신뢰성 있는 그룹키 관리를 위한 키 복구 메커니즘

(A Key Recovery Mechanism for Reliable Group Key Management)

조 태 남 † 김 상 희 †† 이 상 호 †††
(Taenam Cho) (Sanghee Kim) (Sang-Ho Lee)
채 기 준 ††† 박 원 주 †††† 나 재 훈 ††††
(Kijoon Chae) (Wonjoo PARK) (Jaehoon Nah)

요 약 그룹 통신의 보안이나 과금을 위해 사용되는 그룹키는 멤버쉽의 변동이 있을 때 키 갱신 메시지를 통하여 변경해 주어야 한다. 만약 이 메시지를 분실할 경우 그룹 데이타의 복호화가 불가능하며 과금도 어려워지므로, 신뢰성 있는 그룹키 갱신 메시지의 전달뿐 아니라, 분실된 키의 복구는 매우 중요한 문제이다. 특히 멤버의 로그오프 동안 분실된 메시지는 실시간 복구 요청이 불가능하므로 이에 대한 대책이 필요하다. 키 분배 센터가 전송된 메시지를 저장하고 이를 재전송하는 방법은 키 분배 센터의 많은 저장 공간을 요구하고, 불필요한 키들을 전송하고 복호화하도록 하며, 저장되지 않은 메시지는 복구가 불가능하다. 본 논문에서는 키 갱신 메시지의 분실이나 멤버의 로그인시에 발생할 수 있는 문제점을 분석하고 효율적으로 그룹키 및 보조키들을 복구하는 방법을 제시한다. 제안한 방법에서는 키-트리에 저장된 갱신 정보를 이용하여 복구가 요구된 그룹키뿐 아니라 이로부터 복호화가 불가능한 그룹키 및 보조키들을 계산하고 전송한다. 또한 간단한 그룹키 생성 방법을 제시함으로써, 메시지를 저장하지 않고 임의의 그룹키를 복구할 수 있도록 하였으며, 불필요한 보조키들의 전송 및 복호화를 제거하였다.

키워드 : 그룹키 관리, 보안, 멀티캐스트, 신뢰성, 키 복구

Abstract For group security to protect group data or to charge, group keys should be updated via key update messages when the membership of group changes. If we lose these messages, it is not possible to decrypt the group data and so this is why the recovery of lost keys is very significant. Any message lost during a certain member is logged off can not be recovered in real-time. Saving all messages and resending them by KDC (Key Distribution Center) not only requests large saving spaces, but also causes to transmit and decrypt unnecessary keys. This paper analyzes the problem of the loss of key update messages along with other problems that may arise during member login procedure, and also gives an efficient method for recovering group keys and auxiliary keys. This method provides that group keys and auxiliary keys can be recovered and sent effectively using information stored in key-tree. The group key generation method presented in this paper is simple and enable us to recover any group key without storing. It also eliminates the transmissions and decryptions of useless auxiliary keys.

Key words : group key management, security, multicast, reliability, key recovery

· 본 연구는 한국전자통신연구원 정보보호연구본부 네트워크보안연구부 위탁연구과제에 의한 것임
† 정 회 원 : 이화여자대학교 컴퓨터학과
tncho@ewha.ac.kr
†† 비 회 원 : 이화여자대학교 컴퓨터학과
kshee78@nate.com
††† 종신회원 : 이화여자대학교 컴퓨터학과 교수
shlee@ewha.ac.kr
kjchae@ewha.ac.kr
†††† 비 회 원 : 한국전자통신연구원 네트워크보안연구부 연구원
wjpark@etri.re.kr
jhnah@etri.re.kr
논문접수 : 2002년 12월 12일
심사완료 : 2003년 8월 22일

1. 서 론

그룹 통신에서 가장 중요시되고 있는 보안 문제는 기밀성이다. 사내 비밀 회의나 분산 시뮬레이션 등에서는 데이타의 기밀성 유지를 위해 사용자의 접근을 제어해야 하고, 유료 영상 서비스나 유료 인터넷 방송 등에서는 과금을 위하여 접근 제어 및 데이타의 기밀성이 만족되어야 한다. 이를 해결하기 위한 방법으로서, 허용된 사용자만이 그룹 데이타에 접근할 수 있도록 공유한 그룹키를 이용하여 암호화 통신을 한다[1]. 그러므로 인증된 그룹의 멤버만이 안전하게 그룹키를 공유해야 하고, 멤버의 가입이나 탈퇴가 발생할 경우에는 그룹키를 변경함으로써 자격을 상실하거나 취득하기 이전의 부정직한 멤버가 그룹 데이타를 얻을 수 없도록 해야 한다. 이를 위해서는 멤버가 가입하거나 탈퇴할 경우에 중앙의 관리자가 키 갱신 메시지를 통하여 멤버들에게 갱신된 그룹키를 전송한다. 키 갱신 메시지에는 갱신된 그룹키 뿐 아니라 효율적으로 그룹키를 갱신하는데 사용되는 보조키들이 포함된다. 만약 멤버가 키 갱신 메시지를 분실하면, 갱신된 그룹키를 얻을 수 없기 때문에 그룹 데이타를 복호화할 수 없을 뿐 아니라, 이후에 전송되는 키 갱신 메시지도 복호화가 불가능할 수 있다. 이러한 키 갱신의 신뢰성에 관한 문제는 해결해야 할 문제로 논의되고 있으나[2,3], 이에 대한 연구는 미비하다. 현재는 키 갱신 메시지의 수신율을 높이기 위한 몇몇 방안들이 제시되었으나, 분실되었을 때 복구하는 방법에 대해서는 제시된 바가 없다. 특히 멤버의 로그인과 아웃으로 인해 발생하는 문제에 대해서는 연구되지 않고 있다. 본 논문에서는 중앙의 키 분배 센터(KDC: Key Distribution Center)가 키를 생성하여 멤버들에게 분배하는 구조를 기반으로 하여, 통신의 지연이나 멤버의 로그 아웃으로 인하여 메시지 분실 이후, 키 갱신이 발생한 경우에도 키를 복구할 수 있는 방안을 제시한다.

본 논문은 6장으로 구성된다. 2장에서는 그룹키 관리가 만족해야 하는 요구사항을 기술한다. 3장에서는 관련 연구를 기술하고 4장에서는 키 복구 메카니즘을 제안한다. 5장에서는 안전성, 신뢰성 및 효율성을 분석하고 6장에서는 결론 및 향후 연구 과제를 기술한다.

2. 요구사항

인증된 그룹의 멤버만이 안전하게 그룹키를 공유해야 하고, 멤버의 가입이나 탈퇴가 발생할 경우에는 그룹키를 변경함으로써 자격을 상실하거나 취득하기 이전의 부정직한 멤버가 그룹 데이타를 얻을 수 없도록 해야 한다. 이를 위해서는 다음과 같은 요구 조건이 만족되어야 한다[4-6].

- GKS(Group Key Secrecy): 도청자가 그룹키를 알아내는 것은 계산상 불가능해야 한다.
- FS(Forward Secrecy): 일정 기간동안 생성된 그룹키들을 안다고 하더라도, 그 이후에 생성되는 그룹키를 유도할 수 없어야 한다.
- BS(Backward Secrecy): 일정 기간동안 생성된 그룹키들을 안다고 하더라도, 그 이전에 생성된 그룹키를 유도할 수 없어야 한다.

GKS를 만족시키기 위해서는 키의 생성 및 전송 안전하게 이루어져야 하고, FS와 BS를 위해서는 멤버가 가입하거나 탈퇴하여 그룹의 멤버가 변경될 때마다 그룹키가 변경되어야 한다. 따라서 멤버의 가입과 탈퇴가 잦은 그룹에서는 멤버쉽 변동에 따른 그룹키를 갱신 메시지가 매우 빈번하게 전송된다. 그룹 데이타를 수신한 멤버라도 그룹키 갱신 메시지를 제대로 수신하지 못하면 수신한 그룹 데이타를 복호화할 수 없다. 키 갱신 메시지의 분실 및 지연을 고려하여 수신자는 데이타를 저장하는 버퍼를 사용하고, 데이타를 버퍼링할 수 있는 기간동안 키를 수신할 수 있어야 한다[7-10]. [10]에서는 다음과 같은 요구 조건을 제시하면서 키 갱신 메시지를 높은 확률로 멤버들에게 전달하기 위한 기법을 제안하였다.

- 신뢰성(reliability): 그룹의 규모가 커도 멤버는 자신의 모든 암호화된 새 키들을 수신할 수 있어야 한다.
- 약-실시간성(soft real-time): 높은 확률로 다음 키 갱신 이전에 새로운 키들을 전달할 수 있어야 한다.

그룹키를 분배하는 가장 간단한 방법은 KDC가 1:1로 각 멤버들에게 그룹키를 유니캐스트 하는 것이다. 그러나 그룹의 규모가 클 경우에 매우 비효율적이기 때문에 확장성 제공을 위하여 보조키(KEK: Key Encryption Key)들을 이용하여 멀티캐스팅을 이용한 키 갱신을 수행할 수 있다. 대표적 방법은 키-트리(key-tree)를 이용하는 것이다[1]. KEK들은 그룹 데이타의 암·복호화에 사용되는 것이 아니라 그룹키나 또 다른 KEK들을 암호화하여 멤버들에게 안전하게 전송하기 위해 사용되는 키들이다. 만약 어떤 KEK들이 더 이상 그룹키나 다른 KEK의 암호화에 사용되지 않는다면 멤버가 반드시 수신할 필요가 없다. 또한 정당한 멤버가 그룹의 구성원일 동안 분실한 키는 반드시 복구할 수 있어야 하며, 로그아웃 기간동안 분실한 키에 대해서도 키를 복구할 수 있도록 보장하여야 한다. 따라서 본 논문에서는 다음과 같이 신뢰성의 조건을 수정하고 키 복구에 대한 조건을 추가하였다.

- 신뢰성: 멤버가 수신한 그룹 데이타의 복호화에 필요한 모든 새 키들을 수신할 수 있어야 한다.
- 키 복구(key recovery): 합법적인 멤버는 기간과 상관

없이 분실한 그룹키와 필요한 KEK들을 KDC로부터 재전송 받을 수 있어야 한다.

3. 관련 연구

본 장에서는 제안하는 방법에서 사용하는 키-트리를 소개하고, 신뢰성 제공을 위한 연구들을 소개한다.

3.1 키-트리를 이용한 그룹키 관리

규모가 큰 그룹에서 KDC가 모든 멤버들에게 새로운 키를 1:1로 전송하는 것보다 효율적으로 그룹키를 갱신하기 위한 여러 가지 방법들이 연구되었다. 그 중에서 현재 가장 널리 사용되는 방법은 키-트리(LKH: Logical Key Hierarchy)로서, 1998년 [1]에서 처음 소개되었으며 [11]에서는 차수가 2인 이진트리를 사용하였다. 키-트리의 중요한 개선점은 키 갱신 복잡도를 $O(n)$에서 $O(\log n)$ 줄이는 것인데, 이것은 키-트리가 균형 잡혀 있다는 가정 하에 이루어진다. 키-트리의 균형 유지를 위해서, 노드들의 깊이 정보를 이용한 방법[12] AVL-트리를 이용하는 방법[13], 혹은 (2,4)-트리를 이용하는 방법들이[6] 제안되었다. 키값들은 의사 난수 생성기(pseudo random number generator)를 이용하여 생성하거나[6,11-13], 자식 노드키를 이용하여 생성하는 방법들이[5,14,15] 있다.

본 논문에서 제안하는 방법은 다른 키-트리로 쉽게 적용이 가능하므로, 균형 잡힌 이진 키트리를 가정하며 키는 의사 난수 생성기로 생성한다고 가정한다. 트리의 각 노드에는 유일한 키가 할당되어 있고, 단말 노드는 멤버에게 1:1로 대응된다. 모든 멤버는 자신의 단말 노드로부터 루트에 이르는 경로상의 키들을 소유하며, 루트키는 그룹키이다. 만약 멤버가 가입하거나 탈퇴하면 변동된 멤버가 소유하는 키들을 변경하여 배포한다. 그림 1에서 m_7이 탈퇴하면, $k_7, k_{7,8}$은 제거되고 $k_{5,8}$, $k_{1,8}$, $k_{1,16}$은 $k_{5,8}$, $k_{1,8}'$, $k_{1,16}'$로 갱신된다. 갱신된 키들은 메시지 $\{\{k_{5,8}'\}_{k_{5,6}}, \{k_{5,8}'\}_{k_8}, \{k_{1,8}'\}_{k_{1,4}}, \{k_{1,8}'\}_{k_{5,8}'}, \{k_{1,16}'\}_{k_{1,8}'}, \{k_{1,16}'\}_{k_{9,16}}\}$를 통해 멀티캐스트 된다. 여기서 $\{M\}_{key}$는 키 key를 이용하여 M을 암호화한 메시지를 의미한다. 멤버의 가입으로 변경된 키는 자식 노드키가 아니라 기존의 키로 암호화하여 전송된다[1].

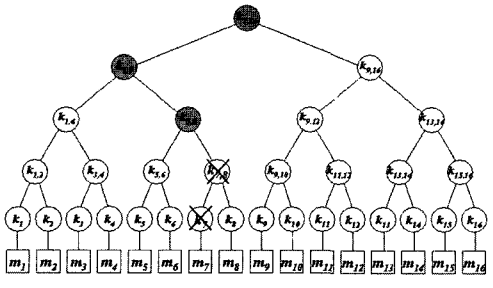

그림 1 이진 키-트리의 구조

3.2 신뢰성 제공을 위한 그룹키 관리

신뢰성 있는 그룹키 분배 방법은 Keystone에서[7] 처음 제안되었다. Keystone에서는 FEC(Forward Error Correction) 코드를 이용한 UDP(User Datagram Protocol) 전송을 제안하였다. [10]에서는 Keystone 방식이 연집 에러(burst error)에 대처하지 못함을 지적하면서, 키 갱신 메시지를 여러 개의 블록으로 분할하여 전송하는 방법을 제시하였다. 각 멤버에게 필요한 모든 키들은 1개의 블록에 포함되도록 구성하고, 각 블록마다 RSE(Reed Solomon Error) 코드를 계산하고 블록들을 중복 전송한다. [9]에서는 [10] 방식에서 키-트리의 상위 레벨에 있는 키일수록 많은 멤버들에게 공유된다는 사실을 감안하여, 레벨에 따라 몇 등급으로 나누어 가중치를 부여한다. 그리고 가중치에 따라 중복 전송하는 횟수를 늘리는 방식이다. 그림 2에서 7개의 삼각형은 분할 메시지 블록을 나타내고, 점선으로 표시된 3개의 사각형은 서로 다른 가중치를 블록을 의미한다.

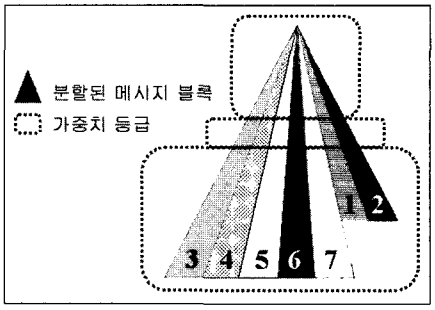

그림 2 메시지 구성 방법

기술한 세 가지 방법들은 약-실시간성으로 멤버들의 키 갱신 메시지 수신율을 높이는 것이 목표이다. 즉, 키 분실의 가능성을 완전히 배제할 수는 없으며 분실되었을 경우에는 재전송을 반복한다. 그러나 그러한 재전송으로 지연이 발생한 상태에서 또 다른 키 갱신 이벤트가 발생할 경우에 대해서는 언급하지 않고 있다. 이러한 상황은 멤버가 로그 아웃한 상태에서 버퍼링 되어 있어야 할 키 갱신 메시지가 분실되었을 때와 동일한 문제이다. 이 두 가지 문제의 공통점은 멤버가 분실한 메시지가 KDC가 전송한 최근 키 갱신 메시지가 아니라는 점이다.

ELK는[5] 키 복구를 위해 데이타에 hint라는 키의 검증 정보를 실어 보낸다. 키 갱신 메시지를 분실한 멤버는 분실한 키들 중 중요 키들의 일부를 전수 조사하여 계산해 내고, hint와 비교하여 올바른 키인지를 확인

한다. 나머지 비 중요 키들은 KDC로부터 유니캐스트로 받는다. 이것은 통신량을 절약하기 위해 고안된 방법으로서, 키를 복구하기 위하여 전수 조사하는 멤버의 계산량이 많다.

4. 키 복구 메커니즘 제안

최근 키 갱신 메시지가 아닌 메시지를 복구하는 기본적인 방법은 메시지를 재전송하는 것이다. 재전송을 위해서는 KDC는 키 복구 지원이 가능한 메시지 개수 w를 정하여 최근 w개의 키 갱신 메시지를 버퍼에 저장하고, 키-트리는 변경된 최종 상태만 유지한다. 멤버의 키 복구 요청이 있으면 요청한 메시지가 버퍼에 존재하는지 검사하고, 존재하면 이를 요청 멤버에게 유니캐스트한다. 만약 버퍼에 존재하지 않으면, 복구 불가로 응답한다. 이 방법은 매우 간단하지만, 버퍼에 저장된 메시지 이전의 키는 복구가 불가능하다. 또한 분실된 메시지를 복구해야만 이후의 메시지들도 복호화할 수 있는 경우가 발생하는데, 분실된 메시지나 이후의 메시지에는 더 이상 유용하지 않은 KEK들이 포함될 수 있다. 즉, 멤버들은 유효하지 않은 KEK들을 복호화하는 비효율성을 감수해야 한다. 자세한 분석은 4.2절에서 다룬다.

본 장에서는 최근의 키 갱신 메시지뿐 아니라 이전의 메시지 분실시에도 효율적으로 복구할 수 있으며, 멤버가 불필요한 KEK들의 복호화 과정을 생략할 수 있는 방법을 제안한다. 4.1절에서는 사용되는 용어를 정의하고, 4.2절에서는 개선을 위한 분석 사항들을 기술한다. 4.3절에서는 키 생성 방법, 수정된 키-트리의 구조와 이를 이용해서 키 복구 메시지를 생성하는 알고리즘을 기술한다.

4.1 용어 정의

- $event_i$: i번째 키 갱신을 유발시킨 멤버의 동작으로서, *join*이나 *leave*이다.
- T_i: $event_i$로 인하여 i번째 갱신이 일어난 후의 키-트리이다.
- $ReKey_i$: T_{i-1}에서 T_i로 변경된 키를 멀티캐스트하는 키 갱신 메시지이다.
- GK_i: $ReKey_i$에 의해 갱신된 그룹키로서, T_i의 루트키이다. i는 그룹키의 버전과 같다.
- m_i: 식별자가 i인 멤버이다.
- k_i: 멤버 m_i와 KDC만의 공유키이다(그림 1, 3 참조).
- $k_{i,j}$: 키-트리에서 DFS(Depth First Search)로 멤버들을 탐색했을 때 m_i부터 m_j까지의 멤버들이 공유하는 키이다(그림 1, 3 참조).
- w: 키 복구 윈도우로서, 복구해 줄 수 있는 최근 키 갱신 메시지 갯수이다.
- $level(node)$: 키-트리의 노드 $node$의 레벨이다. 루트의 레벨은 1이고, 노드의 레벨은 부모 노드의 레벨보다 1이 크다.
- $path_q^r$: T_r에서 m_q의 단말 노드로부터 루트까지의 경로이다.
- l_q^r : T_r에서 m_q와 $event_r$을 유발시킨 멤버 m_x와의 NCA(Nearest Common Ancestor) 키이다. 즉, $path_q^r$과 $path_x^r$가 만나는 첫 노드이다. 예로, (그림 1)이 $m_x = m_9$가 가입하여 생성된 키-트리 T_r이라고 하면 l_{12}^r은 $k_{9,12}$에 해당하는 노드이다.
- $PRF_{key}(s)$: key를 키로 하여 s를 난수로 매핑하는 의사 난수 생성기(Pseudo Random Function)이다[16].

본 장에서는, 키 복구 요청을 한 멤버를 m_q라 하고, 분실한 키 갱신 메시지와 그룹키를 각각 $ReKey_r$, GK_r이라 하며, KDC가 m_q로부터 키 복구 요구를 받았을 당시의 그룹키를 GK_c라 표시한다.

4.2 분석 사항

개선된 키 복구 방법을 설계하기 위하여, LKH를 기반으로 분실된 키 갱신 메시지의 복구에 관한 문제점을 분석한 결과 다음과 같은 사항이 관찰되었다.

(1) 키-트리의 어떤 노드키가 $event_i$로 인하여 갱신되었다면, 이 노드의 선조 노드키들도 함께 갱신되었다.

(2) $ReKey_r$을 분실한 멤버 m_q는 다음과 같은 경우에 $GK_r, \ldots, GK_{s-1}$($r < s \le c+1$)을 복구하지 못한다:

$r < i < s$인 각 i에 대하여 $level(l_q^r) > level(l_q^i)$이거나 $event_i = join$일 때이다. 즉, $ReKey_r$에 m_q의 KEK가 들어 있고, 그 이후 s번째 이전까지는 $path_q^i$ 상에서 l_q^r보다 작은 레벨의 키만 갱신되었거나, 멤버의 가입으로 인하여 갱신되었을 때이다. m_q가 GK_i를 복호화되지 못하는 이유는, GK_i나 복호화에 필요한 KEK들이 $ReKey_r$ 내에 포함되었기 때문이다. 예로, 그림 3에서 m_3가 가입하면 $k_3, k_{3,4}$가 추가되고 $k_{1,4}, k_{1,8}$은 각각 $k_{1,4}', k_{1,8}'$이 된다. m_1이 이에 대한 키 갱신 메시지 $ReKey_r$=

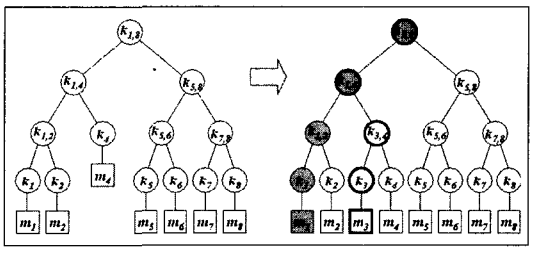

그림 3 멤버 가입으로 인한 키-트리 갱신

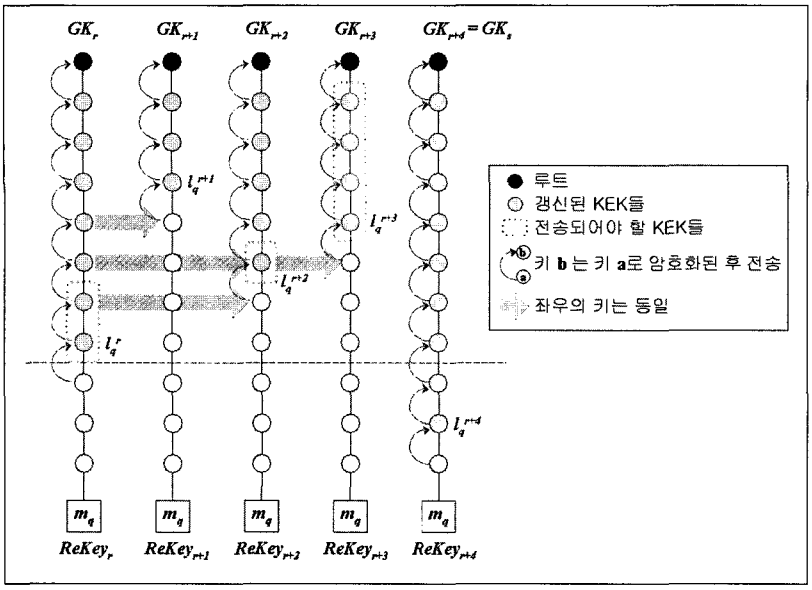

그림 4 멤버의 탈퇴로 인한 키 갱신 체인

$\{\{k_{3,4}\}_{k_4}, \{k_{1,4}'\}_{k_{1,4}}, \{k_{1,8}'\}_{k_{1,8}}\}$를 분실한 상태에서 m_7이 탈퇴하여, $ReKey_{r+1}$ = $\{\{k_{5,8}'\}_{k_{5,6}}, \{k_{5,8}'\}_{k_8}, \{k_{1,8}''\}_{k_{1,4}'}, \{k_{1,8}''\}_{k_{5,8}'}\}$을 수신한다면 $k_{1,8}''$도 복호화하지 못한다.

그림 4는 $path_q^i$에서 멤버 탈퇴로 인한 GK_i 갱신 메시지인 $ReKey_i$에 포함된 키들을 표시한 것이다. 멤버의 탈퇴로 인한 키 갱신 메시지는 경로 상에서 변경되지 않은 최상위(레벨값이 가장 작은) 키에서 시작하여, 차상위(레벨값이 하나 작은) 키를 암호화하는 체인을 형성한다. 그림 4의 예에서 보는 바와 같이 $GK_{r+1}, GK_{r+2}, GK_{r+3}$을 위한 키 체인은 분실된 $ReKey_r$에서 갱신된 키부터 시작하기 때문에 복호화가 불가능하다. 멤버의 가입으로 인한 키 갱신 메시지는 체인을 형성하지 않고, 각 새로운 키들은 동일한 노드의 이전 키값으로 암호화되므로, 분실 메시지를 복구하지 않으면 $level(l_q^r)$ 이하의 키들은 복호화가 불가능하다. 그러나, $level(l_q^r)$보다 큰 레벨의 키들은 $ReKey_r$과 상관없이 항상 복호화가 가능하기 때문에 $level(l_q^r)$ 이하의 키들은 키 복구에 고려하지 않아도 된다.

(3) $ReKey_r$을 분실한 멤버 m_q는 다음과 같은 경우에 GK_s($r < s \le c$)을 복구할 수 있다:

$event_i = leave$이고 $level(l_q^r) \le level(l_q^s)$이면, 그림 4의 예에서와 같이 $ReKey_s$ ($s = r+4$)의 키 체인은 $ReKey_r$에서 분실되지 않은 키부터 시작하므로, $ReKey_r$과 무관하게 복호화할 수 있다.

(4) 복구되어야 할 키 갱신 메시지에 포함된 KEK가, 분실된 메시지 이후에도 변경되었다면 이들은 복구할 필요가 없다. 이 키들은 분실한 그룹키나 이후의 그룹키 및 KEK들을 복호화하기 위한 것이므로, 다시 변경되었다면 유효성을 상실한 것이다. 따라서 그림 4에서 보는 바와 같이 $ReKey_r$에서 분실되지 않은 키로 복구할 수 있는 키 갱신 메시지 $ReKey_s$가 이미 전송되었으면, $ReKey_r$ 및 $ReKey_i$ ($r < i < s$)에 있는 KEK들은 불필요하다. 그러나 그러한 메시지가 존재하지 않으면, 점선으로 표시된 KEK들은 유효하며, 이것은 $ReKey_r$에서 분실된 KEK들의 최근 갱신값 들이다. 즉, $path_q^c$에서 $level(l_q^r)$ 이하의 레벨에 있는 키들이다.

기술한 분석사항은 키값이 자식 노드키로부터 생성되는 스킴들에도[5,14,15] 등에서도 유사하다. 멤버의 가입을 대비하여 별도의 메시지 없이 주기적으로 갱신하는 경우를 제외하면, 변경되는 키 정보를 형제 노드키 정보로 암호화하여 전송하는 것이 다르다. 그러나 이 메시지를 분실할 경우에도, 분실한 키의 부모 키를 생성할 수 없으므로 (2)와 동일한 결과를 초래한다.

4.3 키 복구 알고리즘

KDC가 전송된 메시지를 저장하거나, 과거의 키값들을 저장하지 않으면서 멤버들이 필요한 임의의 키를 복구할 수 있도록 한다. 특히 멤버가 분실한 키 메시지가 최근의 것이 아닌 경우에 복구가 불필요한 KEK들의 전송을 배제하고 최소의 그룹키들과 KEK들만을 전송함으로써, 멤버들의 처리시간을 줄인다.

• 키 식별자

모든 키는 생성시부터 유일한 식별자를 가지며 키 값이 변경되더라도 식별자는 변하지 않는다. 그룹키는 다음과 같이 생성한다.

• 그룹키 생성 방법 및 갱신 방법

KDC는 임의의 시간에 임의의 버전의 그룹키를 생성할 수 있으면서, GKS, FS와 BS를 만족하도록 다음과 같이 생성한다. $Mkey$는 KDC만 알고 있는 비밀키값이고, 그룹키는 GK_1부터 시작하여 갱신될 때마다 버전이 1씩 증가한다.

$$GK_i = PRF_{Mkey}(i)$$

• 키-트리의 노드 구조

키-트리의 각 노드는 키 식별자, 키값, 크기 w의 배열 $flag[]$와 $last$를 포함한다. 2비트 $flag[\,i \bmod w]$는 해당 노드키를 변경시킨 $event_i$를 저장하고, $last$는 해당 노드키가 변경된 최근 이벤트의 색인이다.

• 멤버의 키 저장 방법

각 멤버는 키-트리의 해당 단말 노드로부터 루트까지의 경로 상에 있는 키들을 저장한다. 키들은 고유 식별자와 함께 하위 레벨부터 순서화 되어있다.

• 키 복구 알고리즘

복구 알고리즘은 4.2절의 분석 결과를 토대로, 복구에 필요한 최소한의 키들만 전송하도록 설계하였다. m_q로부터 $ReKey_r$에 대한 복구 요청을 수신하면, l_q^r을 계산하고 4.2절의 (3)을 만족하는 최소 $s(\,r<s\le c)$를 찾는다. 즉, $event_i = leave$이고 $level(l_q^r) \le level(l_q^s)$인 s를 찾으면, $GK_i = PRF_{Mkey}(i)$을 이용하여 $GK_r, \ldots, GK_{s-1}$을 계산하여 m_q에게 전송한다. $ReKey_r$에 포함되어 있던 KEK는 4.2절 (4)에서 분석된 바와 같이 더 이상 유효하지 않기 때문에, 복구할 필요가 없으므로 재전송하지 않는다 ($ReKey_{r+1}, \ldots, ReKey_{s-1}$의 KEK들도 유효하지 않으므로, m_q는 이들을 복호화하지 않는다). 멤버가 $ReKey_s$를 복호화할 수 있다면, m_q는 그 이후의 메시지를 모두 복호화할 수 있게 된다. 만약 그러한 s가 존재하지 않으면, m_q는 $ReKey_r$부터 그 이후의 모든 키 갱신 메시지를 복호화하지 못하게 된다. 그러므로 $GK_r, \ldots, GK_{c-1}$와 $path_q^c$에서 $level(l_q^r)$ 이하의 레벨에 있는 KEK들을 전송한다. 왜냐하면 4.2절의 (4)에서 분석된 바와 같이 이 KEK들은 m_q에게 아직 유효하지만 $ReKey_r$로 인하여 복호화가 불가능한 키들이기 때문이다.

l_q^r와 l_q^s는 $flag[]$를 이용하여 계산한다. KDC는 m_x가 $event_i$를 발생하여 키 갱신이 일어날 때마다, $path_x^i$에서 변경되는 노드들의 $flag[\,i \bmod w]$를 $event_i$로 설정한다. KDC가 m_q로부터 $ReKey_r$에 대한 복구 요청을 수신하면, 키-트리 T_c에서 m_q의 단말 노드 n_q로부터 루트로 올라가면서 $path_q^c$상에서 $flag[\,r \bmod w]$가 $leave$이거나 $join$으로 설정되어 있는 최초의 노드를 찾는다. 이 노드가 l_q^r이다. l_q^s도 유사한 방법으로 찾는다. 즉, l_q^s는 n_q와 l_q^r 사이에서, $flag[\,i \bmod w]$가 $leave$이고 $r<i$인 노드들 중 i가 최소인 노드이다.

만약 복구를 요청한 메시지가 복구 윈도우 이전의 것이라면, 키-트리의 $flag[]$들은 l_q^r에 대한 정보를 가지고 있지 않다. 따라서 이 경우에는 $l_q^r = n_q$로 간주하여 $GK_r, \ldots, GK_{c-1}$와 $path_q^c$을 전송한다. 전송되는 키는 m_q와 KDC와의 공유키인 k_q로 암호화하여 유니캐스트 된다.

$flag[]$는 w를 주기로 값을 재사용하기 때문에 이전에 설정된 값이 남아있게 된다. 그러므로, 키-트리가 갱신될 때나 키 복구 요구를 받아 트리를 검색할 때 이전에 설정된 값을 지운다. 키 복구 시에, 만약 노드에 저장된 $last$와 c와의 차이가 w 이상이면 $flag$의 모든 값을 지워야 하며, 그렇지 않을 경우에는 $flag[(\,last+1) \bmod w]$부터 $flag[\,c \bmod w]$까지 지운다.

이 그룹키 복구 알고리즘을 C와 유사한 가상코드로 프로시쥬어 1에 기술하였다.

5. 분 석

5.1 안전성 및 신뢰성

제안한 스킴에서, 그룹키는 KDC만이 알고 있는 $Mkey$를 키로 하는 의사 난수 생성 함수를 이용하여 계산된다. 계산된 값은 하위 노드키로 암호화되거나 멤버와 KDC와의 공유키로 암호화하여 전송되므로, 이에 사용되는 암호화 알고리즘이 안전하다면 그룹키에 대한 GKS는 보장된다. 그룹키는 KDC의 비밀키 $Mkey$를 키로 하고 그룹키 버전을 인수(seed)로 하여 의사 난수 생성 함수를 통해 생성된다. 난수 생성 함수의 성질에 [16] 따라 함수값을 예측하거나 키값을 예측하는 것은 계산상 불가능하다. 따라서, 일부 그룹키 집합을 알고 있는 멤버라 할지라도 그 이전이나 이후의 그룹키 및 $Mkey$를 알 수 없으므로 FS와 BS가 보장된다.

그룹키는 $MKey$와 그룹키 버전만 가지고 계산이 가능하므로, KDC는 멤버가 요구한 임의의 그룹키를 복구할 수 있다. 또한 KDC는 멤버가 분실한 메시지 이후에 변경된 키들 중에서, 분실한 키 갱신 메시지의 복구 없이는 알 수 없는 그룹키들와 KEK들을 계산하여 전송한다. 따라서 제안한 방법은 신뢰성과 키-복구성을 만족한다.

프로시쥬어 1 그룹키 복구 알고리즘

```
q ← the id of the member who loses a rekey message;
r ← the version number of the requested GK;
c ← the version number of the current GK;
if ( c - r >= w )   {  // the requested GK is not within the recovery window
     for (i← r ; i<= c ; i++)     AddGK(&msg, GK_i);
     node ← node corresponds to m_q;
     while (node != root) {
          AddKEK(&msg, node->id, node->key);
          ModifyFlag(node);
          node ← node->parent;
     }    // while
}         // then
else {          // the requested GK is within the recovery window
     l_q^r ← node corresponds to m_q;
     ModifyFlag( l_q^r );
     s ← c +1;
     while ( l_q^r != root ∧ l_q^r ->flag[ r % w ] == none) {
          t ← the smallest index ∋ r<t<s ∧ l_q^r ->flag[ t % w ] == leave;
          if ( t exist)  s ← t;
          l_q^r ← l_q^r ->parent;
          ModifyFlag( l_q^r );
     }
     t ← the smallest index ∋ r<t<s ∧ l_q^r ->flag[ t % w ] == leave;
     if ( t exist)  s ← t;
     for (i← r ; i< s ; i++)     AddGK(&msg, GK_i);
     if ( s > c ) {
          node ← l_q^r ;
          while (node != root) {
               AddKEK(&msg, node->id, node->key);
               ModifyFlag(node);
               node = node->parent;
          }    // while
     }         // if-then
}              // else
EncUnic( k_q , m_q , msg);
// the msg is encrypted with k_q and unicasted to m_q
return;
```

5.2 효율성

스킴의 효율성은 KDC의 저장 공간, 전송량, 키의 암·복호화에 대하여 분석한다. 먼저 w개의 최근 메시지를 버퍼링하고 키 복구 요구시 해당 메시지를 재전송하는 방법에서의 효율성을 분석해 보자. $event_r = join$이고 이 이벤트가 m_x에 의하여 발생했을 때, $ReKey_i$는 $path_x^r$의 각 키들이 이에 대응하는 $path_x^{r-1}$의 키들로 암호화되어 전송된다. 따라서 $ReKey_r$에 포함된 키의 수는 $path_x^r$의 크기, 즉 T_r의 높이에 해당하는 $\log_2 n$과 같다. $event_r = leave$일 때는, $path_x^r$의 각 키들이 두 자식 노드 키들로 암호화되어 전송되므로 $ReKey_r$에 포함된 키의 수는 $2 \cdot \log_2 n$이다. 따라서, 가입과 탈퇴의 빈도가 동일하다고 가정하고 키의 비트 길이를 K라고 했을 때 전송량은 평균 $1.5K \cdot \log_2 n$이다. 또한 KDC가 필요한 저장공간은 최악의 경우에 대한 w개의 키 갱신 메시지이므로 $2wK \cdot \log_2 n$ 비트이다. 균형 잡힌 키-트리를 가정할 때, $n/2$ 멤버는 각 키 갱신 메시지에서 1개의 키만 갱신하고, $n/4$ 멤버는 2개의 키를 갱신하며, 일반적으로 i개의 키를 갱신하는 멤버의 수는 $n/2^i$이다[5]. 따라서 하나의 키 갱신 메시지당 멤버의 평균 키 갱신의 수는 2개이다. 메시지를 분실한 멤버는 KDC로부터 $ReKey_r$를 재전송 받은 후, $p = c - r + 1$개의 메시지 $ReKey_r, \ldots, ReKey_c$에서 각각 2개의 키를 복호화하므로 $2 \cdot p$개의 복호화가 필요하다. 최악의 경우에는 각 메시지마다 $\log_2 n$개의 키를 복호화하여야 하므로 $p \cdot \log_2 n$개의 복호화가 수행되어야 한다.

제안한 방법에서는 트리의 각 노드마다 $2w$비트의 $flag$가 필요하고, 노드 수는 $n-1$이므로 KDC의 저장공간은 약 $2wn$과 같다. 메시지 전송량의 최악의 경우는 $level(l_q^r)$이 트리의 높이인 $\log_2 n$이고, 조건을 만족하는 s가 존재하지 않을 경우이므로 p개의 그룹키와 $\log_2 n$개의 KEK가 전송된다. 즉, 최악의 경우에 대한 전송량은 $K \cdot (p + \log_2 n)$ 비트이다. 조건을 만족하는 s가 존재할 경우에는 s개의 그룹키 $GK_r, \ldots, GK_{s-1}$만 전송하므로 평균 전송량은 $K \cdot s$비트이다. $event_i$가 $leave$이거나 $join$일 확률이 각각 1/2이고, $l_q^s \le l_q^i$ ($r < i$)일 확률이 2/3이므로 s의 기대치는 3이 된다. 이 값은 평균 전송량이 그룹의 크기나 분실된 메시지가 얼마나 오래전 것인가와 상관없이 상수라는 것을 의미한다. 멤버는 전송되는 키의 개수만큼 복호화를 수행해야 하므로, 최악의 경우에는 $p + \log_2 n$개, 평균 $s = 3$개를 복호화한다. 분석 및 비교 결과를 표 1에 요약하였다.

표 1 효율성 비교

항목		메시지 재전송 방법	제안 방법
KDC 저장 공간 (비트수)		$2wK \cdot \log_2 n$	$2wn$
메시지 전송량 (비트수)	평균	$1.5K \cdot \log_2 n$	$3K$
	최악	$2K \cdot \log_2 n$	$K \cdot (p + \log_2 n)$
계산량(복호화)	평균	$2p$	3
	최악	$p \cdot \log_2 n$	$p + \log_2 n$

종합해 보면, $p > \log_2 n$일 때의 최악의 전송량을 제외하면 메시지 전송량이나 멤버의 계산량은 현격히 감소한다. 그러나 스킴의 효율성은 최악의 경우가 아니라 평균 계산량에 의하여 비교되는 것이 타당할 것이다. 제안 방법에서 KDC의 저장공간은 재전송 방법보다 많은 것으로 분석된다. 그러나 그룹의 규모와 상관없이 멤버의 98%는 6개 이내의 키들만 필요로 하므로[5], 트리의 레벨 6까지만 $flag$를 유지한다면, 평균 효율성의 희생없이 KDC의 저장 공간을 상수개로 줄일 수 있다.

6. 결론 및 향후 연구 과제

그룹 통신의 보안이나 과금을 위해 사용되는 그룹키는 멤버쉽의 변동이 있을 때 키 갱신 메시지를 통하여 변경해 주어야 한다. 만약 이 메시지를 분실할 경우 그룹 데이타의 복호화가 불가능하고 과금도 어려워지므로, 신뢰성 있는 그룹키 갱신 메시지의 전달뿐 아니라 분실된 키의 복구는 매우 중요한 문제이다. 현재 신뢰성 제공 방안에 대한 연구는 많지 않을뿐더러, 그러한 연구들은 최근 키 갱신 메시지 수신의 약-실시간성을 주기 위한 방법에 대한 연구이거나 통신량을 줄이기 위하여 멤버가 전수 조사하여 키를 복구하는 방법이다. 그러나 약-실시간성을 제공하는 방법에서는, 멤버의 로그오프 동안 분실된 메시지는 실시간 복구가 불가능하며, 약-실시간성을 제공하더라도 키 갱신 메시지의 분실을 완전히 방지하지는 못한다. 멤버의 전수 조사에 의한 방법에서는 키 갱신 메시지와 함께 복구에 필요한 정보가 분실되면 이후의 키 갱신 메시지도 복호화하지 못한다. 따라서 분실된 키에 대한 복구 방안이 필요하다. 이를 위해 KDC가 전송된 메시지를 저장하고 이를 재전송한다면, KDC의 많은 저장 공간을 요구할뿐 아니라 저장되지 않은 메시지는 복구가 불가능하다. 또한 더 이상 유효하지 않은 KEK들도 전송하기 때문에, 멤버가 이들에 대한 불필요한 복호화 과정을 수행하도록 한다. 본 논문에서는 LKH를 기반으로, 키 갱신 메시지의 분실이나 멤버의 로그인 시에 발생할 수 있는 문제를 정의하고 분석

하였으며, 효율적으로 그룹키 및 KEK들을 복구하는 방법을 제시하였다. 제안한 방법에서는 간단한 그룹키 생성 방법을 제시함으로써 메시지를 저장하지 않고 임의의 그룹키를 복구할 수 있도록 하였다. 또한 키-트리에 저장된 갱신 정보를 이용하여, 메시지 분실 후 발생한 키 갱신으로 인하여 불필요해진 KEK들의 전송을 없애고, 복구가 불가능한 모든 그룹키들과 최소의 KEK들만 전송하도록 하였다. 이로써 멤버들이 유효하지 않은 KEK들의 복호화 없이 임의의 그룹키를 복구할 수 있다.

제안한 방법은 키 갱신 메시지 전송 시에 사용하는 방법과 무관하게, 멤버가 키를 분실했을 때 KDC에 키 복구를 요청하고 이에 따라 복구 메시지를 전송하는 방법이다. 만약 네트웍의 상태가 불안정할 경우에는, 키 분실율이 높아지고 이에 따라 멤버의 복구 요청이 증가하므로 KDC로의 통신량 폭주가 일어난다. 이를 방지하기 위해서는 [7], [9]나 [10] 등의 방법과 혼용하여 사용하는 것이 바람직할 것이다. 이들 방식만을 이용할 경우, 수신율이 100%에 가깝도록 충분히 많은 중복 전송을 해야 하므로 적절한 수신율을 보장할 수 있도록 중복 전송하고, 분실된 메시지에 대해서는 제안한 방법으로 복구하는 것이 효율적일 것이다. 추후 효율성 극대화를 위한 적절한 수신율에 대한 연구가 필요하다. 또한 요구되는 신뢰도와 p 값과 같은 데이타 및 응용의 특성과 KDC 및 멤버의 계산 능력에 따른 적절한 $w(\geq 0)$ 설정에 대한 실험과 분석이 필요하다.

참 고 문 헌

[1] C. K. Wong, M. Gouda and S. S. Lam, "Secure Group Communications using Key Graphs," Proc. of ACM SIGCOMM, Vol 28, Issue 4, pp68-79, 1988.

[2] http://www.ietf.org/html.charters/msec-charter.html

[3] http://www.securemulticast.org/msec-index.htm

[4] T. Hardjono, B. Cain and B. Doraswamy, "A Framework for Group Key Management for Multicast Security," IETF Internet Draft, 2001.

[5] A. Perrig, D. Song and J. D. Tygar, "ELK, a New Protocol for Efficient Large-Group Key Distribution," 2001 IEEE Symposium on Security and Privacy, pp247-262, 2001.

[6] 조태남, 이상호, "(2,4)-트리를 이용한 그룹키 관리", 정보보호학회 논문지, 제11권, 제4호, pp77-88, 2001.

[7] C. K. Wong and S. Lam, "Keystone: A Group Key Management Service," Proc. of International Conference on Telecommunications, 2000.

[8] S. Setia, S. Zhu and S. Jajodia, "A Comparative Performance Analysis of Reliable Group Rekey Transport Protocols for Secure Multicast," Proc. of the Performance 2002 Conference, 2002.

[9] S. Setia, S. Zhu and S. Jajodia, "A Scalable & Reliable Key Distribution Protocol for Group Rekeying," Technical Report, George Mason Univ., 2002.

[10] X. B. Zhang, S. S. Lam, D. Lee and Y. R. Yang, "Protocol Design for Scalable and Reliable Group Rekeying," Proc. of SPIE Conference on Scalability and Traffic Control in IP Networks, 2001.

[11] D. Wallner, E. Harder and R. Agee, "Key Management for Multicast: Issues and Architectures," RFC 2627, 1999.

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

[13] O. Rodeh, K. P. Birman and D. Dolev, "Optimized Group Rekey for Group Communication Systems," Network and Distributed Systems Security 2000, pp37-48, 2000.

[14] D. Balenson, D. McGrew and A. Sherman, "Key Management for Large Dynamic Groups: One-way Function Trees and Amortized Initialization," IETF Internet Draft, 1999.

[15] S. Rafaeli, L. Mathy and D. Hutchison, "EHBT: An Efficient Protocol for Group Key Management," 3rd International Workshop on Networked Group Communications, pp159-171, 2001.

[16] O. Goldreich, S. Goldwasser and S. Micali, "How to Construct Random Functions," Journal of the ACM, Vol.83, Issue 4, pp792-807, 1986.

조 태 남
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조

김 상 희
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조

이 상 호
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조

채 기 준
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조

박 원 주
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조

나 재 훈
정보과학회논문지 : 정보통신
제 30 권 제 4 호 참조