情報保護學會論文誌
第12券 第6號, 2002. 12

일방향 함수와 XOR을 이용한 효율적인 그룹키 관리 프로토콜: ELKH

권 정 옥*, 황 정 연*, 김 현 정*, 이 동 훈*, 임 종 인*

ELKH, Efficient Group Key Management Protocol Using One-Way Function and XOR

Joung-ok Kwon*, Jung-yeon Hwang*, Hyun-jeong Kim*,
Dong-hoon Lee*, Jong-in Lim*

요 약

다수의 다양한 구성원들로 구성된 멀티캐스트 그룹은 매우 번동적이며, 이로 인하여 구성원들이 매 세션마다 빈번히 추가되거나 삭제된다. 따라서 새로운 세션을 시작하기 위해서는 세션키가 효율적으로 갱신되고 분배되어야 한다. 본 논문에서는 일방향 함수(one-way function)와 XOR 연산자를 이용해서 효율적이면서도 안전하게 그룹키를 갱신하고 분배할 수 있는 그룹키 관리 프로토콜 ELKH(Efficient Logical Key Hierarchy)를 소개한다. LKH에 기반한 문헌의 키 관리 프로토콜들에서는 안전하지 않은 멀티캐스트 채널을 통해 정당한 구성원들에게 키 갱신 메시지를 전달해야 하는 경우에 특정 암호 알고리즘을 필요로 한다. 하지만 본 논문에서 제시하는 프로토콜은 특정 암호 알고리즘 대신에 일방향 함수를 이용하여 메시지를 은닉하는 새로운 접근방법이다. 본 논문의 주된 결과는 ELKH가 EHBT[12]와 비교해, 키 갱신 메시지의 크기가 증가되지 않으면서 키 갱신 속도가 더 빠르다는 것이다. 그리고 주어진 일방향 함수의 안전성에 기반해서 ELKH 프로토콜이 *forward secrecy*와 *backward secrecy*를 만족한다는 것을 증명한다.

ABSTRACT

Since the multicast group which is composed of various members is dynamic, members of the group frequently join or leave. So, for a new session, group keys are efficiently updated and distributed. In this paper, we describe very simple and new efficient logical key hierarchy(ELKH) protocol which is based on an one-way function. In the previous schemes, when the group controller distributes new created keys or updated keys to the members the information is usally encryted and then transmited over a multicast channel. But ELKH secretes the multicast message by using the one-way function and XOR operator instead of encrypting it. Hence our main construction improves the computational efficiency required from the group controller and group memebers while doesn't increase size of re-keying message when compared to EHBT[12]. Assuming the security of an underlying one-way function, we prove that our scheme satisfies *forward secrecy* and *backward secrecy*.

Keyword : *Multicast, Logical Key Hierarchy, One-way Function*

* 고려대학교 정보보호기술연구센터(CIST)
({pitapat, videmot, khj}@cist.korea.ac.kr, {donghlee, jilim}@korea.ac.kr)

Ⅰ. 서 론

최근 멀티캐스트 기법은 인터넷을 기반으로 한 유료 TV, 유료 영상 서비스, 비밀 원격회의 등 공개된 네트워크 상에서 디지털 정보들을 전송하는데 적용되고 있다. 멀티캐스트 통신에서 그룹 메시지는 그룹의 모든 구성원들에게 전달된다. 일반적으로 멀티캐스트 그룹에서 정보에 대한 접근권한을 제어하기 위해, 그룹키(세션키)로 데이터를 암호화 한다. 그룹키는 단지 그룹내의 구성원들에게만 알려져 있고, 이 키를 알고 있는 구성원들만이 메시지를 복호할 수 있다. 멀티캐스트 통신은 메시지를 개별적으로 n명의 그룹 구성원들에게 각각 n번 보내는 대신에 모든 구성원에게 단지 메시지를 한번만 전달하면 되기 때문에 효율성면에서 매우 뛰어나다. 그러나 현실상의 문제는 구성원들의 추가 또는 삭제로 인한 그룹의 변동성 때문에 프라이버시에 대한 안전성과 키 갱신 작업의 효율성에 관한 것이다.

여러 암호기술들이 그룹 통신을 보호하기 위해서 사용되는데, 그 메카니즘 중에 하나가 암호화이다. 암호 알고리즘은 입력값으로 그룹 메시지를 받고, 임의로 생성된 키를 이용하여 변환(transformation)을 수행한다. 결과적으로 이 처리 과정은 암호화된 메시지를 생성한다. 그러나 이 메시지를 복호할 때, 키를 알고 메시지를 복호하는 방법보다 쉽게 암호문으로부터 원래의 메시지를 복호할 수 있는 방법은 없다[4]. 따라서 이러한 암호화 기술을 이용하는 것이 안전한 멀티캐스트 세션의 수행을 가능하게 한다. 그룹 메시지는 선택된 키(그룹키)를 이용해 암호화됨으로써 보호된다. 단지 그룹키를 알고 있는 사람만이 본래의 메시지를 복호할 수 있다.

그러나 정당한 구성원들에게 그룹키를 분배하는 것은 복잡한 문제이다. 비록 새로운 구성원이 추가되는 경우에 그룹의 키들을 변경하는 것이 쉽다 할지라도 구성원이 삭제된 후에 키들을 갱신하는 것은 더 복잡하다. 왜냐하면 갱신된 키들은 삭제된 구성원들이 알고 있는 키들에 의존하지 않는 새로운 키 값들이여야 하기 때문이다.

그룹키 분배 문제와 키 갱신 문제를 해결하기 위해서 제안된 스킴들은 보통 아래의 파라메터들에 기반해서 평가된다.

- **구성원 저장공간.** 각 구성원이 저장해야 하는 키의 수이다. 보통 저장해야 하는 키의 수가 작기 때문에 구성원의 키 저장량은 문제가 되지 않는다.
- **그룹 관리자 저장공간.** 그룹 관리자는 그룹내의 모 든 구성원들의 키들을 가지고 있다. 현재 스킴들에서 전체 구성원의 키들은 약 $O(n)$개이다. 그룹 관리자의 저장량을 줄이기 위해 제시된 스킴이 있다[7]. 그러나 본 논문에서는 그룹 관리자가 수백만의 가입자들에게 데이터를 보내기 때문에 충분한 저장공간을 가지고 있다고 가정한다.
- **구성원과 그룹 관리자의 계산량.** 키 갱신에 소요되는 산량으로, 보통 암·복호화와 다른 타입의 오퍼레이션들을 구분한다. 멀티캐스트 그룹은 매우 변동 적이고 수 많은 가입자들로 구성되기 때문에 키 갱 신 속도가 가능한 빨라야 한다. 본 논문에서 제안하는 새로운 스킴의 주된 목적은 문헌의 키 갱신 스킴 들보다 키 갱신 메시지의 크기가 증가하지 않으면서, 구성원과 그룹 관리자의 키 갱신 계산량을 최대한으로 낮추는 것이다.
- **키 갱신 메시지.** 키 갱신 정보는 그룹내의 모든 구성원들에게 전달되기 때문에 가능한 작아야 한다.

1.1 그룹키 갱신 스킴의 기존 연구들

그룹키 관리를 위해서 키를 갱신하는 경우에 논리적 키 계층(LKH : Logical Key Hierarchy)을 이용한 방법이 효율적이기 때문에 이것에 관한 여러 연구들이 있었다.

LKH에 기반한 그룹키 갱신 스킴은 Wallner 등[8]에 의해 처음으로 제시되었다. [8] 논문에서 제시된 접근 방법에서는 구성원의 변동이 있을 때 마다 변동에 영향을 받는 내부 키들이 새롭게 갱신되고, 새로운 키들은 영향을 받는 노드의 자식 키들로 암호화된 후 멀티캐스트 채널을 통해 분배된다. 한 명의 구성원이 삭제되는 경우에 키 갱신 메시지에는 $2\log_2 n$개의 키가 포함된다. 결과적으로 변경되는 키들이 자식 키들로 암호화되서 전달되기 때문에 키 갱신 메시지의 크기는 매우 크다.

[8]의 개선된 접근방법은 McGrew와 Sherman이 제안한 일방향 함수트리 OFT[6]이다. 여기에서 하나의 노드키 값은 자식 노드들의 키들을 일방향 함수로 블라인드 (blind)한 후 XOR 연산자로 혼합된 결과이다. XOR된 노드키들의 결과값은 KEK (KEK : Key Encryption Key)로 유지된다. 이 스킴이 이전의 스킴보다 개선된 이유는 하나의 노드키가 변경될 때, 자식 키들로 암호화되는 대신에 단지

변경된 키의 블라인드 값이 동기(sibling) 노드의 키로 암호화되서 전달되기 때문이다. 따라서 $\log_2 n$ 개의 키들이 키 갱신 메시지로 전달된다.

Canetti 등[7]은 [6]과 같은 키 갱신 메시지 크기를 갖는 조금 다른 접근 방법을 제안 했다. 이 스킴에서는 새로운 KEK들을 생성하기 위해 일방향 함수를 사용하는 대신에 pseudo-random-generator (PRG[4])를 사용하는데, 이것은 단지 구성원 삭제시에만 적용된다.

Caronni 등[9]은 [8]에서 제시된 프로토콜과 매우 흡사한 VersaKey 방식의 프로토콜들을 제안했는데, [9]에서 제시한 프로토콜들에서는 추가 오퍼레이션 동안에 이전에 제시된 스킴들보다 키 갱신 메시지면에서 향상되었다. 본 논문에서는 [9]의 VersaKey 방식의 프로토콜들 중 트리에 기반한 프로토콜을 LKH^+라 표기한다. [9]에서는 구성원 추가시에 새로운 키 값들을 대응되는 자식 노드의 키들을 이용해서 암호화 하는 대신에 일방향 함수에 새로운 키들을 통과시키는 방법을 제안했다. 따라서 새롭게 변경되는 키들의 인덱스들만을 멀티캐스트로 전달하면, 이전의 키들을 알고 있는 구성원들은 받은 인덱스에 대응되는 키들을 일방향 함수에 통과시킴으로써 키들을 갱신하게 된다. 여기에서 멀티캐스트되는 인덱스의 크기는 키 크기 보다 작기 때문에 키 갱신 메시지의 크기가 감소하게 된다.

Chang 등[10]은 여러명의 구성원들을 동시에 삭제하는 문제에 초점을 맞추고, 이진트리에 boolean function minimization technique을 이용해서 그룹 관리자의 저장량을 크게 줄였다. [10] 스킴에서 구성원의 저장량은 $\log_2 n$이고, 그룹 관리자의 저장량이 $\log_2 n$으로 감소한다. 그러나 이 스킴의 큰 단점은 아이디가 서로 보수인 2명이 공모할 경우에는 시스템내의 모든 키들을 알아낼 수 있다는 것이다.

Rafaeli 등은 앞에서 기술한 스킴들과 비교해 멀티캐스트되는 키 갱신 메시지 크기와 그룹 관리자, 구성원들의 키 갱신 계산량면에서 가장 향상된 스킴 EHBT[12]를 제안했다. 이 스킴에서는 구성원이 추가 될 때, LKH^+와 비슷하게 새로운 키 값들이 일방향 해쉬함수에 통과되므로, 새롭게 변경되는 키들의 인덱스들만 멀티캐스트된다. 인덱스의 길이는 키의 길이보다 작기 때문에, 결과적으로 멀티캐스트 메시지의 크기가 감소하게 된다. LKH^+와 차이점은 다음과 같다. 새로운 구성원이 추가될 때, LKH^+에서는 그룹 관리자가 새로운 구성원의 동기 구성원에게 새로운 고유키를 유니캐스트 채널로 전달하지만, EHBT에서는 그룹 관리자가 따로 고유키를 동기 구성원에게 전달할 필요가 없다. 왜냐하면 동기 구성원 스스로 고유키를 갱신하기 때문이다.

그리고 구성원 삭제시에는 삭제되는 구성원의 동기 구성원의 고유키를 이용해 루트키까지의 키들이 일방향 해쉬함수를 통해 갱신되기 때문에, 삭제되는 구성원의 동기 구성원은 스스로 키를 갱신할 수 있다. 그리고 그 이외의 구성원들에게는 변경된 키가 암호화되어 전달된다. 따라서 동기 구성원에게는 갱신된 키를 암호해서 전달할 필요가 없게 되고, 결과적으로 멀티캐스트 메시지의 크기도 이전의 스킴들보다 줄어들게 된다. 또한 동기 구성원에게는 암호화해서 전달되는 키가 필요없기 때문에 동기 구성의 복호연산이 필요없게 되고, 결국 키 갱신 계산량도 줄어들게 된다.

1.2 본 논문의 연구 결과

본 논문에서 LKH에 기반한 효율적인 그룹키 관리 프로토콜을 제안한다. 이 프로토콜을 ELKH로 부르기로 한다. ELKH 프로토콜은 단일 구성원 추가, 삭제시와 여러명의 구성원이 추가되고 삭제되는 경우에 빠른 키 갱신을 위해서 잘 알려진 일방향 함수와 XOR 연산자를 이용한다.

그리고 기존의 LKH 스킴들에서는 갱신되거나 새로 생성된 키를 정당한 그룹의 구성원들에게 분배할 때, 그룹 관리자는 정당한 구성원들이 알고 있는 노드키로 암호화된 메시지를 멀티캐스트 채널로 전달한다. 하지만 ELKH는 어떤 특정한 암호 알고리즘을 거치지 않고, 단지 일방향 함수와 XOR 만을 이용해서 갱신 메시지를 은닉하는 매우 간단하면서 효율적인 새로운 접근 방법이다. 즉, 암·복호화가 필요 없는 스킴이다. 따라서 ELKH 프로토콜은 이전의 스킴들과 비교해서 그룹 관리자와 구성원들이 효율적으로 키를 갱신할 수 있는 방법을 제공한다.

ELKH는 일방향 함수와 XOR 연산자를 이용해 EHBT[12]와 비슷한 방법으로 키를 갱신한다. 본 논문의 주된 목적은 EHBT의 키 갱신 메시지의 크기보다 키 갱신 메시지의 크기가 증가하지 않으면서 그룹 관리자와 구성원들의 키 갱신 계산량을 최대한으로 줄이는 것이다.

단일 구성원에 대한 사건을 고려할 때, ELKH에서 키 갱신 메시지의 크기와 그룹 관리자와 구성원

들에게 요구되는 키 계산량은 다음과 같다(여기에서 트리의 균형이 사건들로 인해 깨지지 않는 경우에 대해서 고려한다. 그리고 I를 키 인덱스의 길이, K를 하나의 키 길이로 본다).

- 구성원 추가시, 그룹 구성원이 키를 갱신하는데 필요한 계산량은 최대 한번의 XOR 연산과 $\log_2 n$번 해쉬 연산하는 것과 같고, 멀티캐스트 메시지의 크기는 $I \cdot \log_2 n$이다. 그리고 그룹 관리자가 추가되는 구성원의 동기 구성원에게 개별적으로 유니캐스트 해야되는 메시지는 없다.
- 구성원 삭제시, 그룹 구성원이 키를 갱신 할 때, 최대 $\log_2 n$번의 XOR 연산과 $\log_2 n-1$번 해쉬 연산이 필요하고, 그룹 관리자가 멀티캐스트 해야 하는 메시지의 크기는 $(K-1) \cdot \log_2 n$이다.

ELKH에서는 그룹에 여려명의 새로운 구성원이 추가되거나 삭제되는 수와 공모의 수에 제한이 없다. 실제로 ELKH이 2장에서 정의한 안전성 요구조건들을 만족한다는 것을 뒤에서 증명할 것이다.

1.3 본 논문의 구성

2장에서 키 갱신 스킴에 요구되는 그룹 오퍼레이션들과 안전성들을 정의하고, 3장에서 ELKH 프로토콜에서 이용되는 연산식들과 키 갱신 메시지의 형태를 설명한다. 또한 4장에서 ELKH의 키 관리 프로토콜들에 대해 기술한다. 그리고 5장과 6장에서 각각 ELKH의 안전성과 효율성을 구체적으로 분석한다. 마지막으로 7장에서 결론을 내린다.

II. 키 갱신 스킴

■ 수식

본 논문에서는 수식을 아래와 같이 표기한다.

M : 시스템내의 모든 구성원들의 집합
K : 시스템내의 모든 키들의 집합
M_i : 세션 i에서 구성원들의 집합
K_i : 세션 i에서 키 집합
m_{i_j} : 세션 i에서 j번째 구성원
m_i : i번째 구성원
k_{s_i} : 세션 i에서 세션키(그룹키)
k_i : 키 인덱스 i에 대응되는 노드키

■ 모델

멀티캐스트 키 분배 스킴에서 중앙 그룹 관리자(또는 키 서버)가 개개의 수신자를 허가하고 인증할 수 있다고 가정한다. 이 모델에서는 멀티캐스트되는 컨텐츠를 복호하고자 하는 수신자가 유니캐스트로 키 서버에 접촉해서 복호화 키를 요구한다. 그 키 서버는 그 수신자를 표준 인증 프로토콜을 통해 인증하고, 안전한(기밀성, 무결성, 인증을 제공하는) 채널을 확립한다. 그룹 관리자는 컨텐츠를 복호할 수 있는 그룹키와 키 관리 목적들을 위한 구성원 고유의 키들로 구성된 키 정보를 각 구성원에게 보낸다. 이 때, 멀티캐스트되는 정보는 기밀성(confidentiality)과 접근권한(access control)을 유지하기 위해서 그룹키로 암호화된다.

본 논문에서 시스템내의 모든 구성원들의 집합 M과 하나의 그룹 관리자가 존재하는 모델에서의 시나리오를 고려한다. 한 명의 구성원 m은 유일한 키의 집합 $K_i(m) \subseteq K$을 가지고 있고, K는 시스템내의 모든 키들의 집합이다. 그룹의 구성원이 *번동적*이라는 의미는 세션마다 그룹의 구성원이 빈번히 추가되고 탈퇴한다는 것을 의미한다. 즉, 새로운 세션을 시작하기 위해서 키 갱신 스킴을 사용한다. 한 세션 i에서, 하나의 그룹은 하나의 세션키(그룹키) $k_{s_i} \in K$를 공유하는 구성원들의 집합 $M_i = \{m_1, \ldots, m_n\} \subseteq M$로 구성된다. 그리고 그룹 외 구성원 $m \notin M_i$은 세션키 k_{s_i}를 알 수 없다.

■ 그룹 오퍼레이션(operation)

키 갱신 스킴은 아래의 두 가지 그룹 오퍼레이션들로 구성된다.

- *구성원 추가*. 세션 i에서 추가되는 구성원 집합 $J_i \subseteq (M \mathrm{M}_i)$는 M_i에 추가되고, 새로운 세션은 새로운 세션키 $k_{s_{i+1}}$를 공유하는 $M_{i+1} = M_i \cup J_i$로 구성된다.
- 구성원 *삭제*. 세션 i에서 삭제되는 구성원 집합 $R_i \subseteq M_i$는 M_i로부터 삭제되고, 새로운 세션은 새로운 세션키 $k_{s_{i+1}}$를 공유하는 $M_{i+1} = M_i \backslash R_i$로 구성된다.

■ 시스템 오퍼레이션

초기 세션 $i=0$ 동안, 그룹 관리자[1]는 키 트리내의

1. 그룹 관리자가 그 그룹의 구성원일 수 있지만, 이 논문에서는 그룹 관리자를 구성원들의 집합에서 제외하기로 가정한다.

키들 K_0[2]를 생성하고, 키들의 부분집합 $K_0(m) \subseteq K_0$을 구성원 $m \in M_0$[2]에게 안전한 유니캐스트 채널(unicast channel)을 통해 전달한다. 이어지는 모든 세션들에서 그룹 관리자는 안전하지 않은 멀티캐스트 채널상으로 키 갱신 메시지 M_{rkey}를 전달함으로써 키 갱신을 관리한다. 구성원 $m \in M_{i+1}$은 새로운 세션키 $k_{s_{i+1}}$를 계산하기 위해서 자신의 키들의 집합 $K_i(m)$과 키 갱신 메시지 M_{rkey}를 이용한다.

■ 안전성 성질들

이 논문에서 공격자들이 소극적(passive)이고 계산적으로 제한되어 있다고 가정한다. 멀티캐스트 키 갱신 스킴은 아래의 안전성 타입들 중 하나를 제공해야 한다.

[정의 1]

만약 어떤 세션 j에 대해($J_a \subseteq (M \setminus M_a), a \geq j$), 세션 j 이후에 추가되는 모든 구성원 집합 $All(J_j) = \cup J_a$ 내의 구성원들이 공모했을 경우에 $All(J_j) \cap M_b = \emptyset$ $(0 \leq b \leq j)$일 때, 이전 세션 b에서 생성된 키 정보 K_b를 알아내는 것이 계산적으로 무시할 수 있는 (computationally negligible) 확률이라면, 그 키 갱신 프로토콜은 backward secrecy를 제공한다.

[정의 2]

만약 어떤 세션 i에 대해($R_a \subseteq M_a$, $0 \leq a \leq i$), 세션 $i+1$ 이전에 삭제된 모든 구성원 집합 $All(R_i) = \cup R_a$ 내의 구성원들이 공모했을 경우에 $All(R_i) \cap M_b = \emptyset$ $(b \geq i+1)$일 때, 이후 세션 b에서 생성된 키 정보 K_b를 알아내는 것이 계산적으로 무시할 수 있는 (computationally negligible) 확률이라면, 그 키 갱신 프로토콜은 forward secrecy를 제공한다.

위의 정의들을 살펴보면 *forward secrecy*의 경우에 공격자들은 그룹에서 삭제된 구성원들이다. 여기에서 요구되는 조건은 모든 공격자들이 공모해서 이전의 세션들로부터 얻은 정보를 통해 이후에 이어지는 세션들의 키를 얻을 수 없어야 한다는 것이다. 그리고 *backward secrecy*의 경우, 공격자들은 추가되는 구성원들이고 여기에서 요구되는 조건은 모든 공격자들이 공모해서 이어지는 세션들에서 얻은 정보를 통해 이전의 어떤 키도 얻을 수 없어야 한다는 것이다.

Ⅲ. ELKH의 구조

ELKH는 LKH의 키 트리 구조에 기반하고 있다. 이 논문에서 구성원들의 집합을 $M=\{m_1, \ldots, m_n\}$이라 하고, n개의 구성원-노드(최하위 노드)를 가진 트리를 T라 한다. 트리의 노드(노드 키)들은 내부 노드들과 구성원-노드(구성원 고유키)들로 구분된다. 각 노드에 하나의 노드 키 k_i가 주어지고, i는 키를 식별하는 인덱스로써 유일한 수이다. 트리에서 최하위 노드들은 각각 한 명의 구성원 m에 대응된다. 본 논문에서 내부 키들을 KBK(Key Blind Key)라 정의하고, 구성원에 대응되는 최하위 노드 키를 구성원의 고유키로 정의한다.

그룹 관리자는 키 트리 T를 유지하고, 각 사용자에게 대응되는 키들(구성원-노드부터 루트 노드까지 경로내에 대응되는 키)을 구성원들에게 안전한 채널로 전달한다. 구성원 m의 구성원-노드의 상위 노드부터 루트 노드까지의 트리 경로내의 키의 집합을 $AK(m)$이라 할 때, 각 구성원은 자신의 고유키와 $AK(m)$를 유지하게 된다.

ELKH는 그룹 관리자가 갱신된 키들을 멀티캐스트할 때, 어떤 특정한 암호 알고리즘을 사용하지 않고, 효율적인 일방향 함수와 XOR만을 이용해 변경된 키 값을 은닉해서 분배하는 그룹키 관리 프로토콜이다. 따라서 이 키 갱신 메시지를 받은 구성원들은 복호화 과정 대신에 일방향 함수와 XOR만을 적용해서 변경된 키 값을 구할 수 있다. 그리고 구성원 추가 오퍼레이션 동안에는 갱신된 키들을 구성원들에게 전달할 필요가 없다.

■ 함수

*backward*와 *forward secrecy*를 보장하기 위해, 추가되는 구성원들 또는 삭제되는 구성원들에 대응되는 키들은 그룹의 구성원이 변동될 때 마다 변경되야 한다. 구성원 삭제시와 추가시에 다른 키들로부터 새로운 키들을 생성하기 위해서 다음 연산식들이 사용된다.

$$U(k_i) = h(k_i^{left/right} \oplus GK)$$

함수 h는 일방향 함수이고, $\oplus$는 일반적인 XOR 연

2. 일반성을 유지하기 위해서 $K_0 = K$, $M_0 = M$으로 가정한다.

산자이다. 함수 는 h본래의 값 $k_i^{left|right}$, GK를 $z=h(k_i^{left|right}\oplus GK)$ 값으로 숨김으로써, 단지 $k_i^{left|right}$ 또는 GK만 알고 있는 사람이 다른 값을 알 수 없게 만드는 역할을 한다. ⊕의 역할은 $k_i^{left|right}$와 GK의 값을 혼합함으로써 새로운 값을 생성하는 것이다.

구성원 삭제시에 변경되야 하는 키 k_i는 새로운 키 $k_i'=h(k_i^{left|right}\oplus GK)$로 변경된다. 여기서 $k_i^{left|right}$는 키 트리에서 k_i의 왼쪽 또는 오른쪽 자식 노드에 대응되는 키로써 가장 최근의 키(갱신된 키)이고, GK는 사건이 발생하기 바로 전 세션에서 구성원들이 가지고 있던 그룹키이다. 그리고 구성원 추가시, 새로 생성된 노드에 대응되는 키 k_i는 $U(k_i)$를 통해 생성된다. 여기서 GK는 추가사건이 이어서 발생하는 경우에 새로 추가되는 구성원이 받은 정보로부터 이전의 키 값을 알아내는 것을 방지하는 역할을 하고, 함수 h는 삭제된 구성원들과 새로 추가된 구성원들의 공모로부터 $k_i^{left|right}$를 숨기는 역할을 한다.

[설명 1]

EHBT에서는 키 갱신시에 $k_i'=h(k_i^{left|right}\oplus i)$를 사용하는데, ELKH에서는 $k_i'=h(k_i^{left|right}\oplus GK)$를 사용한다. 이 변화가 효율성이나 안전성에 영향을 미치지 않지만, 본 논문에서 인덱스 대신에 그룹키를 사용하는 이유는 전체스킴에 통일성을 주기 위한 것이다(예를 들어 아래의 블라인드 함수에서도 그룹키를 이용한다).

그리고 이전의 키를 이용해 키를 새롭게 생성하기 위해 다음 식이 사용된다.

$$R(k_i)=h(k_i)$$

키 k_i는 $k_i'=h(k_i)$로 새롭게 갱신되고, 기호로 $R(k_i)$를 사용한다.

[설명 2]

EHBT에서 구성원 추가시 함수 $k_i'=h(k_i\oplus i)$를 쓰는데, 만약 인덱스를 첨가하지 않는다면 backward secrecy가 깨지기 때문이다. 예를 들어 한명의 구성원이 삭제되고, 그 삭제된 구성원의 자리에 새로운 구성원이 추가되는 경우에 새로 추가된 구성원에게 추가되기 이전의 키 정보를 주게된다. 하지만 ELKH에서는 위와 같은 경우에 $k_i'=h(k_i)$를 사용해도 backward secrecy가 깨지지 않는다.

또한 갱신된 키들은 아래의 식으로 은닉되고, 그룹 관리자는 정당한 구성원들만이 키를 얻을 수 있도록 하기 위해서 은닉된 값을 키 갱신 메시지에 포함시켜 멀티캐스트 채널로 전달한다. 이 논문에서는 이것을 *블라인드*라는 용어로 사용한다.

$$BK_k(x)=x\oplus h(k\oplus GK)$$

$BK_k(x)$는 k를 이용해 x를 블라인드한 값이다. 여기에서 GK는 사건이 발생할 때 마다 전달되는 멀티캐스트 메시지로부터 키 값을 알아내는 것을 막기 위해서 사용되고, h는 일방향 함수로써 키 값들을 숨기는 역할을 한다. 이식은 간단히 $\langle x\rangle_k$로 표기한다.

[설명 3]

1장에서 살펴본 스킴들에서는 그룹 관리자가 정당한 그룹의 구성원들만이 키를 갱신할 수 있도록하기 위해서 멀티캐스트 메시지를 전달할 때, 키 갱신 메시지를 암호화해서 전달한다. 하지만 ELKH에서는 위와 같은 블라인드 함수를 이용해 메시지를 블라인딩해서 전달한다. 블라인드 함수를 사용하는 이유는 키 갱신 속도를 향상시키기 위해서이다.

만약 블라인드 함수로 $BK_k(x)=x\oplus h(k\oplus GK)$ 대신에 $BK_k(x)=x\oplus h(k)$를 사용한다면 다음과 같은 문제가 발생한다. 키 변경시에 $BK_k(x)=x\oplus h(k)$는 멀티캐스트 메시지로써 누구나 볼 수 있는 메시지이다. 예를 들어 그림 4에서 구성원 m_1이 삭제될 경우에 키 갱신 메시지는 $k_{14}'\oplus h(k_{34}), k_{18}'\oplus h(k_{58})$이다. 만약 구성원 m_2가 연이어서 삭제될 경우에 갱신된 그룹키가 k_{18}''이라고 할 때, 키 갱신 메시지는 $k_{18}''\oplus h(k_{58})$이다. 삭제된 구성원 m_2는 k_{18}'', $h(k_{58})$는 알 수 없지만, 만약 이전의 키 갱신 메시지 $k_{18}'\oplus h(k_{58})$를 저장해 두었다면, m_2는 삭제되기 이전의 그룹키 k_{18}'을 알고있음으로 이 메시지로부터 갱신된 그룹키 k_{18}''를 알아낼 수 있다($k_{18}'\oplus h(k_{58})\oplus k_{18}''\oplus h(k_{58})\rightarrow k_{18}''$). 이러한 공격을 방지하기 위해서 블라인드 함수에 반드시 GK를 추가하여야 한다. 매 사건이 발생할 때마다 이전의 GK를 첨가함으로써 키 갱신 메시지로부터 위의 경우와 같이 값들을 상쇄시킬 수 없게 된다.

블라인드 값으로부터 메시지를 얻기 위해 아래의 식을 이용한다.

$$UB_k(BK_k(x))=BK_k(x)\oplus h(k\oplus GK)$$

위의 $UB_k(BK_k(x))$를 통해 값 x를 얻는다.

h와 ⊕의 성질들은 5.1절에서 자세히 다룬다.

■ **안전성**

이 스킴의 안전성은 키 트리의 현재 상태에 대해 각 구성원들의 *knowledge*가 아래의 정의에 의해 제한된다는 사실에 기반한다.

[FACT 1]

LKH은 각 노드가 하나의 키에 대응되고 각 사용자-노드는 하나의 구성원에 대응되는 트리이다. 각 구성원은 자신의 구성원-노드부터 루트까지 경로내의 키를 알고 있고, 그 이외의 키들은 알 수 없다.

이러한 사실은 그룹으로부터 구성원 추가, 삭제시 모든 사건들에 대해 유지된다.

그룹키의 안전성은 우선 mixing함수 ⊕로부터 기인되고, 다음으로 내부 노드의 키들을 숨기기 위한 일방향 함수 h로부터 중요한 기능을 얻는다: 각 내부 노드 키는 모든 구성원들의 서브그룹에 대한 서브그룹 키로 통신하는데 사용되어 질 수 있다. 이 기능은 추가, 삭제시 효율성에 중요한 영향을 끼친다.

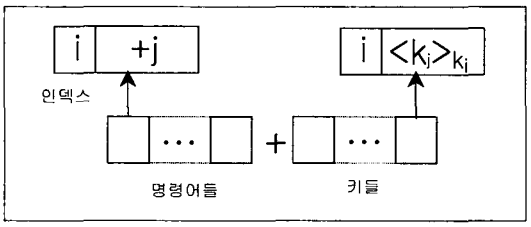
3.1 키 갱신 메시지 형식

ELKH의 키 갱신 메시지 형식은 EHBT[12]의 메시지 형태와 유사하다. 차이점은 키 갱신 메시지에 암호화된 키 정보 대신에 블라인드된 키 정보가 위치한다는 것이다.

구성원은 키 갱신 메시지에서 두 가지 타입의 정보를 얻을 수 있는데, 하나는 키를 갱신해야 하는지의 여부를 알리는 명령어이고 다른 타입은 키의 새로운 값이다. 이 두 타입의 정보 앞에는 키 인덱스(키 식별자)가 포함된다.

키 갱신 메시지를 받은 구성원은 받은 키 갱신 메시지에 만약 자신이 가지고 있는 키들의 집합(키-리스트)에 대응되는 명령어가 포함되어 있다면, 키 인덱스에 대응되는 키의 해당(메시지에 명시된) 명령을 수행한다. 그리고 키 갱신 메시지에 키 값이 포함되어 있는 경우에 수신자는 먼저 메시지내에 자신이 가지고 있는 키 인덱스와 대응되는 인덱스가 존재하는지 확인한다. 만약 대응되는 인덱스가 존재한다면, 그 키 인덱스에 대응되는 키로 자신이 가지고 있는 키를 대체한다. 즉 블라인드된 키 값을 복호한다.

정당한 구성원들만이 갱신된 키들을 얻을 수 있도록 하기 위해 키들은 앞에 붙여진 키 인덱스의 키들로 블라인드되어 있다([그림 1]). 그러나 명령어들은 블라인드되지 않는다. 왜냐하면 명령어들은 공개적으로 알려져도 안전성에 영향을 주지 않기 때문이다. 구성원들은 이 명령어들과 키 값들에 근거해서 어떤 키가 생성되어야 하고, 갱신되어야 하고, 대체되어야 하는지 알 수 있다. 본 논문에서 명령어(아래의 용어 참조)들을 임의로 특정 기호를 사용해서 표현한다.



[그림 1] 키 갱신 메시지 형식

예를 들어 [그림 2]에서 새로운 구성원 m_2가 추가될 경우, 수신자들은 그룹 관리자로부터 키 갱신 메시지 $M_{rkey}=\{[1:+12, 14:*14, 18:*18]\}$를 받게 된다. 그러면 구성원 m_1은 받은 명령어 (+)에 따라 노드 12를 새로 만들고 대응되는 키를 구하고, 키 갱신 메시 지내에 명시된 키들 $\{k_{14}, k_{18}\}$을 갱신한다. 그리고 나머지 구성원들은 키 갱신 메시지내에 자신이 가지고 있는 키-리스트에 대응되는 인덱스의 명령어 (*)를 실행한다. 명령어에 따른 키 관리 프로토콜들은 4장에서 자세히 다룬다.

■ **용어**

본 논문에서는 다음의 용어들을 사용한다.

j : 명령어 : 키 k_j를 가지고 있는 구성원들에 대응되는 명령어

$+i$또는 $-i$또는 $*i$: 키 k_i에 대응되는 명령어(순서 대로 추가, 삭제, 갱신)

[명령어들, 키들] : 명령어들과 키들을 포함한 메시지

Ⅳ. ELKH 키 관리 프로토콜들

본 장에서는 한 명의 구성원이 추가되고, 삭제되는 경우와 여러명의 구성원이 동시에 추가, 삭제되는

사건에 대한 키 관리 프로토콜들을 소개한다. 이 프로토콜들은 아래의 중요한 특성들을 지니는 구조를 공통적으로 가지고 있다.

- 각 구성원의 구성원-노드에 대응되는 키(고유키)는 비밀이다.
- 그룹의 크기가 증가할 때, 새로운 구성원들의 키들은 새롭게 생성되고, 기존 구성원들의 고유키는 변하지 않는다.
- 그룹의 크기가 감소할 때, 삭제되는 구성원들의 고유키들은 새로운 키들로부터 제거되고 (즉, 새로운 키들을 생성 하는데 삭제되는 구성원들의 고유키 값들이 이용되지 않는다), 남아 있는 구성원들의 고유키는 변하지 않는다.

아래의 사실은 그룹의 구성원들이 그룹키를 구하기 위한 최소한의 요구조건을 설명하고 있다.

[FACT 2]
자신의 고유키와 트리 경로상의 KBK들을 알고 있는 구성원이면 이 키들을 이용해 그룹키를 구할 수 있다.

4.1 시스템 초기화

하나의 멀티캐스트 그룹은 세션 0으로 초기화된다. 구성원의 집합을 $M_0=\{m_{1_0},\ldots,m_{n_0}\}$라고 놓는다. 그룹 관리자는 n_0개의 구성원-노드를 가진 트리를 생성한다. 그리고 각 노드에 랜덤하게 선택한 키를 대응시키고 유일한 인덱스를 부여한다. 그리고 각 구성원-노드를 구성원들에 대응시킨다. 그룹 관리자는 모든 노드 키들을 비밀로 유지한다. 그룹 관리자는 구성원 m에게 m의 구성원-노드부터 루트 노드까지 경로내 키들의 집합 $K_0(m)$을 안전한 유니캐스트 채널상으로 전달한다. 각 구성원 m은 $K_0(m)$을 자신의 비밀키들로 유지한다. 모든 구성원들에 의해 공유되는 그룹키는 세션키 k_{s_0}이다. 균형잡힌 이진트리를 가정하기 때문에 그룹 관리자는 $2n-1$개의 키들을 저장하고, 구성원은 $\log n+1$개의 키들을 저장해야 한다.

4.2 구성원 추가

새로운 구성원이 추가되는 사건이 발생할 때, 그룹 관리자는 새로운 구성원에게 하나의 노드(구성원-노드)를 할당한다. 그리고 새로운 구성원 m_i는 최하위 노드부터 루트 노드까지 경로내의 모든 키 $K(m_i)$를 받게된다. 새로 추가된 구성원이 추가되기 이전의 정보를 알 수 없도록 하기 위해(*backward secrecy*), 새로운 구성원이 받는 모든 키들은 이전의 그룹키들과 독립이어야 한다. 즉, 새로운 구성원이 이전의 그룹키들을 유도해 내는 것이 계산적으로 불가능해야 한다는 것이 요구된다.

이전의 스킴들을 살펴보면 키 갱신에 영향 받는 구성원들에게 갱신된 키들을 전달하기 위해 블록암호나 스트림 암호를 이용해서 암호화된 키를 멀티캐스트하거나[6,7,8,10], 인덱스들만을 멀티캐스트한다[9,12]. ELKH에서는 구성원 추가시 효율적인 키 관리를 위해서 명령어만을 멀티캐스트하는 접근 방법을 이용한다. 그룹에 구성원들이 추가되는 경우 그룹 관리자와 구성원들이 키 갱신을 어떻게 수행하는지 4.2.1절에서 자세히 다룬다.

ELKH에서 구성원과 그룹 관리자는 키를 갱신할 때 일방향 함수 h와 XOR만을 이용하므로 매우 효율적으로 키를 갱신할 수 있다. ELKH의 주된 장점은 구성원이 추가될 때, 구성원과 그룹 관리자가 키 갱신을 위해 계산해야 하는 연산량이 적다는 것이다. 그리고 그룹 관리자가 추가되는 구성원의 동기 구성원에게 키 갱신 정보를 유니캐스트로 따로 전달하는 대신에 이 정보를 멀티캐스트 메시지에 포함해서 전달한다(추가되는 구성원의 수에 따라서 멀티캐스트 메시지와 유니캐스트 메시지 사이의 trade-off가 가능하다).

4.2.1 구성원 추가시의 키 갱신 프로토콜

구성원들의 집합 $J_i\subseteq(M\backslash M_i)$가 세션 i에서 추가된다고 가정하자. 그럼 결과적으로 새로운 세션 $i+1$에서 구성원들의 집합은 $M_{i+1}=M_i\cup J_i$가 된다. 세션키 k_{s_i}는 J_i내의 구성원들이 얻을 수 없는 새로운 세션키 $k_{s_{i+1}}$로 갱신되야 한다. 여기서 $|J_i|=j$라고 놓는다.

■ 그룹 관리자

그룹 관리자는 구성원들이 추가될 때, 아래와 같이 수행한다.

① 그룹에 j개의 새로운 구성원-노드를 추가함으로써 기존의 트리 구조를 새롭게 변경한다. 이 때,

새로 추가되는 구성원-노드들을 트리에 추가시키기 위해서 왼쪽에서부터 루트까지 경로가 가장 짧은 노드 x 자리(가능한 트리를 작게 유지하기 위해)에서부터 추가해 간다: 새로운 내부 노드 p를 만들고 x를 p의 왼쪽 자식 노드로, 새로운 구성원-노드를 오른쪽 자식 노드로 추가 시키는 방식으로 모든 J_i의 각 구성원에 대해 트리 구조를 갱신한다. 그리고 갱신된 트리 T'에 영향 받는 노드들과 키들의 레벨을 재배열 한다.

② 새로 추가된 각 구성원-노드 마다 키 집합에서 랜덤하게 선택한 키를 할당하고, 새로운 구성원 $m \in J_i$들을 새로운 구성원-노드에 대응시킨다.

새로운 구성원들의 내부 키들의 집합을 $IK(J_i) = \cup_{m \in J_i} K(m) \backslash \{k_m\}$라 하자. 그리고 $IK(J_i)$내에 새롭게 생성된 노드 키들의 집합을 $NK(J_i)$라 할 때, 키 $k_n \in NK(J_i)$들을 $U(k_n) = h(k_n^{left\,right} \oplus k_{s_i})$를 통해 생성한다.

그리고 $C(NK(J_i))$가 $NK(J_i)$내의 키들을 상위 키로 갖는 키들의 집합이고, $C_{\{k_m\}}(NK(J_i)) = \{C(NK(J_i)) \backslash \{k_m\}\}$일 때 (간단히 $C_Y(NK(J_i))$로 표기한다), M_{rkey}에 $[C_Y((NK(J_i))^{I}: +NK(J_i)^{I}]$를 추가한다 ($X^I$는 X의 키 인덱스이다).

그리고 키 $k_s \in \{RK(J_i) = IK(J_i) \backslash NK(J_i)\}$들을 갱신하고: $k_s' = R(k_s)$, M_{rkey}에 $[RK(J_i)^{I}: *RK(J_i)^{I}]$를 추가한다.

③ 변경된 키들을 구성원들이 갱신할 수 있도록 하기 위해, 변경된 키 인덱스를 포함한 키 갱신 메시지 $M_{rkey} = \{[C_Y(NK(J_i))^{I}: +NK(J_i)^{I}, RK(J_i)^{I}: *RK(J_i)^{I}]\}$를 멀티캐스트한다.

④ 새로 추가되는 구성원 $m \in J_i$들에게 $K(m)$을 유니캐스트 채널로 안전하게 전달한다.

■ 구성원

M_{i+1}내의 구성원들은 트리에 위치한 자리(또는 받은 명령어)에 따라 수행하는 키 갱신 방법이 조금 다른데, 다음과 같다. 키 갱신 방법은 명령어로 제어된다.

- *추가되는 구성원의 동기들* $m_x \in S(J_i)$ ($S(J_i)$: 동기들의 집합)는 $M_{rkey} = \{C_Y(NK(J_i)^{I}: +NK(J_i)^{I}, RK(J_i)^{I}: *RK(J_i)^{I}]\}$를 받은 후, 자신의 키-리스트에 대응되는 명령어를 수행한다 : 먼저 $[C_Y(NK(J_i))^{I}: +NK(J_i)^{I}]$에 따라 새로운 노드 n_p, $p \in NK(J_i)^{I}$를 생성하고, 이 노드를 자신의 구성원-노드 n_x의 상위 노드로 배열한다. 그리고 이 노드에 대응되는 키 k_p를 자신의 고유키 k_x와 현재 자신이 가지고 있는 그룹키 k_{s_i}를 이용해서 계산할 수 있다: $U(k_x, k_{s_i})$. 다음으로 $[RK(J_i)^{I}: *RK(J_i)^{I}]$내에 대응되는 명령어를 수행한다 : 모든 $k_a \in RK(J_i)$에 $k_a' = R(k_a)$. 그리고 새로운 그룹키 $k_{s_{i+1}}$를 얻는다.
- *다른 구성원들* 새로 추가되는 구성원과 동기 구성원 이외의 그룹내의 구성원 $m_y \in O(M_i \backslash S(J_i) \cup J_i)$들은 자신들의 키 경로내의 키들 $K(m_y)$를 알고 있다. 따라서 키를 갱신할 때, 노드의 위치 변경 없이 그룹 관리자의 멀티캐스트 메시지 M_{rkey}를 보고 키들을 갱신할 수 있다 : $[RK(J_i)^{I}: *RK(J_i)^{I}]$내에 자신의 키-리스트에 대응되는 인덱스가 존재하면, 그 키에 대응하는 명령어들을 실행한다: $k_s' = R(k_s)$, $k_s \in RK(J_i)$.

(Note)

새로운 키들을 생성하는데 필요한 키들을 알고있는 사용자들은 그룹 관리자가 멀티캐스트 해주는 메시지 $M_{rkey} = \{[I$:명령어$]\}$를 가지고 스스로 키들을 갱신할 수 있다. 물론, 추가되는 구성원 이외의 구성원들은 새로 추가되는 사용자-노드에 대응하는 키를 알 수 없고, 새로 추가되는 사용자는 추가되기 이전의 키들을 알 수 없다(증명은 V장에서 기술한다).

- ***한 명의 구성원 추가 예***([그림 2]).

새로운 구성원 m_2가 그룹내에 추가될 때.

■ 그룹 관리자

① [그림 2]에서 보는 것처럼 새로운 구성원-노드 n_2와 새로운 노드 n_{12}를 추가시킴으로써 트리 구조를 갱신한다. 점선으로 표시된 노드의 레벨이 변경되므로 이 노드의 레벨을 재배열한다.

② 새로 추가되는 구성원-노드 n_2에 구성원 m_2를 대응시키고, 키 k_2를 키 집합에서 랜덤하게 선택해서 할당한다. 그리고 키들 $IK(J_i) = \{k_{12}, k_{14}, k_{18}\}$에 대해 고려한다.

먼저 $IK(J_i)$내에 새롭게 생성되는 노드 n_{12}의 키 값 $k_{12} = h(k_1 \oplus k_{18})$를 구하고, M_{rkey}에 $[1: +12]$를 추가한다. 그리고 $RK(J_i) = \{k_{14}, k_{18}\}$내의 키들을

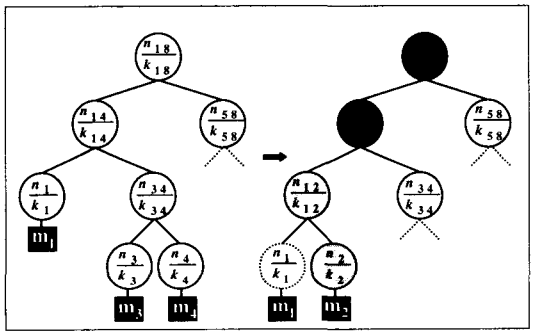
갱신한다 : $k'_{14}=h(k_{14})$, $k'_{18}=h(k_{18})$. 그리고 키 갱신 메시지 M_{rkey}에 [14:*14, 18:*18]를 추가한다.

③ 변경된 키들을 구성원들이 갱신할 수 있도록, $M_{rkey}=\{[1:+12, 14:*14, 18:*18]\}$를 멀티캐스트한다.

④ 갱신된 트리상에 새로 추가되는 구성원 m_2에 대응하는 키 $\{k_2, k_{12}, k'_{14}, k'_{18}\}$를 m_2에게 유니캐스트 채널로 안전하게 전달한다.

■ **구성원** $\{m_1, m_3, \ldots, m_8\}$

- *추가되는 구성원의 동기* m_1은 키 갱신 메시지 $M_{rkey}=\{[1:+12, 14:*14, 18:*18]\}$를 받은 후에 대응되는 첫 번째 명령어를 실행한다. 즉, 노드 n_{12}를 생성하여 이 노드를 자신의 구성원-노드 n_1의 상위 노드로 만든 후 레벨을 재배열하고, 대응되는 키를 생성한다 : $k_{12}=h(k_1 \oplus k_{18})$. 그리고 두 번째와 세 번째 명령어를 실행한다: $k'_{14}=h(k_{14})$, $k'_{18}=h(k_{18})$. 따라서 m_1은 $K(m_1)=\{k_1, k_{14}, k_{18}\}$을 알고 있으므로 스스로 키를 갱신할 수 있다.
- *다른 구성원들* $\{m_3, \ldots, m_8\} \in O(M_i \backslash S(J_i) \cup J_i)$는 $M_{rkey}=\{[1:+12, 14:*14, 18:*18]\}$를 받은 후, 자신의 키-리스트에 대응하는 인덱스 14, 18에 대응하는 명령어를 실행한다 : $k'_{14}=h(k_{14})$, $k'_{18}=h(k_{18})$.



〔그림 2〕 새로운 구성원 m_2가 그룹내에 추가될 때 갱신된 T

• ***여러 명의 구성원 추가 예***(〔그림 3〕).
$J_i=\{m_2, m_3\}$가 그룹에 추가될 때.

■ **그룹 관리자**

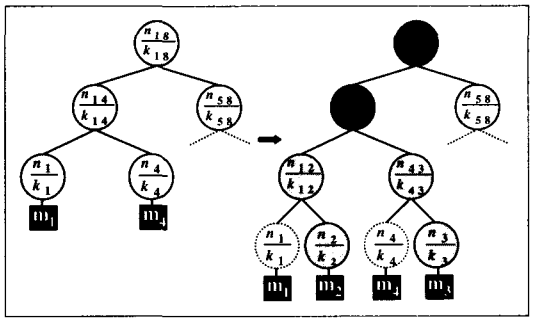
① 〔그림 3〕에서 보는 것처럼 새로운 노드 n_2, n_{12}와 n_3, n_{43}을 추가 시키고, 점선으로 표시된 노드들의 레벨이 변경되므로 이 노드와 키 레벨들을 재배열한다.

② 새로 추가되는 구성원-노드 n_2, n_3에 각각 m_2, m_3을 대응시키고, 키 k_2, k_3를 키 집합에서 랜덤하게 선택해서 각각 할당한다.

먼저 노드 n_{12}, n_{43}에 대응되는 키 $NK(J_i)=\{k_{12}, k_{43}\}$들을 생성한다 : $k_{12}=h(k_1 \oplus k_{18})$, $k_{43}=h(k_4 \oplus k_{18})$. 그리고 M_{rkey}에 [1:+12, 4:+43]을 추가한다. 또한, 키들 $RK(J_i)=\{k_{14}, k_{18}\}$을 기존의 키를 이용해서 갱신한다: $k'_{14}=h(k_{14})$, $k'_{18}=h(k_{18})$ (이 키들은 단지 한 번만 갱신된다). 그리고 키 갱신 메시지 M_{rkey}에 [14:*14, 18:*18]을 추가한다.

③ 변경된 키들을 구성원들이 갱신할 수 있도록, $M_{rkey}=\{[1:+12, 4:+43, 14:*14, 18:*18]\}$를 멀티캐스트한다.

④ 갱신된 트리상에 새로 추가되는 구성원 m_2에 대응하는 키들 $\{k_2, k_{12}, k'_{14}, k'_{18}\}$를 m_2에게, 구성원 m_3에 대응하는 키들 $\{k_3, k_{43}, k'_{14}, k'_{18}\}$를 m_3에게 유니캐스트 채널로 안전하게 전달한다.



〔그림 3〕 새로운 구성원들$\{m_2, m_3\}$가 그룹내에 추가될 때 갱신된 T'

■ **구성원** $\{m_1, m_4, \ldots, m_8\}$

- *추가되는 구성원의 동기들* $\{m_1, m_4\} \in S(J_i)$은 $M_{rkey}=\{[1:+12, 4:+43, 14:*14, 18:*18]\}$을 받은 후 m_1은 자신의 키-리스트에 대응되는 명령어를 수행하게 된다. 즉, 첫 번째 명령어대로 새로운 노드 n_{12}를 생성하여 이 노드를 자신의 구성원-노드 n_1의 상위 노드로 만든 후 레벨을 재배열하고, 대응되는 키를 생성한다 : $k_{12}=h(k_1 \oplus k_{18})$. 그리고 세 번째와 네 번째 명령어를 실행한다 : $k'_{14}=h(k_{14})$, $k'_{18}=h(k_{18})$. 구성원 m_4도 같은 방법으로 키들 $\{k_{43}, k_{14}, k_{18}\}$을 갱신한다.

- *다른 구성원들* $\{m_5, \ldots, m_8\} \in O(M_i \setminus S(J_i) \cup J_i)$ 는 M_{rkey}를 받은 후, 자신의 키-리스트와 대응되는 인덱스 18에 대응되는 명령어를 실행한다 : $k'_{18} = h(k_{18})$.

4.3 구성원 삭제

구성원 삭제 프로토콜은 구성원 추가 프로토콜보다 복잡하다. 왜냐하면 그룹에서 삭제된 구성원이 알고 있는 모든 키를 삭제된 구성원이 계산할 수 없는 새로운 키로 변경해야하기 때문이다(*forward secrecy*).

이전의 스킴들에서는 갱신된 키들을 분배할 때, 일반적으로 블록 암호나 스트림 암호 알고리즘을 이용해 갱신된 키들을 암호화한다. 그러나 ELKH는 어떤 특정한 암호 알고리즘을 사용하는 대신에 일방향 함수와 XOR 만을 이용해서 갱신된 키들을 멀티캐스트하는 새로운 접근 방법이다. 또한 키를 갱신할 때에도 효율성을 위해서 일방향 함수와 XOR을 이용한다.

4.3.1 구성원 삭제시의 키 갱신 프로토콜

세션 $i+1$에서 구성원들의 집합 $R_i \subseteq M_i$가 그룹에서 삭제되고, 새로운 구성원들의 집합을 $M_{i+1} = M_i \setminus R_i$라고 가정하자. 세션키 k_{s_i}는 단지 M_{i+1}내의 구성원들만이 구할 수 있는 값 $k_{s_{i+1}}$로 갱신된다.

■ **그룹 관리자**는 구성원들이 삭제될 때, 아래와 같이 수행한다.

① 우선 R_i내의 구성원-노드들을 제거한다. 이 때, 새롭게 변경된 트리에는 단지 하나의 자식 노드를 갖는 과잉 내부 노드들이 존재하게 된다. 이러한 과잉 내부 노드 키들의 집합을 SK라 정의한다.

먼저 M_{rkey}에 $[C(SK)^I : -SK^I]$를 추가하고, 과잉 노드 자리에 그것의 자식 노드를 대체함으로 이러한 과잉 노드와 대응되는 키들을 저장공간으로부터 제거한다. 그리고 갱신된 트리내에 영향 받는 노드들과 키들의 레벨을 재배열한다. 갱신된 트리를 T'라 하고, 레벨이 변경된 키들의 집합을 $K_a(T')$라 한다.

② 변경된 트리 T'에서 갱신되어야 하는 내부 키들의 집합을 $IK(R_i) = \cup_{m \in R_i} K(m) \setminus K_a(T')$라 한다. 이때, 그룹 관리자는 루트키를 갱신하기 위해서 먼저 $K_a(T')$중 가장 왼쪽의 구성원 m_x의 고유키 k_x를 이용해 모든 $k_j \in AK(m_x)$ 키들을 U를 통해 갱신한다 : ($k_j = h(k_j^{left/right} \oplus k_{s_i})$).

다음으로 $RK(R_i) = IK(R_i) \setminus AK(m_x)$내의 키들도 R를 통해 갱신한다. 먼저 $AK(m_x)$내의 갱신된 키들의 집합을 $AK(m_x)'$으로 정의하고, $AK(m_x)'$내의 키들을 각각 $C_{AK(m_x)}(AK(m_x)')$내의 키를 이용해 블라인드 한다.(여기에서 $C_{AK(m_x)}(AK(m_x)')$은 $AK(m_x)'$를 부모키로 갖는 키들의 집합 $C(AK(m_x)')$에서 $AK(m_x)$를 제외한 키로써, 간단히 $C_Y(AK(m_x)')$로 표기한다.)

그리고 $RK(R_i)$내의 갱신된 키들의 집합을 $RK(R_i)'$라고 하면, $RK(R_i)'$내의 키들도 각각 $C_{Ka(T')}(RK(R_i)')$내의 키를 이용해 블라인드 한다. (간단히 $C_Y(RK(m_x)')$로 표기한다.)

그리고 M_{rkey}에 $\langle AK(m_x)' \rangle_{C_Y(AK(m_x))}$, $C_Y(AK(m_x))^I : C_Y(RK(R_i))^I : \langle RK(R_i)' \rangle_{C_Y(RK(R_i))}]\}$를 추가한다.

③ 변경된 키들을 구성원들이 갱신할 수 있도록, $M_{rkey} = \{[C(SK)^I : -SK^I,\ C_Y(AK(m_x))^I : : \langle RK(R_i)' \rangle_{C_Y(RK(R_i))}]\} \langle AK(m_x)' \rangle_{C_Y(AK(m_x))}$, $C_Y(RK(R_i))^I$를 구성원들에게 멀티캐스트한다.

■ **구성원**

M_{i+1}내의 구성원들은 트리에 위치한 자리(또는 받은 명령어)에 따라 수행하는 키 갱신 방법이 조금 다른데, 다음과 같다. 키 갱신 방법은 명령어로 제어된다.

- *삭제되는 구성원의 동기*들 받은 갱신 메시지 $M_{rkey} = \{[C(SK)^I : -SK^I,\ C_Y(AK(m_x))^I : \langle AK(m_x)' \rangle_{C_Y(AK(m_x))}, C_Y(RK(R_i))^I : \langle RK(R_i)' \rangle_{C_Y(RK(R_i))}]\}$에 자신의 키-리스트에 대응되는 명령어들을 수행한다. 여기에서 구성원들은 $C(SK)^I$에 자신이 가지고 있는 키 인덱스가 존재하므로, 명령어에 따라 자신의 키-리스트에서 SK^I에 대응되는 노드와 키를 제거하고, 이 자리에 자신의 구성원-노드와 고유키를 할당하고, 레벨을 재배열한다. SK^I에 대응되는 노드와 키를 제거했기 때문에 제거된 노드의 상위 노드들의 키들을 U를 통해 갱신한다.

이하부터는 여러명의 구성원들이 동시에 삭제될 경우에만 적용된다. 만약 $[C_Y(AK(m_x))^I:\langle AK(m_x)\rangle_{C_Y(AK(m_x))}, C_Y(RK(R_i))^I:\langle RK(R_i)\rangle_{C_Y(RK(R_i))}]$ 내에 자신이 갱신한 키에 대응되는 인덱스가 존재하면(즉, 갱신한 키로 블라인드 된 키가 존재하면), 더 이상 키를 U를 이용해 갱신하지 않고, 그 인덱스에 대응되는 키로 블라인드된 키를 UB를 통해 갱신된 키를 얻는다.

- *다른 사용자들* 삭제되는 구성원의 동기 이외의 그룹내의 모든 구성원들은 자신의 키 경로내의 키들을 알고 있고, 이 키들을 갱신할 때 자신이 가지고 있는 노드의 위치 변경 없이 관리자가 멀티캐스트한 메시지를 이용하여 키를 갱신할 수 있다(키 갱신 메시지의 키들은 이들 구성원들의 키 경로내의 키 중 하나의 키로 블라인드되어 있으므로). 따라서 이 구성원들은 키 갱신 메시지내의 키-리스트에 자신이 가지고 있는 키 인덱스들이 존재하면 그 인덱스에 대응되는 키를 이용해 UB를 통해 갱신된 키를 얻을 수 있다.

▪ ***한 명의 구성원 삭제 예***([그림 4]). 구성원 m_2가 그룹내에서 삭제될 때.

■ **그룹 관리자**

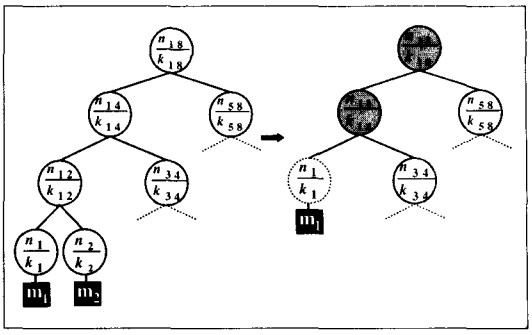
① 구성원-노드 n_2를 삭제하고, M_{rkey}에 $[1:-12]$를 추가한다. 트리내의 과잉 내부 노드 n_{12} 자리로 n_1을 올리고, 그대로 키 k_1을 할당한다. n_1과 k_1의 레벨을 변경한다.

② m_1의 고유키 k_1을 이용해 $AK(m_1)=\{k_{14}, k_{18}\}$을 갱신한다 : $k_{14}'=h(k_1\oplus k_{18})$, $k_{18}'=h(k_{14}'\oplus k_{18})$. 그리고 M_{rkey}에 $[34:\langle k_{14}'\rangle_{k_{34}}, 58:\langle k_{18}'\rangle_{k_{58}}]$을 추가한다.

③ 변경된 키들을 구성원들이 갱신할 수 있도록, $M_{rkey}=\{[1:-12, 34:\langle k_{14}'\rangle_{k_{34}}, 58:\langle k_{18}'\rangle_{k_{58}}]\}$를 멀티캐스트 한다.

■ **구성원** $\{m_1, m_3, \ldots, m_8\}$

- *삭제되는 구성원의 동기* m_1은 받은 키 갱신 메시지 $M_{rkey}=\{[1:-12, 34:\langle k_{14}'\rangle_{k_{34}}, 58:\langle k_{18}'\rangle_{k_{58}}]\}$에 따라 노드 n_{12}와 k_{12}를 제거하고, 그 자리에 자신의 구성원-노드 n_1을 올리고, 고유키 k_1을 할당한다. k_{12}가 삭제되고 그 자리에 k_1로 변경되었으므로 $AK(m_1)=\{k_{14}, k_{18}\}$의 키들을 갱신한다 : $k_{14}'=h(k_1\oplus k_{18})$,



〔그림 4〕 구성원 m_2가 그룹에서 삭제될 때 갱신된 트리 T

$k_{18}'=h(k_{14}'\oplus k_{18})$(M_{rkey}의 키-리스트에 자신이 갱신한 k_{14}', k_{18}'이 없기 때문에 $AK(m_1)$내의 모든 키를 갱신한다).

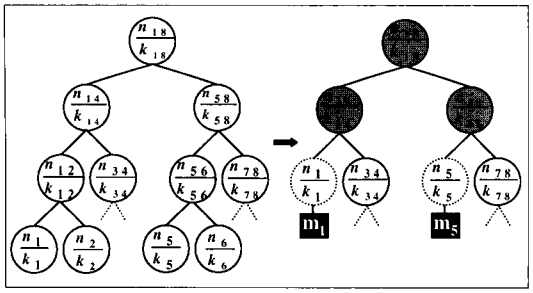
- *다른 구성원들* 먼저 $\{m_3, m_4\}$는 키 갱신 메시지 M_{rkey}를 받고, 대응되는 $[34:\langle k_{14}'\rangle_{k_{34}}]$를 실행한다. 가지고 있는키 k_{34}를 이용해 블라인드된 값 $\langle k_{14}'\rangle_{k_{34}}$으로부터 갱신된 키 k_{14}'를 얻는다: $\langle k_{14}'\rangle_{k_{34}}\oplus h(k_{34}\oplus k_{18})$. 그리고 갱신한 k_{14}'로 블라인드된 값이 없으므로, k_{18}을 갱신한다. 갱신한 k_{14}'과 이전의 그룹키 k_{18}를 이용해 갱신된 그룹키 k_{18}'을 계산할 수 있다 : $k_{18}'=h(k_{14}'\oplus k_{18})$. 마찬가지로 $\{m_5, m_8\}$은 받은 M_{rkey}에서 자신들의 키-리스트에 대응되는 $\{58:\langle k_{18}'\rangle_{k_{58}}\}$를 실행한다. 가지고 있는 키 k_{58}과 그룹키 k_{18}을 이용해 갱신된 그룹키 k_{18}'을 얻는다: $\langle k_{18}'\rangle_{k_{58}}\oplus h(k_{58}\oplus k_{18})$.

▪ ***여러 명의 구성원 삭제 예***([그림 5]) $R_i=\{m_2, m_6\}$가 그룹에서 삭제될 때.

■ **그룹 관리자**는 여러 구성원들이 동시에 삭제될 때, 아래와 같이 수행한다.

① 구성원-노드 n_2, n_6을 제거하고, 새로운 M_{rkey}에 $[1:-12, 5:-56]$을 추가한다. 그리고 트리내의 과잉 내부 노드 n_{12}, n_{56} 자리로 n_1, n_5를 차례대로 올리고, 키 k_1, k_5를 할당한다. 그리고 n_1, k_1과 n_5, k_5의 레벨을 변경함으로써 전체 트리구조를 T'으로 새롭게 변경한다.

② 새롭게 변경된 트리 T'에서 갱신해야 하는 내부 키들은 $IK(R_i)=\{k_{14}, k_{58}, k_{18}\}$이므로, 먼저 $K_a(T')$



〔그림 5〕 구성원{ m_2, m_6}가 그룹에서 삭제될 때 갱신된 T'

$=\{k_1, k_5\}$ 중 가장 왼쪽의 키 k_1을 이용해 $AK(m_1)$ $=\{k_{14}, k_{18}\}$을 갱신한다 : $k_{14}' = h(k_1 \oplus k_{18}), k_{18}' = h(k_{14}' \oplus k_{18})$. 그리고 나머지 내부키 $RK(R_i) = \{k_{58}\}$를 $k_{58}' = h(k_5 \oplus k_{18})$로 갱신한다.

($AK(m_1)$와 $RK(R_i)$의 키들을 반드시 순차적으로 갱신할 필요는 없고, 방법에 따라서 동시에 키를 갱신할 수 있다).

키 갱신이 끝난 후, $AK(m_1)' = \{k_{14}', k_{18}'\}$ 키들을 블라인드 한다 : $\langle k_{14}' \rangle_{k_{34}}, \langle k_{18}' \rangle_{k_{58}'}$. 그리고 $RK(R_i)' = \{k_{58}'\}$을 블라인드 한다 : $\langle k_{58}' \rangle_{k_{78}}$. 또한 키 갱신 메시지 M_{rkey}에 $[34: \langle k_{14}' \rangle_{k_{34}}, 78: \langle k_{58}' \rangle_{k_{78}}, 58: \langle k_{18}' \rangle_{k_{58}'}]$을 추가한다 .

③ 변경된 키들을 구성원들이 갱신할 수 있도록, M_{rkey} $=\{[1: -12, 5: -56, 34: \langle k_{14}' \rangle_{k_{34}}, 78: \langle k_{58}' \rangle_{k_{78}}, 58: \langle k_{18}' \rangle_{k_{58}'}]\}$을 멀티캐스트 한다.

■ **구성원** $\{m_1, m_3, m_4, m_5, m_7, m_8\}$

- *삭제되는 구성원의 동기들* $\{m_1, m_5\}$는 받은 갱신 메시지 $M_{rkey} = \{[1: -12, 5: -56, 34: \langle k_{14}' \rangle_{k_{34}}, 78: \langle k_{58}' \rangle_{k_{78}}, 58: \langle k_{18}' \rangle_{k_{58}'}]\}$에 따라, 구성원 m_1은 노드 n_{12}와 k_{12}를 제거하고, 그 자리에 자신의 구성원-노드 n_1을 올리고 고유키 k_1을 할당한다. 키 k_1을 삭제했으므로 $AK(m_1) = \{k_{14}, k_{18}\}$의 키들을 갱신한다 : $k_{14}' = h(k_1 \oplus k_{18})$, $k_{18}' = h(k_{14}' \oplus k_{18})$ (M_{rkey}의 키-리스트에 자신이 갱신한 k_{14}', k_{18}'로 블라인드된 키들이 존재하지 않기 때문에 $AK(m_1)$ 내의 모든 키를 갱신한다). 마찬가지로 구성원 m_5는 노드 n_{56}와 k_{56}을 제거하고 그 자리에 자신의 구성원-노드 n_5를 올린 후 고유키 k_5를 할당한다. 키 k_{56}을 삭제했으므로 $AK(m_5) = \{k_{58}, k_{18}\}$의 키들을 갱신한다. 그런데 M_{rkey}의 키-리스트에 자신이 갱신한 키에 대응하는 인덱스 58이 존재하므로, 키 k_{18}은 갱신하지 않는다 : $k_{58}' = h(k_5 \oplus k_{18})$. 그리고 이 갱신한 키 k_{58}'과 이전의 그룹키 k_{18}를 이용해서 갱신된 그룹키 k_{18}'을 얻는다 : $k_{18}' = \langle k_{18} \rangle_{k_{58}'} \oplus h(k_{58}' \oplus k_{18})$.
- *다른 구성원들* $\{m_3, m_4, m_7, m_8\}$은 키 갱신 메시지 M_{rkey}를 받은 후에 자신들의 키-리스트에 대응되는 $[34: \langle k_{14}' \rangle_{k_{34}}, 78: \langle k_{58}' \rangle_{k_{78}}, 58: \langle k_{18}' \rangle_{k_{58}'}]$을 실행한다. 구성원 $\{m_3, m_4\}$는 가지고 있는 키 k_{34}를 이용해 블라인드 메시지 $\langle k_{14}' \rangle_{k_{34}}$로부터 갱신된 키 k_{14}'을 계산한다 : $UB_{k_{34}}(k_{14}') \oplus h(k_{34} \oplus k_{18})$. 그리고 갱신한 k_{14}'와 이전의 그룹키 k_{18}을 이용해 갱신된 그룹키 k_{18}'을 계산할 수 있다 : $k_{18}' = h(k_{14}' \oplus k_{18})$.

  마찬가지로 $\{m_7, m_8\}$은 자신들의 키-리스트에 대응되는 $[78: \langle k_{58}' \rangle_{k_{78}}, 58: \langle k_{18}' \rangle_{k_{58}'}]$를 실행한다. 우선 블라인드된 키 $\langle k_{58}' \rangle_{k_{78}}$로부터 알고 있는 키 k_{78}와 그룹키 k_{18}을 이용해 키 k_{58}'을 계산한다 : $\langle k_{58}' \rangle_{k_{78}} \oplus h(k_{78} \oplus k_{18})$. 그리고 M_{rkey}에 갱신한 키 k_{58}'로 블라인드된 값 $\langle k_{18}' \rangle_{k_{58}'}$이 존재하므로, 키 k_{58}'를 이용해서 새로운 그룹키를 얻을 수 있다 : $k_{18}' = \langle k_{18}' \rangle_{k_{58}'} \oplus h(k_{58}' \oplus k_{18})$.

V. 안전성 분석

본 장에서는 ELKH 프로토콜의 안전성을 분석한다. 우선 안전한 ELKH 프로토콜을 구성하기 위해, 직관적으로 요구되는 성질들이 무엇인지 확인하고, ELKH 프로토콜이 정의 1과 2를 만족하는 **안전한** 키 갱신 스킴이라는 것을 증명한다. 먼저 ELKH에서 이용되는 일방향 함수 h와 블라인드 함수 BK의 안전성에 대한 가정들을 분명히 한다.

5.1 프리미티브들의 가정

이 절에서는 함수 h와 블라인드 함수 BK의 중요한 성질들을 정의한다.

[성질 1]

함수 h는 일방향이다: $h(x)$가 주어졌을 때, 어떤 무시할수 없는 확률로 x를 알아내는 것이 계산적으로 불가능하다.

이 성질은 구성원 추가시 $k_x' = h(k_x)$가 주어졌을

때, 노드키 k_x를 찾는 것을 불가능하게 만든다. 또한 구성원 추가, 삭제시에 $k_z = h(k_x \oplus k_y)$와 k_x가 주어졌을 때, 알려지지 않은 키 값 k_y를 찾는 것을 불가능하게 만든다. 이 성질은 가장 최근에 삭제된 구성원이 공모로부터 유용한 정보를 얻을 수 없음으로, 삭제된 구성원들의 완전한 집합(삭제된 모든 구성원들의 집합)이 공모를 한다고 해도 새로운 키 값을 알아내는 것을 막는다.

위의 성질로부터 아래의 성질을 이끌어낼 수 있다. 아래의 성질 2는 멀티캐스트되는 메시지인 블라인드 값들로부터 키 값을 알아내려는 공격을 막기 위한 것이다. 사건이 발생할 때마다 그룹키가 변경되므로, 멀티캐스트되는 블라인드 메시지도 매 사건마다 변경되게 된다.

[성질 2]

그룹키 GK와 블라인드 메시지 $\langle x' \rangle_k = x' \oplus h(k \oplus GK)$가 주어졌을 때, 이것으로부터 알려진 값 이외의 어떤 다른 키 값에 대해 무시할 수 없는 확률로 정보를 알아내는 것이 계산적으로 불가능하다.

ELKH 프로토콜의 안전성은 일방향 함수의 암호학적 특성들에 의존한다. 그러나 일방향 해쉬함수들은 증명된 안전성이 아니다[3]; 하지만 아직까지는 완전한 MD5[1]나 또는 SHA[2] 알고리즘들[5] 상에 성공적인 어떤 공격도 존재하지 않는다. 따라서 함수 h를 해쉬 함수들로 고려할 때, 숨겨진 키에 대한 공격은 brute- force공격으로 제한된다.

5.2 안전성 정리

다음 정리 1에서 ELKH 스킴이 *backward secrecy*를 만족한다는 것을 증명한다.

[정리 1]

함수 h가 성질 1을 만족할 때, ELKH 키 갱신 스킴은 정의 1의 관점에서 안전하다.

[증명(Sketch)]

트리에 한 명의 구성원이 추가될 경우, 추가되는 구성원에 대응되는 키들은 새로운 구성원에게 과거의 정보를 주는 것을 막기 위해 변경된다. 일반성을 잃지 않고 세션 j에서 새로운 구성원의 구성원-노드 n_i가 트리내에 추가되는 경우를 생각해 보자. 이 때, 새로운 구성원 m_i는 그룹 관리자로부터 자신의 키 집합 $K(m_i) = \{k_i, AK(m_i)\}$를 받게된다 : k_i는 새로운 구성원 m_i의 고유키로써 그룹 관리자에 의해 키 집합에서 랜덤하게 선택된 난수값이고, k_i의 부모키부터 루트키 까지 경로내의 키 집합 $AK(m_i)$내의 키들은 m_i가 추가되기 이전의 키 값들을 이용해 일방향 함수 h를 통해 생성된 값이다. 따라서 새로 추가된 구성원 m_i는 자신이 받은 키들을 예전의 키 값들을 알아내기 위해서 이용할 수 없다. 왜냐하면 새로운 구성원은 일방향 함수에 의해 숨겨진 새로운 값들을 받게되고, 일방향 함수는 안전하다(무시할 수 없는 확률로 역원을 알아내는 것은 계산적으로 불가능 하다)고 가정하기 때문에 *brute-force* 공격 이외에 이전의 키들을 알아낼 수 있는 다른 방법이 존재하지 않는다.

위에서의 논의는 세션을 의미하는 인덱스 j와 새로운 구성원을 나타내는 인덱스에 대한 귀납적 방법을 통해 쉽게 전체 세션과 여러 구성원 추가사건의 경우에 대한 일반적인 논의로 확장되어짐을 알 수 있다.

따라서 ELKH 키 갱신 스킴은 정의 1의 관점에서 안전하다. ∎

정리 2에서는 ELKH 스킴이 *forward secrecy*를 만족한다는 것을 증명한다. 우선 단일 구성원이 삭제되는 경우를 살펴보고, 여러 구성원들이 동시에 삭제되는 일반적인 경우를 살펴보기로 한다.

[정리 2]

함수 h가 성질 1을 만족하고, 함수 BK가 성질 2를 만족할 때, ELKH 키 갱신 스킴은 정의 2의 관점에서 안전하다.

[증명(Sketch)]

삭제사건이 발생하는 경우, 그룹내의 키 갱신 작업은 크게 두 부분으로 나누어 생각할 수 있다. 첫째, 삭제된 구성원이 가졌던 키 값들을 삭제된 구성원이 계산할 수 없는 새로운 키 값들로 갱신한다. 만약 삭제되는 구성원의 동기 구성원이 존재하는 경우에 이 동기 구성원은 자신의 고유키를 이용해서 스스로 변경된 키 값들을 계산한다. 둘째, 그룹 관리자는 삭제된 구성원과 동기 구성원 이외의 그룹내의 모든 구성원들에게 안전하지 않은 멀티캐스트 채널로 갱신된 키 정보를 분배한다. 이 때, 그룹 관리자는 이

전에 다른 구성원들과 공유된 노드키를 사용하여 갱신된 키 값들을 블라인드 (은닉)한다.

이제 구체적으로 스킴의 안전성을 살펴보자.

세션 j에서 일반성을 잃지 않고, 구성원 m_i가 삭제되는 경우를 가정해 보자. 이 증명에서는 삭제사건이 발생하기 이전의 트리가 균형 잡혀 있고, $n \geq 4$로 가정한다. 그리고 l을 삭제사건이 발생하기 이전 트리의 깊이로, k_i^l을 삭제된 구성원 m_i의 고유키로 정의하고, 키 k_i^l의 부모키부터 루트키까지 경로내의 키들을 $\{k_i^{l-1}, \ldots, k_i^0\}$로 정의한다. 그러면 이 경우에 구성원 m_i가 알고 있는 기존의 정보들은 $I(m_i) = \{k_i^l, k_i^{l-1}, \ldots, k_i^0\}$이고, 이외의 정보는 알지 못 한다고 가정한다.

먼저 그룹 관리자는 구성원 m_i의 구성원-노드 n_i를 삭제하고, m_i가 소유했던 키 값들 $AK(m_i) = \{k_i^{l-1}, \ldots, k_i^0\}$을 변경함으로써 키 트리 구조를 갱신한다. 세부적으로 살펴보면, 삭제되는 구성원 m_i의 동기 구성원을 m_i^{sib}라고 할 때, 키 k_i^{l-1}은 제거되고 그 자리에 m_i^{sib}의 고유키 k_i^{sib}가 대응된다. 그리고 기존의 노드키 값들 $\{k_i^{l-2}, \ldots, k_i^0\}$은 새로운 값 $k_i^{l-2'} = h(k_i^{sib} \oplus k_i^0), \ldots, k_i^{0'} = h(k_i^{1'} \oplus k_i^0)$들로 갱신된다. 주목해야 할 것은 키 갱신 작업에서 새로운 노드키 값들이 삭제된 구성원 m_i의 동기 구성원 m_i^{sib}의 고유키 k_i^{sib}을 사용하여 루트까지 순차적으로 유도된다는 것이다. 여기서 다음과 같은 사실을 쉽게 알 수 있다. "삭제된 구성원 m_i가 알고 있던 기존의 정보 $\{k_i^l, k_i^{l-1}, \ldots, k_i^0\}$로부터 얻을 수 있는 갱신된 키 정보는 주어진 키 공간의 확률분포에 따라 임의로 추측하는 것뿐이다." 여기서 갱신된 키 값들은 삭제된 구성원 m_i에게는 알려지지 않은 값이다. 따라서 m_i가 갱신된 키 값들을 알기 위해서는 일방향 함수 h의 특성상 입력값을 반드시 알아야 한다. 따라서 갱신된 키 값 $k_i^{l-2'} = h(k_i^{sib} \oplus k_i^0), \ldots, k_i^{0'} = h(k_i^{1'} \oplus k_i^0)$들의 입력값들은 $k_i^{sib} \oplus k_i^0, \ldots, k_i^{1'} \oplus k_i^0$이다. 비록 바로 전 세션의 그룹키 k_i^0가 m_i에게 알려진 값이지만, $k_i^{0'}$이 앞의 해쉬값 $k_i^{1'}$으로 계산되어 지므로 귀납적 방법으로 인해 결과적으로는 k_i^{sib}의 정보를 m_i가 알도록 요구하게 된다. 하지만 이것은 초기 가정에 모순이 된다.

사실 구성원 m_i가 이용할 수 있는 정보는 사건이 발생하기 이전에 소유했던 $I(m_i) = \{k_i^l, k_i^{l-1}, \ldots, k_i^0\}$ 이외에, 그룹 관리자가 변경된 키 값들을 구성원 m_i^{sib}를 제외한 다른 그룹 구성원들에게 알려주기 위해 멀티캐스트하는 키 갱신 메시지가 있다. 하지만 이 경우에도 추가적으로 이용 가능한 정보가 새로 변경된 키 값들에 대한 정보와는 독립이며 임의의 난수값에 불과함을 다음과 같이 쉽게 알 수 있다.

본 증명에서 "그룹 관리자→{A:키정보}"는 그룹 관리자가 구성원 A의 집합에게 전달하는 키정보의 집합을 나타내고, "$Sub_d(k)$"가 키 k의 동기(sibling) 키를(깊이가 낮은) 상위키로 가지는 구성원들의 집합으로써 d번째로 큰 집합을 나타내고, "$K(Sub_d)$"이 집합 Sub_d의 최상위 노드키(즉, 부분집합의 루트키)를 나타낸다고 할 때, 구성원 m_i가 그룹에서 삭제될 경우에 멀티캐스트 메시지는 다음과 같다.

그룹 관리자 →

$$\{Sub_1(k_i^{l-1'}) : \langle k_i^{l-2'} \rangle_{K(Sub_1)} = k_i^{l-2'} \oplus h(K(Sub_1) \oplus k_i^0),$$
$$\vdots$$
$$Sub_{d-1}(k_i^{0+1'}) : \langle k_i^{0'} \rangle_{K(Sub_{d-1})} = k_i^{0'} \oplus h(K(Sub_{d-1}) \oplus k_i^0)\}.$$

키 $\{K(Sub_1), \ldots, K(Sub_{d-1})\}$들은 삭제된 구성원 m_i와는 독립적인 임의의 다른 구성원들의 노드키 값들이며(이것은 가정에 의해 m_i가 알지 못 한다), 변경된 키값 $\{k_i^{l-2'}, \ldots, k_i^{0'}\}$들 역시 m_i가 알지 못하는 동기 구성원 m_i^{sib}의 고유키 값인 k_i^{sib}으로 유도된 값들이다. 그러므로 삭제된 구성원 m_i가 이용할 수 있는 키 값은 k_i^0 하나 뿐이다. 따라서 주어진 함수 h의 일방향 특성을 이용하면 이 경우에도 제시된 스킴이 안전함을 알 수 있다. 위에서 언급한 단일 구성원 삭제의 논의는 임의의 세션에 대한 여러 구성원이 동시에 삭제되는 경우로 쉽게 일반화된다. 두 명의 구성원이 동시에 삭제되는 경우를 생각해 보자 (세명 이상의 구성원이 동시에 삭제되는 경우는 두 명의 구성원이 삭제되는 경우의 귀납적 방법을 통해 같은 논의로 쉽게 증명된다).

두 명의 구성원이 삭제되는 경우, 키 갱신 작업은 중복된 그룹키 이외에는 단일 구성원 삭제시와 비슷하게 이루어진다. 즉, 각 구성원들에 이르는 노드키들은 각각 독립적으로 서로 다른 동기 구성원의 비밀키이므로 루트전 노드키까지 순차적으로 계산되어

진다. 따라서 갱신된 키 정보를 알아낼 수 있는 확률은 일방향 함수의 일방향 특성으로 인해 무시할 수 있는 양이다.

마지막으로 그룹키 경우도 삭제된 구성원들이 알지 못 하는 루트 노드의 자식 노드에 대응하는 키 값이 일방향 함수의 입력값으로 들어가 함수값이 계산되어 지므로 역시 일방향 함수의 특성을 이용하면 이 두 삭제된 구성원이 알아내기는 어려운 키 값이다.

따라서 다수의 구성원이 삭제되는 경우에도 ELKH가 정의 2의 관점에서 안전하다. ∎

Ⅵ. 효율성 분석

본 장에서는 1장에서 소개한 다른 스킴들 : EHBT[3] (Sandro 등), PRGT[4](Canetti 등), LKH$^+$(Caronni 등), OFT(McGrew 와 Sherman)와 이 논문에서 새롭게 제안한 ELKH 스킴의 파라메터들을 비교한다. 본 장에서는 그룹 관리자의 계산량과 추가되는 구성원(추가시)의 계산량, 동기 구성원의 계산량(추가되는 구성원/삭제되는 구성원의 동기)과 그 이외의 그룹에 다른 구성원들의 계산량과 멀티캐스트되는 키 갱신 메시지의 크기와 유니캐스트 메시지의 크기 기준에 초점을 맞춘다.

아래의 표들에서는 아래의 기호들을 사용한다.

n : 그룹내 구성원들의 수
d : 트리의 깊이(균형잡힌 트리에 대해서 $d=\log_2 n$)
I : 하나의 키 인덱스의 bits사이즈
K : 키 하나의 bits사이즈
G : 키 생성 연산
H : 해쉬 함수 연산
X : XOR 연산
E : 암호화 연산
D : 복호화 연산

〔표 1〕은 구성원 한 명 추가시에 그룹 관리자에게 요구되는 계산량과 추가되는 구성원과 추가되는 구성원의 동기 구성원의 계산량, 그리고 그 이외의 다른 구성원들의 계산량을 비교한다. 여기에서 다른 구성원들의 계산량은 모두 최악의 경우에 대해 비교한다. 예를 들어 〔그림 2〕에서구성원들 $\{m_3, m_4\}$의 경우에 대해서 고려한다. 그리고 〔표 2〕에서는 추가되는 구성원과 추가 되는 구성원의 동기에게 유니캐스트로 보내는 메시지의 크기와 추가 오퍼레이션들을 수행하는 동안에 멀티캐스트되는 메시지의 크기

〔표 1〕 단일 구성원 추가시 계산량

스킴	그룹 관리자	추가되는 구성원	동기 구성원	다른 구성원
ELKH	$G+(d+1)E+dH+X$	$(d+1)D$	$dH+X$	$(d-1)H$
EHBT	$G+(d+1)(E+H+X)$	$(d+1)D$	$(d+1)(H+X)$	$(d-1)(H+X)$
PRGT	$2G+(d+1)E+dH$	$(d+1)D$	$D+dH$	$D+(d-2)H$
LKH$^+$	$2G+(d+1)E+dH$	$(d+1)D$	$D+dH$	$D+(d-2)H$
OFT	$G+3dE+(d+1)H+dX$	$(d+1)D+d(H+X)$	$2D+d(H+X)$	$D+(d-2)H+(d-1)X$

〔표 2〕 단일 구성원 추가시 키 갱신 메시지의 크기

스킴	추가되는 구성원 – 유니캐스트–	동기 구성원 – 유니캐스트–	멀티캐스트
ELKH	$(d+1)K$	0	$(d+1)I$
EHBT	$(d+1)K$	I	dI
PRGT	$(d+1)K$	$I+K$	dI
LKH$^+$	$(d+1)K$	$I+K$	dI
OFT	$(d+1)K$	$I+2K$	$(d+1)K$

〔표 3〕 여러명의 구성원 추가시 계산량

스킴	그룹 관리자	추가되는 구성원	동기 구성원
ELKH	$nG+n(d+1)E+(2n-1)H+nX$	$(d+1)D$	$dH+X$
EHBT	$nG+n(d+1)E+(3n-1)(H+X)$	$(d+1)D$	$(d+1)(H+X)$
PRGT	$2nG+n(d+2)E+(n-1)H$	$(d+1)D$	$D+dH$
LKH$^+$	$2nG+n(d+2)E+(n-1)H$	$(d+1)D$	$D+dH$
OFT	$nG+(nd+5n-1)E+(4n-2)(H+X)$	$(d+1)D+d(X+H)$	$2D+d(H+X)$

3. EHBT에서는 encryption과 decryption에 RC5 알고리즘을 사용하고, MD5 해쉬함수를 사용한다.
4. Canetti는 PRG 함수를 사용하기 위해 알고리즘을 명확히 지정하지 않았기 때문에, 본 논문에서는 RC5 알고리즘을 사용 한다고 가정한다.

에 대해서 비교한다.

〔표 3〕과 〔표 4〕에서는 여러명의 구성원들이 동시에 추가될 경우에 대해서 분석한다. 각 비교 파라메터들은 〔표 1〕과 〔표 2〕에서 사용된 파라메터들과 같다. 표 3과 표 4의 식들은 기존의 구성원들의 수가 두 배가 되는 경우에 유효하다. 즉, 기존의 구성원들이 각자 하나의 동기 구성원을 얻고, 기존 트리내의 키들에 영향을 받는 모든 경우에 대해서 고려한다. 이것은 추가 오퍼레이션 동안에 최악의 경우를 나타낸다. 이 표들에서는 n을 추가사건이 발생하기 이전의 그룹의 구성원들의 수로, d를 새롭게 변경된 트리의 깊이로 표기한다.

ELKH는 표에서 보듯이 다른 스킴들과 계산량을 비교해서 더 효율적이다.

〔표 5〕에서 단일 구성원 삭제시 그룹 관리자의 계산량, 삭제되는 동기의 계산량과 그 이외의 구성원들의 계산량과 멀티캐스트되는 키 갱신 메시지에 대해 비교한다. 다른 구성원의 계산량은 최악의 경우를 고려한다. 예를 들어 〔그림 4〕에서 구성원들 $\{m_3, m_4\}$.

그리고 여러명의 구성원이 동시에 삭제될 때의 연산량들을 〔표 6〕에서 분석한다. 여러명의 구성원들이

〔표 4〕 여러명의 구성원 추가시 키 갱신 메시지의 크기

스킴	추가되는 구성원 - 유니캐스트-	동기 구성원 - 유니캐스트-	멀티캐스트
ELKH	$n(d+1)K$	0	$(2n-1)I$
EHBT	$n(d+1)K$	$n\ I$	$(n-1)I$
PRGT	$n(d+1)K$	$n\ I+K$	$(n-1)I$
LKH$^+$	$n(d+1)K$	$n\ I+K$	$(n-1)I$
OFT	$n(d+1)K$	$n\ I+2K$	$(2n-2)K$

〔표 5〕 단일 구성원 삭제시 계산량과 키 갱신 메시지

스킴/ 리소스	계산량			멀티 캐스트
	그룹 관리자	동기 구성원	다른 구성원	
ELKH	$2(d-1)H+3(d-1)X$	$(d-1)(H+X)$	$(d-1)H+dX$	$I+(d-1)K$
EHBT	$d(E+H+X)$	$d(H+X)$	$D+(d-1)(H+X)$	$I+(d-1)K$
PRGT	$(2d+1)E$	$D+dE$	$(d-1)E$	$I+(d+1)K$
LKH$^+$	$2dE$	dD	dD	$I+2dK$
OFT	$d(E+H+X)$	$D+d(H+X)$	$D+(d-2)H+(d-1)X$	$I+(d+1)K$

〔표 6〕 여러명의 구성원 삭제시 계산량과 키 갱신 메시지

스킴/ 리소스	계산량		멀티 캐스트
	그룹 관리자	동기 구성원	
ELKH	$2(n-1)H+3(n-1)X$	$dH+(d+1)X$	$nI+(n-1)K$
EHBT	$(n-1)E+(2n-1)(H+X)$	$D+(d+1)(H+X)$	$nI+(n-1)K$
PRGT	$(5n/2-2)E$	$D+dE$	$(3n/2-1)K$
LKH$^+$	$(2n-2)E$	dD	$nI+2(n-1)K$
OFT	$(3n-2)E+(2n-2)H+(n-1)X$	$(d+1)D+d(H+X)$	$nI+(3n-2)K$

이 그룹에서 삭제될 때, 여기에서는 구성원들의 정확히 반이 그룹에서 삭제되는 상황을 고려한다. 삭제되는 모든 구성원들의 동기들이 트리내에 남게되고, 결과적으로 트리내의 모든 키들이 영향을 받는다. 이 표들에서는 n을 삭제사건이 발생하기 이전의 그룹의 구성원들의 수로, d를 새롭게 변경된 트리의 깊이로 표기한다.

삭제 오퍼레이션에 대해서도 마찬가지로 ELKH는 다른 스킴들 보다 계산량에 대해서 더 효율적이다. 왜냐하면 ELKH는 그룹 관리자가 갱신된 키를 분배할 때, 암호화 과정을 거치지 않고 일방향 함수와 XOR연산만을 사용해서 갱신된 키들을 블라인드하기 때문에 다른 구성원들이 갱신된 키를 얻기 위해서 복호화 과정 없이, 단지 일방향 함수연산과 XOR연산만을 수행하면 되고, 동기 구성원의 고유키가 변하지 않기 때문이다.

〔표 7〕은 여러명의 구성원들이 동시에 추가되는 경우에 그룹 관리자와 추가되는 구성원과 추가되는 구성원의 동기 구성원이 키 갱신에 소요하는 계산속도를 분석한 것이고, 〔표 8〕은 여러명의 구성원들이 삭제되는 경우에 계산속도를 분석한 것이다.

이 결과치들은 EHBT[12]의 분석자료를 이용한 것이다. 즉, 암호화, 복호화에 RC5 알고리즘을 사용하고, MD5 해쉬함수를 사용한다. 그리고 16비트 길이의 키들을 사용하고, Java 버젼 1.3과 850Mhz Mobile Pentium III 프로세서 상에 IAIK[13] 암호툴킷을 사용한다.

그리고 다음의 환경하에서 키 갱신시에 어느 정도의 계산량이 요구되는지 실험한다. 그룹에 16384명의 구성원이 존재한다고 가정한다. 그리고 여러명의 구성원 추가 오퍼레이션 동안에 기존의 구성원의 두 배가

〔표 7〕 여러명의 구성원 추가시 계산속도

스킴	그룹 관리자	추가되는 구성원	동기 구성원
ELKH	*5099*	*0.25*	*0.08*
EHBT	*5232*	*0.25*	*0.11*
PRGT	*5393*	*0.25*	*0.09*
LKH$^+$	*5393*	*0.25*	*0.09*
OFT	*6346*	*0.35*	*0.13*

〔표 8〕 여러명의 구성원 삭제시 계산속도

스킴	그룹 관리자	동기 구성원
ELKH	*240.33*	*0.09*
EHBT	*496.08*	*0.11*
PRGT	*704.56*	*0.26*
LKH$^+$	*563.57*	*0.24*
OFT	*1033.62*	*0.35*

증가한 32768의 구성원이 그룹내에 존재하게 된다고 가정한다. 또한 여러명의 구성원 삭제 오퍼레이션 동안에는 추가된 모든 구성원이 그룹에서 삭제된다고 가정한다. 결국, 그룹에는 기존의 16384명의 그룹 구성원이 존재하게 된다. 그리고 연산량을 측정하는 기준은 다음과 같다.

G : 16비트 키를 생성할 경우에 $7.33 \cdot 10^{-3}$ms가 소요
E : 16비트 키를 이용해서 16 비트 키를 암호화하는 경 우에 $1.72 \cdot 10^{-2}$ms가 소요
D : $1.73 \cdot 10^{-2}$ms가 소요
H : 16비트 키를 해쉬할 경우에 $4.95 \cdot 10^{-3}$ms가 소요
X : 16비트 키를 XOR할 경우에 $1.59 \cdot 10^{-3}$ms가 소요

Ⅵ. 결 론

키 갱신 메시지를 분배할 때, 암·복호화 과정 없이 일방향 함수와 XOR 연산을 사용하고, 키 갱신 시에도 효율적인 일방향 함수와 XOR 연산을 사용함으로 인해서 다른 LKH 프로토콜들보다 전체적으로 성능이 뛰어나면서 안전한 ELKH 프로토콜을 구성했다.

본 논문에서 제시한 새로운 프로토콜 ELKH는 다른 LKH 프로토콜들과 비교해서 그룹 관리자와 그룹 구성원의 키 저장공간과 멀티캐스트되는 키 갱신 메시지의 크기가 증가하지 않으면서, 그룹내의 구성원들의 변동이 발생할 경우에 (특히) 그룹 관리자와 그룹 구성원들이 키 갱신에 요구되는 계산량이 작다. 결과적으로 키 갱신 속도가 향상된다.

보통 그룹 관리자가 갱신된 키들을 분배할 때, 안전하지 않은 멀티캐스트 채널로 키 갱신 정보를 전달하기 때문에 정당한 구성원들만이 키를 갱신할 수 있도록 하기 위해 보통 블록 암호나 스트림 암호를 사용해서 메시지를 암호화 한다. 그러나 본 논문에서 제시한 ELKH 프로토콜은 기존의 스킴들과 다른 접근 방법을 제시한다. 즉, 키 갱신 메시지를 암호화하는 대신에 일방향 함수와 XOR연산자 만을 이용해서 메시지를 은닉한다.

사실 블록 암호나 스트림 암호 알고리즘들의 연산량과 일방향 함수와 XOR의 연산량이 크게 차이나지는 않는다. 하지만 그룹의 가입자의 수가 매우 많고 변동이 빈번한 그룹들에 안전하고 효율적인 어플리케이션들이 요구될 때, ELKH 프로토콜은 키 갱신에 소요 되는 계산량을 줄이는 실용적인 접근 방법을 제공한다. 또한, 그룹 관리자나 그룹 구성원들이 키 갱신에 소요하는 속도가 매우 중요시되는 모바일 멀티캐스트 등에 유용하게 사용될 수 있다.

참 고 문 헌

〔1〕 R. Rivest, "The MD5 Message-Digest Algorithm", *RFC 1321*, April 1992.

〔2〕 N. F. P, "Secure Hash Standard", *Nationa l Institute of Standards and Technolgy, U.S. Department of Commerce, DRAFT*, pp. 180 ~1 , May 1994.

〔3〕 S. Bakhtiari, R. Safavi-Naini, and J. Pieprz yk. "Cryptographic Hash Functions: A survey", *University of Wollongong, Tec hnical Report*, pp. 95~09, July 1995.

〔4〕 B. Schneier, "Applied Cryptography Sec ond Edition: Protocols, algorithms, and source code in C", John Wiley & Sons, Inc., 1996. ISBN 0-471-11709-9.

〔5〕 W. Stallings, "Cryptography and Network Security", Prentice.Hall, 1998. ISBN 0-13 8-69017-0.

〔6〕 D. A. McGrewa and A. T. Sherman, "Key Establishment in Large Dynamic Group s Using One-Way Function Trees",

Tech nical Report No. 0755, TIS Labs at Network Associates, Inc., Glenwood, M D, May 1998.

[7] R. Canetti, J. Garay, G. Itkis, D. Micc iancio, M. Naorr, and B. Pinkas, "Mul ticast Security: A Taxonomy and Some Effcient Constructions", *In Proc. of IN FOCOM 99*, volume 2, pp. 708~71 6, New Yok, NY, USA, March 1999.

[8] D. Wallner, E. Harder, and R. Agree, "Key Management for Multicast: Issues and Architectures", *RFC 2627*, June 19 99.

[9] M. Waldvogel, G. Caronni, D. Sun, N. Weiler, and B. Plattner, "The VersaKey Framework: Versatile Group Key Ma nagement", *IEEE Journal on Selected Areas in Communications*(Special Issue on Middle-ware), pp. 17(9): 1614.16 31, August 1999.

[10] I. Chang, R. Engel, D. Kandlur, D. Pen darakis and D. Saha, "Key Manage-ment for Secure Internet Multicast Using Boolean Function Minimization Techniques:, *Proceedings of INFOCO M' 99*, pp. 689~698, 1999

[11] H. Kurnio, R. Safavi-Naini, and Huaxio ng Wang, "A Secure Re-keying Scheme with Key Recovery Property", *ACISP 2002*, LNCS 2384, pp. 40~55, 2002.

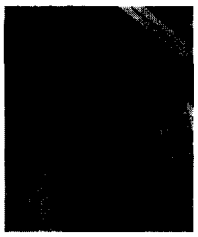
[12] S. Rafaeli, L.Mathy, and D. Hutchison, "EHBT: An Efficient Protocol for Group Key Management", J. Crowcroft and M. Hofmann(Eds.): *NGC 2001*, LNCS 2233, pp. 159~171. 2001.

[13] I.-J. Group. IAIK, java-crypto toolkit. Web site at http://jcewww.iaik.tugraz.ac.at/index.htm.

〈著者紹介〉

권 정 옥 (Joung-ok Kwon) 학생회원
2000년 2월 : 동덕여자대학교 전자계산학과 학사
2000년 9월~현재 : 고려대학교 정보보호기술학과 석사과정
〈관심분야〉 암호 프로토콜, 암호이론

황 정 연 (Jung Yeon Hwang) 학생회원
1999년 2월 : 고려대학교 수학과 학사
2001년 3월~현재 : 고려대학교 정보보호대학원 석사과정
〈관심분야〉 공개키 암호 알고리즘, 암호 프로토콜



김 현 정 (Hyun-Jeong Kim) 학생회원
1994년 2월 : 경희대학교 수학과 졸업
1994년 1월~1999년 12월 : 삼성SDS 근무
1999년 9월~2001년 8월 : 고려대학교 수학과 석사
2001년 9월~현재 : 고려대학교 정보보호 대학원 박사과정
〈관심분야〉 암호이론, 암호 프로토콜, 정보은닉

이 동 훈 (Dong Hoon Lee) 정회원
1984년 2월 : 고려대학교 경제학과 학사
1987년 2월 : Oklahoma Univ. 전산학 석사
1992년 2월 : Oklahoma Univ. 전산학 박사
1993년 3월~현재 : 고려대학교 전산학과 정교수
2000년 3월~현재 : 고려대학교 정보보호대학원 교수
〈관심분야〉 암호이론, 암호 프로토콜, 정보이론

임 종 인 (JongIn Lim) 정회원
1980년 2월 : 고려대학교 수학과 학사
1982년 2월 : 고려대학교 수학과 석사
1986년 2월 : 고려대학교 수학과 박사
1986년 3월~현재 : 고려대학교 수학과 정교수
1999년 2월~현재 : 고려대학교 자연과학대학 정교수, 고려대학교 정보보호 대학원 원장, 고려대학교 정보보호 기술센타장
〈관심분야〉 블록 암호 및 스트림 암호 분석 및 설계, 암호 프로토콜, 공개키 암호 알고리즘 분석, 스테가노 그래피