AVL Trees for Fault Tolerant Group Key Management," Technical Report, Institute of Computer Science, The Hebrew Univ. of Jerusalem, October 2000.

[15] Stardust Technologies, "Introduction to IP Multicast Routing," Techinical Report, Stardust Technologies, 1997.

[16] C. K. Wong, M. Gouda, S. S. Lam, "Secure Group Communications using Key Graphs," Proc. of ACM SIGCOMM, 1988.

〈著者紹介〉

조 태 남 (Tae-Nam Cho) 정회원
1986년 2월 : 이화여자대학교 전자계산학과 학사
1988년 2월 : 이화여자대학교 대학원 전자계산학과 석사
1988년 3월~1996년 3월 : 한국전자통신연구원 선임연구원
1998년 3월~현재 : 이화여자대학교 과학기술대학원 컴퓨터학과 박사과정
〈관심분야〉 정보보호, 암호프로토콜, 알고리즘 설계

이 상 호 (Sang-Ho Lee) 종신회원
1979년 2월 : 서울대학교 계산통계학과 학사
1981년 2월 : 한국과학기술원 전산학과 석사
1987년 8월 : 한국과학기술원 전산학과 박사
1983년 9월~현재 : 이화여자대학교 컴퓨터학과 교수
2000년~현재 : 한국정보과학회 총무이사, 정보보호연구회 부위원장
〈관심분야〉 정보보호, 암호프로토콜, 정보은닉, 알고리즘 설계, 그래프 이론 및 드로잉, 계산복잡도 이론, 계산기하