情報保護學會論文誌
第 22 卷 第 5 號, 2012. 10

선형 암호문을 이용하는 그룹 서명 기법에서 정상적인 사용자를 위한 예외 처리*

강 전 일,[1†] 양 대 헌,[1] 이 경 희[2‡]
[1]인하대학교, [2]수원대학교

Exception Management of Well-behaved Users in Group Signature Schemes based on Linear Encryption*

Jeonil Kang,[1†] DaeHun Nyang,[1] KyungHee Lee[2‡]
[1]INHA University, [2]The University of Suwon

요 약

그룹 서명 기법에서의 Open과 Judge는 잘못된 행동을 하는 사용자를 처리하기 위해 동원될 수 있는 방법 중에 하나이다. 불행하게도 어떠한 그룹 서명의 Open과 Judge는 같은 선형 암호문을 사용하는 다른 그룹 서명에서 사용될 때, 정상적인 사용자의 뜻하지 않은 오류를 처리하는 데에는 적합하지 않을 수도 있다. 이 논문에서는 선형 암호문을 이용하는 모든 그룹 서명 기법을 위해서 정상적인 사용자의 뜻하지 않은 오류 처리를 위한 Open과 Judge에 대해서 알아보고, 이를 대체하기 위한 기법들을 제안한다.

ABSTRACT

Open and Judge in group signature schemes can be brought into the management of misbehaving users. Unfortunately, when Open and Judge of a certain group signature are used into another group signature that adopts the same linear encryption, they are not suitable for processing exceptions due to well-behaved users. In this paper, for all group signatures based on the linear encryption, we propose and discuss new Open and Judge that are suitable for processing exceptions due to well-behaved users.

Keywords: group signature, linear encryption, anonymity

I. 서 론

서비스 제공의 증거로 디지털 서명을 받는 것은 자연스러운 일이다. 국내 은행들과 지불 대행업체들은 이미 자금 이체와 지불 등의 증거로써 사용자들에게 디지털 서명을 요구하고 있다. 이러한 디지털 서명에는 사용자의 실명에 대한 정보가 들어 있고, 이를 근거로 사용자들에게 서비스 제공의 대가를 요구하고 있다. 그러나 이러한 지불 시스템은 사용자의 개인 정보에 대한 별다른 인식 없이 이루어지고 있다. 이는 앞으로 익명성을 가진 서명 기법이 자금 이체와 지불 등의 증거로써 사용될 수 있는 이유이기도 하다.

그룹 서명 기법은 서명자의 익명성을 제공하는 서명 기법 중에 하나로, D. Chaum의 기본 개념의 제안[1] 이후로 꾸준히 연구가 이루어졌다. 특히 최근에는 그룹 서명의 응용에 관한 연구가 이루어지고 있다. 그룹 서명 기법은 특히나 사용자의 익명성에만 신경 쓴 것이 아니라 익명성을 악용하는 경우에 대한 고려도 포함하고 있어, 시스템에서 그룹 서명 기법을 어떻

접수일(2012년 4월 23일), 수정일(2012년 9월 14일), 게재확정일(2012년 10월 8일)
* 이 논문은 인하대학교의 지원에 의하여 연구되었음.
† 주저자, dreamx@isrl.kr
‡ 교신저자, khlee@suwon.ac.kr

게 구성하고 사용하느냐에 따라 개인 정보 보호와 공익을 균형 있게 관리할 수 있다. 그러나 그룹 서명 기법은 오로지 잘못된 행동을 반복하는 사용자에 대한 고려만 주로 되어 있어, 정직한 사용자의 의도하지 않은 오류에 대해서 지나치게 민감하게 반응하여, 사용자의 익명성을 취소하려고 할지도 모른다. 이는 경우에 따라서 지나치게 가혹한 처분일 수 있다.

이 논문에서는 다음과 같은 내용을 포함하고 있다.

- 그룹 서명 기법에서 정상적인 사용자들에게서 발생할 수 있는 문제에 대한 예외 처리의 필요성의 고찰
- MMO06[2]에서 사용되는 Open/Judge가 이러한 예외 처리에 사용될 수 없는 이유의 고찰
- 이를 동일한 선형 암호문을 사용하는 BBS04[3]와 같이 잘 알려진 기존의 그룹 서명 기법에 적용하였을 때 발생할 수 있는 문제
- 이를 대체하기 위한 OpenHU과 JudgeHU에 대해서 제안하고, 이 제안이 기존의 MMO06의 Open/Judge에서와 같은 문제가 발생하지 않음을 랜덤 오라클을 이용하여 증명
- 응용 환경에 효율적으로 동작할 수 있는 Open/BatchProof/BatchJudge의 제안

이 논문의 나머지 구성은 다음과 같다. II장은 이 논문을 이해하기 위한 기초적인 내용을 다루고, III장에서는 선형 암호문을 사용하는 그룹 서명 기법에서 의도치 않은 오류를 위한 Open/Judge에 대해서 MMO06을 예를 들어 알아본다. IV장에서는 사용자 식별자만을 노출하는 새로운 Open과 Judge에 대해서 제안하고, 그 안전성에 대해서 살펴본다. V장은 이 논문의 결론을 담는다.

II. 일러두기

2.1 겹선형 접합

겹선형 접합(bilinear pairing) $e:\mathbb{G}_1\times\mathbb{G}_2\rightarrow\mathbb{G}_T$는 두 순환군(cyclic group) $\mathbb{G}_1$과 $\mathbb{G}_2$ 위의 각각 한 원소로부터 한 순환군 $\mathbb{G}_T$의 한 원소를 구하는 맵(map)으로 다음과 같은 성질을 갖는다.

- bilinearity: 모든 $g_1\in\mathbb{G}_1$, $g_2\in\mathbb{G}_2$와 $a,b\in\mathbb{Z}$에 대해서, $e(g_1^a,g_2^b)=e(g_1,g_2)^{ab}$이다.
- non-degeneracy: 만약 $\mathbb{G}_1=\langle g_1\rangle$, $\mathbb{G}_2=\langle g_2\rangle$이고, 모든 $x_1\in\mathbb{G}_1$, $x_2\in\mathbb{G}_2$에 대해서 $e(g_1,x_1)\neq 1_{\mathbb{G}_T}$, $e(x_1,g_2)\neq 1_{\mathbb{G}_T}$이다. (여기서 $1_{\mathbb{G}_T}$는 $\mathbb{G}_T$의 항등원)
- computable: 모든 $g_1\in\mathbb{G}_1$, $g_2\in\mathbb{G}_2$에 대해서, $e(g_1,g_2)$는 쉽게 계산될 수 있다.

2.2 그룹 서명 기법과 선형 암호화 (Linear Encryption)

D. Chaum이 제안한 그룹 서명(group signature) 기법은 서명자의 익명성을 보장하는 서명 기법 중에 하나이다[1]. 그룹 서명에는 그룹과 사용자에게 멤버십 인증서를 발급하고 관리하는 멤버십 관리자(issuer)와 익명취소 관리자(opener)가 존재하며 때로는 이 둘을 그룹 매니저(group manager, GM)로 가정하기도 한다[4,5]. 그룹 서명 기법에서 다음과 같은 구성을 갖는다.

- GKeyGen: 그룹 공개키, 멤버십 인증서 발행키, 익명 철폐키를 생성한다.
- Join/Iss: 멤버십 관리자는 특정한 그룹 구성원의 멤버십 인증서(또는 그룹 서명키)를 생성한다. Join은 사용자가 Iss는 멤버십 관리자가 실행한다. 이 둘은 서로 통신 프로토콜로써 연결되어 동작한다.
- GSign: 사용자는 멤버십 인증서(또는 그룹 서명키)를 이용하여 메시지의 그룹 서명을 생성한다.
- GVerify: 그룹 서명의 유효성을 확인한다.
- Open: 익명취소 관리자는 익명 철폐키를 이용하여 서명으로부터 사용자의 식별자와 그 계산 과정이 올바르다는 증거를 구한다.
- Judge: Open 결과의 유효성을 확인한다.

[2,3]과 같은 그룹 서명 기법은 서명을 생성할 때, 사용자의 식별자를 암호화하기 위하여 선형 암호화(linear encryption) 기법을 이용한다. 선형 암호화 기법은 DDH(Decisional Diffie-Hellman) 문제가 안전하지 않은 그룹에서 IND-CPA 안전하도록 설계된 암호화 기법으로, 오더(order)가 소수 p인 그룹 $\mathbb{G}$에 대해서 다음과 같이 동작한다.

- $(pk,sk)\leftarrow\mathrm{KeyGen}()$: $(\xi_1,\xi_2)\leftarrow_R\{\mathbb{Z}_p\}^2$, $\mathbb{G}=\langle h\rangle$, $u^{\xi_1}=v^{\xi_2}=h$, $pk\leftarrow(u,v,h)$, $sk\leftarrow(\xi_1,\xi_2)$
- $C\leftarrow\mathrm{Enc}(pk,M)$: $\alpha,\beta\leftarrow_R\{\mathbb{Z}_p\}^2$, $C\leftarrow(C_1=u^\alpha, C_2=v^\beta, C_3=Mh^{\alpha+\beta})$
- $M\leftarrow\mathrm{Dec}(pk,sk,C)$: $M\leftarrow C_3/(C_1^{\xi_1}\cdot C_2^{\xi_2})$

2.3 그룹 서명을 위한 보안 정의

그룹 서명은 공식적인 성질이 정립되기 전까지 다양한 그룹 서명에서 정의되어 사용된 비공식적인 보안 정의들과 BMW04 모델[4]이나 BSZ05 모델[5]에서 정립된 공식적인 보안 정의들을 가지고 있다. 이 논문의 원활한 이해를 위해서는 다음과 같은 비공식적인 정의 또는 공식적인 정의에 대해서 알아볼 필요가 있다.

- 익명성(anonymity)과 비연결성(unlinkability): 전통적으로 익명성은 GM을 제외한 누군가에게 그룹 서명이 주어졌을 때, 이 그룹 서명이 누가 생성했는지 알 수 없는 것을 의미한다. 비연결성은 GM을 제외한 누군가에게 두 그룹 서명을 주었을 때, 이 둘이 같은 사용자가 생성한 것인지 아닌지 알 수 없다는 것이다. 이 둘 모두는 BMW04 모델[4]의 '완전 익명성(full-anonymity)'과 BSZ05 모델[5]의 '익명성'에 포함된다. 완전 익명성의 경우 GM을 제외한 사용자의 그룹 서명키까지 주어진 환경에서의 익명성을 의미하며, BSZ05 모델의 익명성은 익명 철폐키를 제외한 멤버십 인증서 발급키 등을 포함한 나머지 모든 정보가 주어졌을 때의 익명성을 의미한다. MMO06[2]의 CPA-익명성(CPA-anonymity)은, BBS04[3]의 CPA-완전 익명성(CPA-full-anonymity)이 BMW04 모델[4]의 완전 익명성에서 그룹 서명키를 공격자에게 주지 않은 경우를 의미하듯, BSZ05 모델[5]의 익명성에서 Join/Iss, Open 등에 관련된 오라클들이 주어지지 않은 경우에 대한 것이다.
- 지역 연결성(local linkability)과 전역 연결성(global linkability): 강전일 등과 황정연 등이 그들의 논문[6,7]에서 이 두 개념을 사용하였으며, '지역 연결성'이란 '특정한 비밀키(연결키)를 갖는 지역적으로 제한된 어떠한 관리자(서비스 제공자)가 자신에게 주어진 그룹 서명들을 동일한 사용자가 서명한 그룹 서명들끼리 분류할 수 있는 성질'로 요약할 수 있다. 이 들은 익명 취소 관리자가 갖는 연결성을 전역 연결성이라고 하였다.
- 무죄입증성(exculpability)과 강한 무죄입증성(strong-exculabililty): 무죄입증성이란 '서명자 이외에 Open의 결과가 그 사용자인 그룹 서명을 생성할 수 없다'는 것으로 GM이 제외된 경우에 한한다. 이에 반해서 강한 무죄입증성은 'GM을 포함한' 무죄입증성이라고 볼 수 있다. BMW04 모델[4]의 '완전 추적 가능성(full-traceability)'과 BSZ05 모델[5]의 '추적 가능성(traceability)'은 강한 무죄입증성을 포함하고 있다. 일반적으로 GM이 알 수 없고 사용자만 알고 있는 어떤 비밀키가 그룹 서명의 생성에 사용되면 강한 무죄입증성을 가질 수 있어, 이 논문에서는 이 차이를 설명하기 위하여 사용한다.
- 비편제성(non-frameability): 여러 형태로 BSZ05 모델[5]에 다시 정의된 보안 성질로, 사용자가 정말로 생성한 그룹 서명이 아니라면 Judge를 통과할 수 있는 Open 결과(즉, 증명)를 생성할 수 없다는 것이다. 이 모델은 공격자에게 멤버십 인증서 발급키와 익명 철폐키를 주며, 공격자가 서명 오라클(signing oracle)에 질의하지 않은 메시지와 그룹 서명에 대해서 Open의 결과를 요구한다.

III. 선형 암호문을 사용하는 그룹 서명 기법에서 의도치 않은 오류의 처리

3.1 응용 환경: 서비스 제공의 증거로의 그룹 서명

서비스 제공자는 익명 사용자에게 서비스를 제공해주고, 서비스 제공자는 익명 사용자로부터 그룹 서명을 받는다. 서비스 제공자는 이러한 서비스 제공의 대가로 익명 사용자로부터 금전적 보상을 받아야만 한다. 그러나 서비스 제공자가 사용자의 실명과 관련하여 어떠한 정보도 없을 때, 서비스 제공의 대가를 사용자에게 직접 요구하는 것은 쉽지 않다. 당연하게도, 서비스 제공의 대가와 관련된 요구는 GM을 통할 수 밖에 없다. 서비스 제공자는 GM에게 그룹 서명을 전달하고, 서비스 제공의 대가를 사용자에게 요구해줄 것으로 요청해야 한다. GM은 서비스 제공자에게 받은 그룹 서명을 열어보고 사용자의 식별자를 알아낸 뒤, 이를 근거로 사용자에게 서비스 제공의 대가를 요구할 수 있다. 이 때, 사용자는 자신이 생성했던 그룹 서명도 저장해놓지 않는 이상 자신이 생성했는지 확인할 수 없으므로, GM은 사용자의 식별자가 정당한 절차를 거쳐 얻었음을 보이기 위한 증명을 첨부해야 한다. 증명이 올바른 경우, 사용자는 서비스 대가를

$\sigma \leftarrow \text{MMO06.Sign}(gpk, gsk=(A,B,x,s,q), m)$

$\alpha_1, \alpha_2, \beta, r_1, r_2, r_3, r_4, r_5, r_6 \leftarrow_R \{\mathbb{Z}_p\}^2$, $T_1 \leftarrow \psi(u)^{\alpha_1}$,
$T_2 \leftarrow \psi(v)^{\alpha_2}$, $T_3 \leftarrow A h^{\alpha_1+\beta_1}$, $D_1 \leftarrow (wh^x)^{\beta}$,
$D_2 \leftarrow (B\tilde{u}(\tilde{v})^s)^{\beta} D_1^{\alpha_1+\alpha_2}$, $R_1 \leftarrow h^{r_1} w^{r_2}$,
$R_2 \leftarrow h^{r_3}(\tilde{u})^{r_2}(\tilde{v})^{r_4} D_1^{r_5+r_6}$, $R_3 \leftarrow \psi(u)^{r_5}$, $R_4 \leftarrow \psi(v)^{r_6}$,
$c \leftarrow H(R_1, R_2, R_3, R_4, m)$, $s_1 \leftarrow r_1 + c\beta x$, $s_2 \leftarrow r_2 + c\beta$,
$s_3 \leftarrow r_3 + c\beta q$, $s_4 \leftarrow r_4 + c\beta s$, $s_5 \leftarrow r_5 + c\alpha_1$, $s_6 \leftarrow r_6 + c\alpha_1$,
$\sigma \leftarrow (T_1, T_2, T_3, D_1, D_2, c, s_1, s_2, s_3, s_4, s_5, s_6)$

$(A, \pi) \leftarrow \text{MMO06.Open}(gpk, ok=(\xi_1, \xi_2), m, \sigma)$

$A \leftarrow T_3/(T_1^{\xi_1} \cdot T_2^{\xi_2})$, $(r_1, r_2) \leftarrow_R \{\mathbb{Z}_p\}^2$,
$X_1 \leftarrow T_1^{\xi_1}, X_2 \leftarrow T_2^{\xi_2}, R_1 \leftarrow T_1^{r_1}, R_2 \leftarrow T_2^{r_2}, R_3 \leftarrow h^{r_1}, R_4 \leftarrow h^{r_2}$,
$c \leftarrow H(R_1, R_2, R_3, R_4)$,
$s_1 \leftarrow r_1 + c\xi_1$, $s_2 \leftarrow r_2 + c\xi_2$,
$\pi \leftarrow (pk_i, sig, B, x, s, X_1, X_2, c, s_1, s_2)$

$0/1 \leftarrow \text{MMO06.Verify}(gpk, m, \sigma)$

if $(e(T_3, D_1) \neq e(g_1, D_2))$ return 0
$\widetilde{R_1} \leftarrow h^{s_1} w^{s_2} / D_1^c$, $\widetilde{R_2} \leftarrow h^{s_3}(\tilde{u})^{s_2}(\tilde{v})^{s_4} D_1^{s_5+s_6} / D_2^c$,
$\widetilde{R_3} \leftarrow \psi(u)^{s_5} / T_1^c$, $\widetilde{R_4} \leftarrow \psi(v)^{s_6} / T_2^c$,
if $(c = H(R_1, R_2, R_3, R_4, m))$ return 1
else return 0

$0/1 \leftarrow \text{MMO06.Judge}(gpk, m, \sigma, A, \pi)$x

if $(\text{Verify}(pk_i, sig, B) \neq 1$ OR
$e(A, wh^x) \neq e(g_1, B\tilde{u}(\tilde{v})^s)$ OR $A \neq T_3/(X_1 \cdot X_2))$
return 0
$\widetilde{R_1} \leftarrow T_1^{s_1}/X_1^c$, $\widetilde{R_2} \leftarrow T_2^{s_2}/X_2^c$, $\widetilde{R_3} \leftarrow h^{s_1}/u^c$, $\widetilde{R_4} \leftarrow h^{s_2}/v^c$
if $(c = H(\widetilde{R_1}, \widetilde{R_2}, \widetilde{R_3}, \widetilde{R_4}))$ return 1
else return 0

〔그림 1〕 Makita-Manabe-Okamoto-06의 그룹 서명 기법[2]

GM을 통해 서비스 제공자에게 지불한다.

서비스 제공자는 당연히 자신이 사용자에게 받았던 그룹 서명을 근거로 자신이 받게 되는 요금을 예상할 수 있을 것이다. 하지만 서비스 제공자가 실제로 GM으로부터 받을 수 있는 요금은 그와 다를 수 있다. 사용자가 뜻하지 않은 오류로 인하여 그룹 서명을 중복하여 생성한다거나, 잘못된 메시지에 그룹 서명을 생성하여 전달하는 경우, 사용자는 자신이 서명하였다 하더라도, 자신에게 청구되는 요금에 대해서 동의하지 못할 수도 있다. 예를 들어, 현재에도 인터넷에 있어서 흔히 있는 프로그램 오류나 사용자의 이용 미숙에 따른 이중 결제 등이 여기에 속한다. 이 때, GM은 서명된 메시지와 서명자를 확인하여 사용자의 항의를 어느 정도 검증할 수 있다. 따라서 GM은 서비스 제공자에게 서비스 제공자가 실제로 얻게 되는 요금이 다르다는 사실을 설득해야 할 필요가 있다.

그룹 서명 기법은 위와 같은 경우에 대해서 그룹 서명의 익명성을 취소하는 방법으로 Open과 Judge를 제공하고 하고 있다. GM은 Open과 Judge 과정을 통하여 그룹 서명으로부터 사용자의 식별자가 올바르게 복호화 되었음을 증명할 수 있다. 그러나 일반적으로 Open 과정은 잘못된 행위의 사용자를 대상으로 하기 때문에 위와 같은 경우의 사용에 적합하지 않을 수 있다. 만약, Open의 결과 사용자의 멤버십 인증서 전부가 노출되거나, 일부가 노출되어 더 이상 익명성이 보장이 안 된다면, 해당 멤버십 인증서는 더 이상 사용할 수 없을 뿐만 아니라 반드시 사용자 퇴출 절차를 밟아서 사용하지 못하도록 해야 한다. 이는 정상적인 사용자에게 있어서 너무 가혹한 처분이며, 시스템의 입장에 있어서도 부담이 될 수밖에 없다.

〔6,7〕과 같이 지역 연결성을 갖도록 수정된 그룹 서명 기법의 경우, 위와 같은 경우에 서비스 제공자 스스로 문제를 해결할 수 있다. 즉, 지역 연결성을 갖는 그룹 서명의 경우 Open이나 Judge는 오로지 잘못된 행동을 하는 악의적인 사용자를 제지하기 위한 수단으로 사용될 수 있다. 그러나 지역 연결성이 없는 대부분의 수정되지 않은 그룹 서명의 경우 Open이나 Judge는 조금 더 신중하게 사용될 필요가 있다.

그룹 서명 기법이 가질 수 있는 이러한 문제에 대해서 MMO06의 예를 들어 생각해보기로 하자.

3.2. Makita-Manabe-Okamoto-06에서의 Open/Judge의 이해

3.2.1. 익명성이 취소되는 Open/Judge

MMO06[2]은 선형 암호화 기법을 사용하면서, Judge 기능을 제공해주는 그룹 서명 기법이다. 이 그룹 서명 기법에서는 그룹 공개키 $(\tilde{u}, \tilde{v}, h, u, v, w) \in \mathbb{G}_2$, 멤버십 인증서 (A, B, x, s, q), 멤버십 인증서 발행키 γ, 익명 철폐키 (ξ_1, ξ_2)는 $B = h^q$, $A = \psi(B\tilde{u}(\tilde{v})^s)^{1/(x+\gamma)}$, $w = h^{\gamma}$, $u^{\xi_1} = v^{\xi_2} = h$ 와 같은 관계에 이며, ψ는 $\mathbb{G}_2$로부터 $\mathbb{G}_1$으로의 동형 사상(isomorphism)이다. 또한 $g_1 = \psi(h)$이다. 최종적으로 서명은 $\sigma = (T_1, T_2, T_3, D_1, D_2, c, s_1, s_2, s_3, s_4, s_5, s_6)$ 와 같은 형태를 가지고 있다. 여기서 (T_1, T_2, T_3)는 사용자 식별자 A의 선형 암호문이

0/1←Attack.Trace.MMO06($gpk, ik=\gamma, (A,B,x,s)$, $\sigma=(T_1,T_2,T_3,D_1,D_2,\cdots)$)

/* $T_1=\psi(u)^{\alpha_1}, T_2=\psi(v)^{\alpha_2}, T_3=Ag^{\alpha_1+\alpha_2}$,
$D_1=(wh^x)^\beta, D_2=(B\tilde{u}(\tilde{v})^s)^\beta D_1^{\alpha_1+\alpha_2}$ */

if $e(g_1,D_2)=e(\psi(D_1^{1/(\gamma+x)}),B\tilde{u}(\tilde{v})^s)\cdot e(T_3/A,D_1)$
    return 1
else return 0

〔그림 2〕 MMO06에서의 멤버십 인증서 발행키를 가진 공격자에 의한 추적 공격

다. 해당 기법을 익명 철폐키의 형태[1]에 맞추어 수정한 Open과 Judge는 〔그림 1〕과 같은 과정으로 이루어진다. 여기서 sig는 B를 사용자의 개인키 sk_i로 서명한 것으로 사용자의 공개키 pk_i로 검증 가능하다.

Open과정에서 멤버십 인증서는 (A,B,x,s,q) 중 (A,B,x,s)가 노출된다. 멤버십 인증서에서 멤버십 관리자가 생성하였던 부분이 모두 노출되긴 하지만, q는 사용자 자신이 선택한 비밀로 노출되지 않기 때문에, 여전히 그룹 서명은 서명자만이 생성 가능하다. 즉, 강한 무죄증명성을 가지고 있는 MMO06은 앞서 지적했던 것과 같이 멤버십 인증서를 퇴출하지 않아도 문제가 발생하지 않을 것처럼 보인다. 그러나 이것은 어디까지나 서명자만 그룹 서명을 생성할 수 있다는 것에 대한 확신일 뿐이다. BSZ05 모델[5]을 따르면 익명성(anonymity)은 공격자에게 멤버십 인증서 발급키 γ를 주었을 때에도 확보되어야 한다. 이는 멤버십 관리자가 공격자가 될 수 있다는 것을 의미한다. MMO06에서는 익명성에 관하여 BSZ05 모델[5]을 따르지 않지만, 멤버십 인증서 발급키를 공격자에게 주고 있다. 하지만 멤버십 관리자처럼 사용자 데이터베이스에 접근할 수 있는 권한은 주어지지 않고 있다. 즉, MMO06에서 가정하는 공격자는 멤버십 관리자가 아니라 우연히 멤버십 인증서 발급키를 손에 넣은 공격자이다. 이러한 공격자에게 익명 취소 관리자에 의해 어떤 사용자의 멤버십 인증서의 일부 (A,B,x,s)가 노출되었다면, 멤버십 인증서 발급키를 가진 공격자는 〔그림 2〕와 같이 해당 사용자가 생성한 모든 그룹 서명을 추적할 수 있다.

〔그림 2〕에서 공격자는

$$
\begin{aligned}
&e(g_1, D_2)\\
&\quad = e(g_1,(B\tilde{u}(\tilde{v})^s)^\beta D_1^{\alpha_1+\alpha_2})\\
&\quad = e(g_1,B\tilde{u}(\tilde{v})^s)^\beta e(g_1,D_1)^{\alpha_1+\alpha_2}\\
&\quad = e(g_1,B\tilde{u}(\tilde{v})^s)^\beta e(g_1,D_1)^{\alpha_1+\alpha_2}\\
&\quad = e((g_1)^\beta,B\tilde{u}(\tilde{v})^s)e((g_1)^{\alpha_1+\alpha_2},D_1)\\
&\quad = e(\psi(D_1^{1/(\gamma+x)}),B\tilde{u}(\tilde{v})^s)e(T_3/A,D_1)
\end{aligned}
\qquad (1)
$$

이기 때문에 $e(g_1,D_2)$와 $e(\psi(D_1^{1/(\gamma+x)}),B\tilde{u}(\tilde{v})^s)$ $e(T_1/A,D_1)$를 각각 구하여 비교함으로써, 주어진 서명의 서명자의 멤버십 인증서의 일부가 (A,B,x,s)임을 확인할 수 있다.

여기서 $\psi(D_1^{1/(\gamma+x)})=\psi(((wh^x)^\beta)^{1/(\gamma+x)})=$ $\psi((h^\gamma h^x)^{\beta/(\gamma+x)})=\psi(h^\beta)=\psi(h)^\beta=(g_1)^\beta$이다. 따라서 MMO06에서 Open에 의해 (A,B,x,s)가 노출될 경우 해당 서명자의 모든 그룹 서명들은 멤버십 관리자에게 있어서 더 이상의 익명성을 갖지 않는다고 할 수 있다. 즉, MMO06에서는 어떠한 그룹 서명이 한 번 Open되면 해당 그룹 서명에 사용된 멤버십 인증서는 반드시 퇴출되어야 한다.

3.2.2. MMO06의 Open/Judge의 전략

MMO06에서는 Open의 사용에 따라 발생하는 위와 같은 일에 대해서는 특별히 고려되고 있지 않다. 대부분의 그룹 서명에서 Open은 정직하지 않은 사용자에 대한 것으로 널리 받아들이고 있기 때문에 MMO06에 특별한 문제점이 있는 것은 아니다. 그러나 이 논문에서 고려하는 Open이후에서 사용 가능한 멤버십 인증서를 위해서는 MMO06의 Open과 Judge는 다소 수정되어야할 필요가 있을 것으로 보인다. 이를 위하여 MMO06에서 어떠한 Open/Judge의 전략을 사용하는지 이해해야할 것이다.

익명취소 관리자는 Open 과정에서 익명 철폐키 (ξ_1,ξ_2)에 대해서 NIZK(Non-Interactive Zero-Knowledge proof)을 수행한다. $\{R_3=g_2^{r_1}, R_4=g_2^{r_2}, c=H(\cdots,R_3,R_4), s_1=r_1+c\xi_1, s_2=r_2+c\xi_2\}$이 그에 해당한다. 이는 안전성이 충분히 검증되어 널리 사용되는 NIZK 방식을 익명 철폐키 ξ_1와 ξ_2에 대해서 각각 독립적으로 수행한 것으로, (ξ_1,ξ_2) 없이는 생성해낼 수 없다.

다음으로, 익명취소 관리자는 (ξ_1,ξ_2)를 이용하여 $X_1=T_1^{\xi_1}$과 $X_2=T_2^{\xi_2}$처럼 계산하였음을 증명하려고 한다. 하지만 저자들은 X_1과 X_2는 익명취소 관리자가

1) MMO06의 익명 철폐키는 $(1/\xi_1,1/\xi_2)$로 사용하나, 이 논문에서는 (ξ_1,ξ_2)의 형태로 사용하였다.

임의로 생성하는 것으로 (ξ_1,ξ_2)를 이용하지 않고 전혀 다른 값으로 생성했을 가능성이 있다고 생각하였다. 익명 철폐키 ξ_1와 ξ_2의 NIZK이 조작하기 힘든 이유 중 하나는 Judge때 $\widetilde{R_3}$와 $\widetilde{R_4}$의 생성에 사용되는 u와 v가 사전에 공개되어 신뢰할 수 있기 때문인데, $\widetilde{R_1}$와 $\widetilde{R_2}$ 의 생성에 사용되는 X_1과 X_2의 경우에는 증명하는 시점에서야 결정되고 증명과 함께 배포된다. s_1과 s_2를 (ξ_1,ξ_2)의 NIZK와 공유하고 있고 c를 예측하지 못하기 때문에, (X_1,R_1)과 (X_2,R_2)를 조작하기 힘들 것처럼 보이지만, R_1과 R_2를 미리 무작위로 선택하면 X_1과 X_2를 계산하는 시점에서 c는 고정된 값이다. 따라서 $X_1 \leftarrow (T_1^{s_1}/R_1)^{1/c}$과 $X_2 \leftarrow (T_2^{s_2}/R_2)^{1/c}$로 계산하면 $R_1 = T_1^{s_1}/X_1^c$과 $R_2 = T_2^{s_2}/X_2^c$인 관계가 성립하도록 할 수 있다. 이러한 X_1과 X_2에 대해서 $A \neq T_3/(X_1 \cdot X_2)$임은 명백하다.

이러한 조작을 하지 못하도록 하기 위해서, 저자들은 X_1과 X_2이 유효한 값인지 확인하는 과정이 추가적으로 필요했다. 만약 X_1과 X_2가 유효하지 않다는 것은 $A' = T_3/(X_1 \cdot X_2)$ 값이 실제로는 A와 다르다는 의미가 된다. A는 $\psi(B\tilde{u}(\tilde{v})^s)^{1/(x+\gamma)}$와로 이루어져 있으므로, (B,x,s)와 γ를 알려줄 수 있다면 $T_3/(X_1 \cdot X_2) = \psi(B\tilde{u}(\tilde{v})^s)^{1/(x+\gamma)}$임을 확인하도록 할 수 있고, 결과적으로, X_1과 X_2가 (ξ_1,ξ_2)로 만들어진 유효한 값임을 확신할 수 있다. 하지만 멤버십 인증서 발행키 γ를 공개할 수 없으므로, $e(T_3/(X_1 \cdot X_2), wh^x) = e(g_1, B\tilde{u}(\tilde{v})^s)$와 같이 그룹 공개키 중 하나인 w를 이용하여 검증하도록 해야 한다.

여기에 증명을 고의적으로 왜곡하려는 공격자(멤버십 인증서 발행키 γ를 알고 있다)에 대한 고려로써[5], 서명자의 외부의 서명용 공개키 pk_i와 B의 서명 sig를 추가적으로 공개하여 (B,x,s)가 공격자에 의해서 증명하는 시점에 생성되지 않았음을 보였다. $B = A'^{\gamma+x}/(uv^s)$로, 공격자가 무작위로 선택한 R_1과 R_2 때문에 생성되는 A'로부터 계산해낼 수 있다. 이는 NIZK와는 관계가 없고 오로지 그룹 서명 기법의 비편제성(non-frameability)의 확보 때문에 필요한 것이다.

3.2.3. BBS04로의 Open과 Judge의 이식

BBS04는 MMO06과 다르게 서명의 길이가 짧을 뿐만 아니라, 대부분의 연산을 $\mathbb{G}_1$에서 수행하기 때문에 MMO06보다 고속으로 동작한다. 따라서 그룹 서명의 응용 환경에 있어서 BBS04의 사용은 고려될 만 하다. 하지만 앞서 이 논문에서 고려하였던 응용 환경에서는 BBS04는 사용될 수 없을 것처럼 보인다. 왜냐하면 BBS04 자체로는 Open과 Judge를 제공하고 있지 않기 때문이다. 그러나 MMO06과 BBS04는 동일한 선형 암호문을 사용하고 있는 그룹 서명들이기 때문에, MMO06의 Open과 Judge를 BBS04에 맞도록 수정한 다음 어렵지 않게 사용할 수 있다. 원래의 BBS04에는 Judge기능이 없기 때문에 비편제성을 위한 고려는 필요가 없다.

이렇게 BBS04에 Open/Judge 기능이 생겼다면, 이 논문에서 고려하였던 응용 환경에 사용할 수 있는지 확인해볼 필요가 있다. MMO06과 다르게 BBS04는 서명 내에 D_1과 D_2와 같은 요소가 없기 때문에, 앞서 MMO06에서의 Attack.Trace. MMO06과 동일한 공격은 수행할 수 없다. 그러나 BBS04는 MMO06의 Open을 이용할 경우, 강한 무죄입증성이 없기 때문에 멤버십 인증서 (A,x)가 모두 노출된다. 멤버십 인증서가 노출되면, 다른 누군가는 이를 이용하여 유효한 서명을 생성할 수 있기 때문에, 그 즉시 멤버십 인증서를 폐기해야 한다. 이는 사실상 사용자 퇴출과 다르지 않다.

따라서 MMO06의 Open/Judge를 BBS04에 사용해도, 이 논문에서 원하는 응용 환경에서의 사용은 여전히 불가능하다. 따라서 BBS04를 위해 다른 Open과 Judge를 설계하든가, 아니면 BBS04 자체를 해당 응용 환경에서의 사용을 제한하여야 한다.

IV. 사용자 식별자만을 노출하는 새로운 Open과 Judge

4.1. 정상적인 사용자를 위한 Open과 Judge

이상과 같은 이유에서 MMO06의 Open과 Judge는 자신뿐만 아니라, BBS04에서도 정상적인 사용자의 뜻하지 않은 오류에는 사용될 수 없을 것 같다. 일반적으로 그룹 서명 기법에서 사용자의 식별자를 알아낸다는 의미는 사용자의 실명을 알아낸 것과 동일시되고 있다. 이 또한 정확한 의미에서는 특정 그룹 서명에 대한 식별자만을 알아냈다는 것으로 Open하지 않은 그룹 서명에 대하여 실명이 노출된 것은 아니다. 필요에 따라 Open을 할 수 있고, 특정 그룹 서명의 사용자 식별자가 노출될 수는 있지만, 한 번의

$(A,\pi)\leftarrow$OpenHU
$(gpk,ok=(\xi_1,\xi_2),m,\sigma=(T_1,T_2,T_3,\cdots))$

$A\leftarrow T_3/(T_1^{\xi_1}\cdot T_2^{\xi_2})$, $(r_1,r_2)\leftarrow_R\{\mathbb{Z}_p\}^2$,
$X_1\leftarrow T_1^{\xi_1}, X_2\leftarrow T_2^{\xi_2}, R_1\leftarrow T_1^{r_1}, R_2\leftarrow T_2^{r_2}, R_3\leftarrow h^{r_1}, R_4\leftarrow h^{r_2}$,
$c\leftarrow H(X_1,X_2,R_1,R_2,R_3,R_4)$, $s_1\leftarrow r_1+c\xi_1$, $s_2\leftarrow r_2+c\xi_2$,
$\pi\leftarrow(X_1,X_2,c,s_1,s_2)$

$0/1\leftarrow$JudgeHU(gpk,m,σ,A,π)
if $(X_1=1$ OR $X_2=1$ OR $A\neq T_3/(X_1\cdot X_2))$
return 0
$\widetilde{R}_1\leftarrow T_1^{s_1}/X_1^c$, $\widetilde{R}_2\leftarrow T_2^{s_2}/X_2^c$, $\widetilde{R}_3\leftarrow h^{s_1}/u^c$, $\widetilde{R}_4\leftarrow h^{s_2}/v^c$,
if $(c=H(X_1,X_2,\widetilde{R}_1,\widetilde{R}_2,\widetilde{R}_3,\widetilde{R}_4))$ return 1,
else return 0

〔그림 3〕 정직한 사용자를 위한 Open과 Judge

Open이 사용자의 그룹 퇴출로 이어지는 것은 가혹하다. 정상적인 사용자의 뜻하지 않은 오류에 대해서 그룹 서명 기법이 가진 예외처리 방법은 Open밖에 없으므로 반드시 이를 필요로 하지만, 사용자 퇴출로 이어지는 Open이라면 사실상 사용할 수 없다. 따라서 MMO06의 Open과 Judge를 대신하여 사용할 수 있는 새로운 OpenHU 및 JudgeHU가 필요하다.

MMO06의 Open에서 c는 $H(R_1,R_2,R_3,R_4)$처럼 계산하였기 때문에, X_1과 X_2의 증명 시점에서의 유효성에 문제가 발생하였고, 결국 사용자 식별자 A의 유효성까지 영향을 주어 A가 올바른 값인지 확인하는 과정이 추가적으로 필요했다. X_1과 X_2의 유효성만 확보할 수 있다면, 문제를 쉽게 가져갈 수 있을 것이다.

X_1과 X_2가 R_1과 R_2를 보고 생성된 것이 아님을 보이기 위해서는 간단히 c를 $H(X_1,X_2,R_1,R_2,R_3,R_4)$로 계산하도록 하면 된다. 앞서 설명했던 것과 같이 s_1과 s_2, 그리고 R_3와 R_4는 익명 철폐키 (ξ_1,ξ_2)의 NIZK와 맞물려 원천적으로 조작이 불가능하다. 따라서 X_1과 X_2를 조작하려면, $R_1=T_1^{s_1}/X_1^c$, $R_2=T_2^{s_2}/X_2^c$인 관계가 되도록 R_1과 R_2를 선택할 수 있어야 한다. 그런데 c는 (X_1,R_1)와 (X_2,R_2)가 모두 정해지는 시점에서 결정되고 주어진 서명의 T_1과 T_2는 조작할 수 없으므로, 이는 불가능하다. c와 관계없는 (X_1,R_1)와 (X_2,R_2)를 만드는 유일한 경우는 오로지 $X_1=X_2=1$, $R_1=T_1^{s_1}$, $R_2=T_2^{s_2}$인 경우인데, 정상적인 서명에 대해서, $T_1^{\xi_1}=1$ 또는 $T_2^{\xi_2}=1$이 될 확률은 매우 낮을 뿐만 아니라,[2] 이는 Judge 과정에서도 쉽게 검출이 가능하다. 따라서 MMO06과 BBS04 둘 모두에 대해서 사용자의 식별자만을 노출하는 〔그림 3〕과 같은 OpenHU와 JudgeHU를 생각해볼 수 있다.

OpenHU와 JudgeHU는 정상적인 사용자의 실명 정보를 노출하고자 하는 것이 아니므로, MMO06의 Open처럼 사용자의 실명확인을 위한 과정이 없다. MMO06이 Open/Judge 대신 OpenHU/JudgeHU를 사용한다고 해도 비편제성은 여전히 확보가능하다. 왜냐하면 MMO06에서, 강한 무죄증명성 때문에 어떠한 공격자도 멤버십 인증서를 알 수 없는 정상적인 사용자에 대한 서명은 위조할 수 없으며, 정직한 사용자가 주어진 멤버십 인증서에 다른 A'를 넣어 그룹 서명을 생성할 수도 없다. 앞서 설명한 바와 같은 이유에서 X_1와 X_2는 $T_1^{\xi_1}$와 $T_2^{\xi_2}$이외에 다른 값일 가능성은 없으므로, OpenHU의 결과 A는 언제나 주어진 서명에 암호화된 멤버십 인증서의 식별자이다. 즉, 그룹 서명이 현재 올바르다면 A는 언제나 정직한 사용자의 멤버십 인증서 식별자이다. 우리가 더 확인해 보아야 할 것은 OpenHU와 JudgeHU를 사용할 때, Attack .Trace.MMO06과 같은 공격이 여전히 가능한 가일 것이다.

정리 1. 선형 암호화 기법이 (t',ε')-CPA 의미론적(semantically)으로 안전할 때, OpenHU/ JudgeHU를 Open/Judge로 사용하는 선형 암호화 기법 기반의 그룹 서명에 대해서, 어떠한 메시지 m_1과 그에 대한 선형 암호문을 사용하는 그룹 서명 σ_1이 있고, 그에 대한 OpenHU의 결과로써 (A,π)가 주어졌을 때, 어떠한 확률 기반의 다항식 알고리즘 $\mathscr{A}$가 또 다른 메시지 m_2과 그룹 서명 σ_2에 대해서 σ_1과 σ_2가 동일한 사용자에 의해서 서명되었다는 것을 $t\geq t'-O(q_H)$시간 안에 취할 수 있는 이득은 $\varepsilon\leq\varepsilon'$와 같다. 알고리즘 $\mathscr{A}$의 이득은 $\varepsilon=|\Pr[\mathscr{A}(gpk,(m_1,\sigma_1),(A_0,\pi),(m_2,\sigma_2)_{A_0})=1]-\Pr[\mathscr{A}(gpk,(m_1,\sigma_1),(A_0,\pi),(m_2,\sigma_2)_{A_1})=1]|$로 계산하며, q_H는 랜덤 오라클 O^H로의 질의 횟수이다.

증명. 어떠한 알고리즘 $\mathscr{A}$가 t시간 안에 ε만큼의

2) 이러한 T_1과 T_2를 고의적으로 생성할 수 있는 주체는 (ξ_1,ξ_2)를 알고 있는 익명 취소 관리자밖에 없는데, 현재 이 논의는 (ξ_1,ξ_2)를 알고 있는 공격자를 막고자 하는 것이므로, 이러한 고려는 일견 이유가 없다고 할 수 있지만, 어떠한 외적인 요인으로 인하여 정상적인 사용자가 그러한 T_1과 T_2를 선택하게 될 경우를 위하여 이를 확인하는 절차를 고려한다.

이득으로 주어진 문제를 해결하는 한다고 할 때, 이를 이용하여 선형 암호화 기법의 IND-CPA를 깨는 또 다른 알고리즘 $\mathscr{B}$를 설계할 수 있다.

알고리즘 $\mathscr{B}$는 어떠한 다른 알고리즘 $\mathscr{C}$로부터 선형 암호화 기법에 대한 공개키 $(u,v,h)\in\{\mathbb{G}_2\}^3$를 받는다. 알고리즘 $\mathscr{B}$는 멤버십 인증서 발행키 $\gamma\in\mathbb{Z}_p$를 무작위로 선택하고, 이에 따라 (u,v,h)를 포함하는 gpk를 생성한다. 예를 들어, MMO06에서는 $gpk=(\tilde{u},\tilde{v},h,u,v,w)$를 생성하기 위해서는 $w=h^{\gamma}$를 계산하고, 나머지 $(\tilde{u},\tilde{v})\in_R\{\mathbb{G}_2\}^2$를 무작위로 선택하면 된다. 알고리즘 $\mathscr{B}$는 임의의 멤버십 인증서 $gsk=(A,\cdots)$를 생성하고(이는 멤버십 인증서 발행키가 있기 때문에 가능하다), gpk와 gsk를 이용하여 임의로 선택한 메시지 m_1에 대하여 서명 $\sigma_1=(T_1,T_2,T_3,\cdots)$을 생성한다. 이 때, $A=\psi(A_0)$인 관계에 있다. 알고리즘 $\mathscr{B}$는 무작위로 $(c,s_1,s_2)\in_R\{\mathbb{Z}_p\}^3$와 $X_1\in_R\mathbb{G}_1$를 생성하고, $X_2=T_3/(A\cdot X_1)$, $R_1=T_1^{s_1}/X_1^c$, $R_2\leftarrow T_2^{s_2}/X_2^c$, $R_3\leftarrow h^{s_1}/u^c$, $R_4\leftarrow h^{s_2}/v^c$로 계산하고, $\pi=(X_1,X_2,c,s_1,s_2)$로 놓는다. 그 후 알고리즘 $\mathscr{B}$는 랜덤 오라클 O^H에 $c=H(X_1,X_2,R_1,R_2,R_3,R_4)$가 되도록 프로그램 한다. 알고리즘 $\mathscr{B}$는 알고리즘 $\mathscr{C}$에게 A_0와 무작위로 선택한 $A_1\in_R\mathbb{G}_2$를 보낸다.

알고리즘 $\mathscr{C}$는 A_0와 A_1중에 하나를 무작위로 선택하여 A_b라하고, 이를 (u,v,h)를 이용하여 선형 암호화한 암호문 $C_b=(c_1=u^{\alpha},c_2=v^{\beta},c_3=A_bh^{\alpha+\beta})$를 알고리즘 $\mathscr{B}$에게 전달한다. 알고리즘 $\mathscr{B}$는 이를 이용하여 무작위로 선택한 메시지 m_2에 대한 그룹 서명 σ_2를 생성해야 한다. 이때 σ_2는 C_b를 이용하여 생성해야 하는데, 알고리즘 $\mathscr{B}$는 C_b에 사용된 α와 β를 알 수 없으므로, 정상적인 서명을 생성할 수 없다. 알고리즘 $\mathscr{B}$는 서명의 나머지 요소들을 랜덤하게 채우고 랜덤 오라클 O^H을 조작하여 해당 서명이 올바르도록 한다. 이는 위와 유사한 과정을 통해 이루어질 수 있다. 단, MMO06의 경우 $e(\psi(c_3),D_{1|\sigma_2})=e(g_1,D_{2|\sigma_2})$인 관계가 되도록 $D_{1|\sigma_2}$과 $D_{2|\sigma_2}$를 조작해야 하는데, 무작위로 선택한 $\delta\leftarrow_R\mathbb{Z}_p$에 대해서 $D_{1|\sigma_2}=h^{\delta}$, $D_{2|\sigma_2}=c_3^{\delta}$로 놓으면 된다.[3] 이제 알고리즘 $\mathscr{B}$는 $(gpk,(m_1,\sigma_1),(A_0,\pi),(m_2,\sigma_2))$를 알고리즘 $\mathscr{A}$에 입력하고 알고리즘 $\mathscr{A}$를 동작시킨다.

랜덤 오라클 O^H는 알고리즘 $\mathscr{B}$가 조작한 두 개의 질의에 대해서만 저장된 값을 응답으로써 돌려주고, 나머지 질의에 대해서는 무작위 값을 선택하여 돌려주되, 이를 기록하여 동일한 질의가 두 번 이상 도착했을 때에는 같은 응답을 주도록 한다. 알고리즘 $\mathscr{A}$는 자신이 원하는 언제든 랜덤 오라클 O^H에 질의를 할 수 있다.

알고리즘 $\mathscr{A}$는 t시간 안에 $(1/2+\varepsilon)$의 확률로 두 서명이 동일한 사용자에 의해 생성되었는지에 대해서 0 또는 1로 응답할 것이다. 만약 알고리즘 $\mathscr{A}$가 다른 사용자에 의해서 서명되었다는 의미로 0을 돌려주었다면 알고리즘 $\mathscr{B}$는 자신이 받은 C_b가 A_1의 암호문이라는 의미로 1을 알고리즘 $\mathscr{C}$에게 돌려준다. 만약 반대로 알고리즘 $\mathscr{A}$가 1을 돌려주었다면 알고리즘 $\mathscr{B}$는 C_b가 A_0의 암호문이라는 의미로 0을 알고리즘 $\mathscr{C}$에게 돌려준다.

이 과정에서 알고리즘 $\mathscr{A}$는 (A_0,π)와 (m_1,σ_1)과 (m_2,σ_2)를 검증하는 과정이 필요하다. 이 중 (m_1,σ_1)는 정상적으로 생성하였으므로 항상 유효하지만, 나머지 (A_0,π)와 (m_2,σ_2)는 그렇지 않다. 알고리즘 $\mathscr{A}$가 랜덤 오라클 없이 둘 중 하나라도 π와 σ_2에 포함된 해시 함수를 계산했을 수도 있다면, 이 둘이 잘못된 것을 알고 알고리즘 $\mathscr{A}$는 중간에 연산을 중단할 것이다. 그러나 이 확률은 $1/p$보다 작으므로 무시할 만하다. 또한 이 과정에서 알고리즘 $\mathscr{A}$는 q_H만큼의 질의를 랜덤 오라클 O^H에 보냈을 것이고, 이 시간은 $O(q_H)$로 표현 가능하다. 따라서 만약 선형 암호화 기법이 (t',ε')-CPA 안전하다면, $\varepsilon'\geq\varepsilon$와 같으며, $t'\leq t+O(q_H)$와 같다. □

위 증명은 MMO06에만 제한된 것이 아니며, 선형 암호문을 사용하는 대부분의 그룹 서명에 유효하므로, BBS04에도 적용 가능하다. 따라서 OpenHU와 JudgeHU는 멤버십 인증서 발행키를 손에 넣은 공격자에게 Attack.Trace와 같은 공격에 안전하다.

4.2. 동일 식별자를 갖는 그룹 서명들을 위한 일괄 Open/Judge

3.1절에서의 시나리오에 따르면 GM은 같은 사용자 식별자를 갖는 복수의 서명을 동시에 Open해야 할 필요도 있다. n개의 선형 암호문을 사용하는 그룹 서명에 대해서 Open과 Judge를 수행하는 가장 단순

3) $h=\langle\mathbb{G}_2\rangle$이므로 어떤 알 수 없는 η에 대해서 $A_b=h^{\eta}$이다. 이 때, $e(\psi(A_bh^{\alpha+\beta}),h^{\delta})=e(g_1,h)^{\delta(\eta+\alpha+\beta)}$
$=e(g_1,(h^{\eta+\alpha+\beta})^{\delta})=e(g_1,(A_bh^{\alpha+\beta})^{\delta})$이다.

$A \leftarrow \text{Open}(gpk, ok=(\xi_1,\xi_2), m, \sigma)$	$0/1 \leftarrow \text{BatchJudge}(gpk, \{m_1,\cdots,m_n\}, \{\sigma_1,\cdots,\sigma_n\}, A, \pi)$
$A \leftarrow T_{3,i}/(T_{1,i}^{\xi_1} \cdot T_{2,i}^{\xi_2})$	if $(X_1=1$ OR $X_2=1$ OR $X_3=1$ OR $X_4=1)$ return 0
$\pi \leftarrow \text{BatchProof}(gpk, ok=(\xi_1,\xi_2), \{m_1,\cdots,m_n\}, \{\sigma_1,\cdots,\sigma_n\})$	$(b_1,b_2,\cdots,b_n,t) \leftarrow \text{PRNG}(\sigma_1,\sigma_2,\cdots,\sigma_n),\ \ell=\sum_{i=1}^{n} b_n,$
$(b_1,b_2,\cdots,b_n,t) \leftarrow \text{PRNG}(\sigma_1,\sigma_2,\cdots,\sigma_n),$	$E_1=\prod_{i=1}^{n} T_{1,i}^{b_i},\ E_2=\prod_{i=1}^{n} T_{2,i}^{b_i},\ E_3 \leftarrow \prod_{i=1}^{n} T_{3,i}^{b_i},$
$(r_1,r_2) \leftarrow_R \{\mathbb{Z}_p\}^2, E_1=\prod_{i=1}^{n} T_{1,i}^{b_i},\ E_2=\prod_{i=1}^{n} T_{2,i}^{b_i},$	if $(A \neq T_{3,t}/(X_3X_4)$ OR $A^{\ell} \neq E_3/(X_1X_2))$ return 0
$X_1 \leftarrow E_1^{\xi_1},\ X_2 \leftarrow E_2^{\xi_2},\ X_3 \leftarrow T_{1,t}^{\xi_1},\ X_4 \leftarrow T_{2,t}^{\xi_2},$	$\widetilde{R_1} \leftarrow E_1^{s_1}/X_1^c,\ \widetilde{R_2} \leftarrow E_2^{s_2}/X_2^c,\ \widetilde{R_3} \leftarrow T_{1,t}^{s_1}/X_3^c,$
$R_1 \leftarrow E_1^{r_1},\ R_2 \leftarrow E_2^{r_2},\ R_3 \leftarrow T_{1,t}^{r_1},\ R_4 \leftarrow T_{2,t}^{r_2},\ R_5 \leftarrow h^{r_1},\ R_6 \leftarrow h^{r_2},$	$\widetilde{R_4} \leftarrow T_{2,t}^{s_2}/X_4^c,\ \widetilde{R_5} \leftarrow h^{s_1}/u^c,\ \widetilde{R_6} \leftarrow h^{s_2}/v^c,$
$c \leftarrow H(E_1,E_2,X_1,X_2,X_3,X_4,R_1,R_2,R_3,R_4,R_5,R_6),$	if $(c=H(E_1,E_2,X_1,X_2,X_3,X_4,\widetilde{R_1},\widetilde{R_2},\widetilde{R_3},\widetilde{R_4},\widetilde{R_5},\widetilde{R_6}))$ return 1 else return 0
$s_1 \leftarrow r_1+c\xi_1,\ s_2 \leftarrow r_2+c\xi_2,\ \pi \leftarrow (X_1,X_2,X_3,X_4,c,s_1,s_2)$	

〔그림 4〕 정직한 사용자의 예외 처리를 위한 Open/BatchProof/BatchJudge

한 방법은 GM과 서비스 제공자가 4.1절과 같은 과정을 n번 반복하는 것이다. ξ_1과 ξ_2의 증명과 확인 값 c를 공유함으로써 약간의 연산과 통신비용을 줄일 수 있지만, 큰 차이는 없다. 이러한 환경에서 다음과 같은 연산과 공간에 있어서 충분한 이득을 주는 Open과 Judge 구조를 생각해볼 수 있다.

GM은 위 Open을 수행해서 같은 사용자 식별자를 가진 서명을 골라낸 뒤, Judge를 위한 증명 π를 BatchProof를 통해서 생성해낸다. 이러한 과정을 거쳐 선택된 그룹 서명들 $\{\sigma_1,\sigma_2,\cdots,\sigma_n\}$과 이들의 식별자 A, 그리고 증명 π를 서비스 제공자에게 전송해주면, 서비스 제공자는 BatchJudge를 통해서 서명들의 식별자가 A가 올바른지 확인한다.

BatchProof의 동작 방식을 간략히 설명하면, 다음과 같다. 모두 같은 식별자를 가진 서명에 대해서

$$\begin{aligned} A^n &= \prod_{i=1}^{n}\left(T_{3,i}/(T_{1,i}^{\xi_1}T_{2,i}^{\xi_2})\right) \\ &= \left(\prod_{i=1}^{n} T_{3,i}\right)/\left(\prod_{i=1}^{n} T_{1,i}^{\xi_1}\prod_{i=1}^{n} T_{2,i}^{\xi_2}\right) \\ &= Y_3/(Y_1^{\xi_1}Y_2^{\xi_2}) \end{aligned} \quad (2)$$

인 관계를 유지한다. (Y_1,Y_2,Y_3)는 서명이 공개되어 있어 이를 조작할 수 없기 때문에, $A^n=Y_1/(Y_2^{\xi_1}Y_3^{\xi_2})$를 증명하는 것은 4.1절과 같은 방식으로 증명될 수 있다. 그러나 이는 정확한 의미에서 그룹 서명들 $\{\sigma_1,\sigma_2,\cdots,\sigma_n\}$의 식별자의 평균이 A와 같다는 것을 증명한 것으로 GM은 동일하지 않은 식별자를 가진 서명들을 특정한 사용자의 식별자인 것처럼 위조하거나, 무작위로 선택한 서명들의 평균을 구해 A로 제시할 수 있다. 이를 막기 위해서, 각각의 서명의 $(T_{1,i},T_{2,i},T_{3,i})$에 무작위 상수 b_i를 곱하여 (E_1,E_2,E_3)를 만들어 사용하고, 무작위로 t번째 그룹 서명을 선택하여 4.1절과 같은 증명을 수행한다. 그러나 무작위로 선택되는 $\{b_i\}_{i=1..n}$와 t에 따라 GM은 여전히 수식에 맞는 그룹 서명을 선택할 수 있으므로, 이를 방지하기 위해서 PRNG의 입력으로 그룹 서명 $\{\sigma_1,\sigma_2,\cdots,\sigma_n\}$을 사용하여 $\{b_i\}_{i=1..n}$와 t를 구하도록 한다. $\{b_i\}_{i=1..n}$와 t는 어떠한 그룹 서명이 선택하기 전에는 알 수 없고, 동일하지 않은 식별자를 가진 그룹 서명들이 $E_1/(E_2^{\xi_1}E_3^{\xi_2})$가 $(T_{3,t}/(X_3X_4))^{\ell}$와 동일한 값이 되도록 하기 어렵다.

이러한 Open/BatchProof/BatchJudge도 〔정리 1〕과 그 증명과 동일하게 멤버십 인증서 발행키를 가진 공격자에게 추적되지 않음을 보일 수 있다. 이는 단지 (A,π)를 바꾸는 것으로, π의 유효성을 믿게 하기 위하여 프로그램 가능한 랜덤 오라클을 이용할 수 있다.

4.3. 강한 무죄입증성의 필요성

앞서 살펴보았던 것처럼 MMO06의 경우 강한 무죄입증성은 멤버십 인증서 발급키를 가진 공격자에게 별다른 의미가 없었지만, BBS04에서와 같은 예를 살펴보았을 때, 강한 무죄입증성은 중요한 역할을 한다. 이 논문에서는 MMO06과 마찬가지로 멤버십 인증서 발급키만을 가진 공격자를 우선 가정하지만, 멤버십 관리자의 경우 과거 자신이 발행했던 정보를 모두 가지고 있을 수 있다. 만약 멤버십 관리자가 공격자가

된다면, 이 논문에서 제안하는 것과 같은 OpenHU/JudgeHU를 사용한다고 해도, 식별자 A로부터 멤버십 인증서 중 사용자만 알고 있는 비밀키를 제외한 모든 멤버십 인증서가 노출된다. 강한 무죄입증성이 없을 때, 이는 곧 멤버십 인증서의 노출을 의미한다.

이러한 고민은 결국 다시 멤버십 관리자를 신뢰할 수 있는가 없는가의 문제로 되돌아가, 강한 무죄입증성이 없는 경우 싫어도 멤버십 관리자를 신뢰해야만 한다. 이는 보안을 바라보는 보수적 입장에서 그룹 서명 기법에는 강한 무죄입증성이 필요함을 역설한다.

V. 결론

이 논문에서는 정상적인 사용자들에게 발생할 수 있는 뜻하지 않은 예외 상황의 처리를 위해서 선형 암호문을 이용하는 그룹 서명 기법의 Open과 Judge가 어떻게 사용되어야하는 지 살펴보았다. MMO06과 같은 그룹 서명의 경우 정상적인 사용자들이 뜻하지 않은 오류로 인하여 그룹에서 퇴출당할 수 있음을 확인하였고, BBS04에 MMO06의 Open/Judge를 적용하였을 때에도 마찬가지였다. 이를 위한 이 논문에서는 사용자의 식별자만을 노출하는 OpenHU과 JudgeHU에 대해서 제안하였으며, 이러한 OpenHU/JudgeHU를 사용했을 때, 사용자의 그룹 서명이 멤버십 인증서 발행키를 가진 공격자에게도 추적되지 않음을 랜덤 오라클 모델에서 증명하였다. 여기에 더해, 복수의 동일한 사용자 식별자의 그룹 서명들을 빠르게 처리하기 위한 일괄 Open/BatchProof/BatchJudge를 추가적으로 제안하였다.

이처럼 그룹 서명 기법은 이상적인 환경을 가정하고 만들어졌기 때문에, 현실 세계를 충실히 고려하지 못한 측면이 많이 존재한다. 따라서 현실 세계의 응용 환경을 실제로 고려하였을 때, 그룹 서명 기법에 나타날 수 있는 문제점을 고찰해보고 이를 해결할 수 있는 연구가 지속되어야할 것이다.

참고문헌

[1] D. Chaum and E. van Heyst, "Group signatures," EUROCRYPT '91, LNCS 547. pp. 257-265, 1991.

[2] T. Makita, Y. Manabe, and T. Okamoto, "Short group signatures with efficient flexible join," Proceedings of the 2006 Symposium on Cryptography and Information Security, Jan. 2006.

[3] D. Boneh, X. Boyen, and H. Shacham, "Short Group Signatures," CRYPTO '04, LNCS 3152, pp. 41-55, 2004.

[4] M. Bellare, D. Micciancio, and B. Warinschi, "Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions," EUROCRYPT '03, LNCS 2656, pp. 614-629, 2003.

[5] M. Bellare, H. Shi, and C. Zhang, "Foundations of group signatures: The case of dynamic groups," CT-RSA '05, LNCS 3376, pp. 136-153, 2005.

[6] 강전일, 양대헌, 이석준, 이경희, "실생활 응용을 위한 짧은 그룹 서명 기법(BBS04)에 대한 연구", 정보보호학회논문지, 19(5), pp. 3-15, 2009년 10월.

[7] 황정연, 이석준, 정병호, 양대헌, "지역 연결성을 제공하는 효율적인 그룹 서명 기법", 대한전자공학회 하계종합학술대회발표집, pp. 863-865, 2010년 6월.

〈著者紹介〉

강 전 일 (Jeonil Kang) 학생회원
2003년 2월: 인하대학교 컴퓨터 공학과 졸업
2006년 2월: 인하대학교 정보통신대학원 석사
2006년 3월~현재: 인하대학교 정보공학과 박사 과정
〈관심분야〉 RFID 보안, 생체 인식 보안, 무선 센서 네트워크 보안, 무선 인터넷 보안, 웹 인증 보안, 암호 프로토콜

양 대 헌 (DaeHun Nyang) 정회원
1994년 2월: 한국과학기술원 과학기술 대학 전기 및 전자 공학과 졸업
1996년 2월: 연세대학교 컴퓨터 과학과 석사
2000년 8월: 연세대학교 컴퓨터 과학과 박사
2000년 9월~2003년 2월: 한국전자통신연구원 정보보호연구본부 선임연구원
2003년 2월~현재: 인하대학교 컴퓨터정보공학부 부교수
〈관심분야〉 암호 이론, 암호 프로토콜, 인증 프로토콜, 무선 인터넷 보안

이 경 희 (Kyunghee Lee) 정회원
1993년 2월: 연세대학교 컴퓨터과학과 학사
1998년 8월: 연세대학교 컴퓨터과학과 석사
2004년 2월: 연세대학교 컴퓨터과학과 박사
1993년 1월~1996년 5월: LG소프트(주) 연구원
2000년 12월~2005년 2월: 한국전자통신연구원 선임연구원
2005년 3월~현재: 수원대학교 전기공학과 조교수
〈관심분야〉 바이오인식, 정보보호, 컴퓨터비전, 패턴인식