특집논문-08-13-1-08

비대칭환경에 적합한 유동적인 묶음서명 생성기법

윤 택 영[a], 이 세 원[a], 박 영 호[b], 김 창 한[c], 임 종 인[a]‡

Flexible Batch Signing Techniques for Imbalanced Environments

Taek-Young Youn[a], Se-Won Lee[a] Young-Ho Park[b], Changhan Kim[c], Jongin Lim[a]‡

요 약

묶음서명 생성기법은 동시에 다수의 메시지에 대한 서명을 효율적으로 생성하는 암호학적 도구이다. 기존에 제안된 묶음서명 생성기법은 동시에 주어지지 않는 서명 요구들에 대한 묶음서명을 생성할 수 없다. 즉, 동시에 주어지는 서명 요청들에 대해서만 묶음서명 생성기법의 장점이 적용될 수 있다. 본 논문에서는 동시에 주어지지 않는 서명 요청에 대해서도 묶음서명을 생성할 수 있는 유동적 묶음서명 생성기법을 제안한다. 또한, 제안하는 유동적 묶음서명 생성기법은 통신주체의 연산 능력이 상이한 비대칭 통신환경에 적합하다.

ABSTRACT

Batch signature is a cryptographic tool which efficiently generates several signatures for multiple messages. Previously proposed batch signature schemes do not allow a signer to generate a signature immediately for sequentially asked signing queries. In other words, the previous schemes are not applicable when signing requests are not simultaneously given to the signer. In this paper, we propose flexible batch signatures which can provide batch signing for sequential signing requests. Moreover, our schemes are well suited for imbalanced environment where two communicating parties have different computing powers.

Keyword : Batch signing technique, Flexibility, Imbalanced environment

Ⅰ. 서 론

묶음서명 생성기법은 한 번 서명하는 시간과 비용으로 다수의 메시지에 대한 서명을 생성할 수 있도록 구성된 암호학적 방법이다. 지금까지 소수의 묶음서명 생성기법이 제안되었다[1][2][6]. 그 중에서 두 가지 기법[1][6]은 해쉬 함수로 주어진 메시지를 압축 한 후, 잘 알려진 DSA[5]나 Schnorr 서명[7]과 같은 서명 기법으로 서명을 하는 형태로 구성되어 있다. 두 기법의 차이점은 메시지의 압축 방법에 있다. 하나의 방법[1]은 주어진 메시지의 해쉬 함수값을 연접(concatenate)하여 연접된 값에 다시 해쉬 함수를 적용하는 방법을 사용한다. 또 다른 방법[6]은 Merkle의 바이너리 해쉬 트리(binary tree)를 이용해서 주어진 메시지들의 해쉬 값에 대한 바이너리 트리를 생성하는 방법을 사용한다. 메시지들에 대한 해쉬 값을 복원하기 위한 정보가 서명 값으로 포함된다. 두 묶음서명 생성방법에서 생성된 서명을 검증하기 위해서 검증자는 메시지들에 대한 해쉬 값을 계산하고 이에 대한 서명을 검증한다. 두 묶음서명 생성기법은 일반 모델로써 기반하고 있는 서명 생성기법의 종류에 구애받지

a) 고려대학교
Korea University
b) 세종 사이버대학교
Sejong Cyber University
c) 세명대학교
Semyeong University
‡ 교신저자 : 임종인(jilim@korea.ac.kr)

않는다. 즉, DSA나 Schnorr 이외에도 알려진 어떤 서명 기법을 사용하여서도 구성할 수 있다. 이와는 달리 고정된 구조로 설계된 묶음서명 생성기법이 있다. 이는 RSA 암호기법을 기반으로 설계되어 있다[2]. RSA 기반의 묶음서명 생성기법은 묶음서명을 생성할 서명의 개수에 따라 공개키를 설정하여 개별 서명을 다른 키로 생성하는 방법으로 구성된다.

위에서 언급한 기법들은 동시에 주어진 서명에 대해서만 묶음서명을 수행할 수 있다. 즉, 묶음서명 생성기법의 장점인 효율성을 유지하려면 서명을 생성하기 위한 메시지들이 동시에 주어져야 한다. 따라서 동시에 주어지지 않은 서명 요청들에 대해서는 묶음서명을 생성할 수 없다. 충분한 서명 요청이 모이면 묶음서명을 생성함으로써 효율성을 유지할 수 있으나 충분한 양이 모일 때까지 서명 요청자와 서명 생성자가 기다려야 하는 문제를 야기한다. 기존의 기법들에서는 서명 요청자와 서명 생성자가 기다리는 방식을 채택하지 않으면 비동시적으로 요청되는 서명 생성 질의를 묶음서명으로 처리할 수 없다. 그러나 실제 통신상에서 서명 요청자는 대기하기를 원하지 않으므로 위의 방법을 적용하여 묶음서명의 효율성을 유지하는 것은 비현실적이다. 따라서 기존의 묶음서명 생성기법들은 현실적으로 비동시적인 서명 요청을 묶음서명으로 처리할 수 없다. 비동시적인 서명 요청에 대한 묶음서명 생성 가능성 여부의 비교하기 위해 하나의 개념을 정의하자. 비동시적으로 주어지는 서명 요청에 대해서 서명자와 요청자가 충분한 양의 서명 요청이 모이기까지 기다리지 않고 비동시적인 서명 요청들에 대한 묶음서명을 생성할 수 있는 묶음서명 생성기법을 유동적(Flexible)인 묶음서명 생성기법이라고 하자. 앞에서 기술하였듯이, 기존에 제안된 묶음서명 생성기법들은 유동성을 제공하지 못하고 동시에 주어지는 요청들에 대해서만 묶음서명을 생성할 수 있다. 따라서 기존의 묶음서명 생성기법들은 동시에 주어지지 않는 서명 요청에 대해서는 묶음서명 생성기법의 장점을 충분히 활용하지 못한다. 묶음서명 생성기법은 아니지만 많은 지수승을 동시에 계산함으로써 많은 서명을 효율적으로 생성하는 효율적인 이산대수문제(DLP: discrete logarithm problem)기반 서명 생성 기법이 제안되었다[4]. 비록 동시에 많은 지수승을 계산함으로써 다수의 서명을 효율적으로 생성한다는 측면에서 다수의 서명을 동시에 계산함으로써 효율성을 제공하는 묶음서명 생성기법과 목적은 유사하지만 묶음서명 생성기법은 아니다.

본 논문에서는 최초의 유동적인 묶음서명 생성기법을 제안한다. 본 논문에서 제안하는 묶음서명 생성기법은 서명을 생성하기 위한 메시지가 동시에 주어지지 않은 경우에도 묶음서명의 장점인 효율성을 획득할 수 있는 유동성을 제공한다. 즉, 비동시적으로 주어지는 서명요청에 대해서도 묶음서명을 생성할 수 있다. 본 논문에서 제안하는 서명 기법은 기존의 기법들에서 소수의 위수를 갖는 단일 생성원을 사용하는 것과는 달리 서로 다른 위수를 가지는 다수의 생성원을 사용한다. 각 생성원은 서로 다른 서명을 생성되기 위해 사용되며 서명 생성을 위한 계산 과정이 묶음 단위로 한 번에 처리되어 계산상의 효율성을 제공한다. 본 논문에서 제안하는 서명은 6개의 서명 요청에 대해, 한 번의 지수승만 수행하고 각 서명 요청에 대해 한 번의 곱셈을 수행한다. 따라서 160비트 지수에 대한 지수승의 경우 평균적으로 한 개의 서명 요청을 41번의 곱셈으로 처리할 수 있다. 제안하는 서명기법은 고정된 기저(base)를 사용하여 연산을 수행하기 때문에 Comb-method[3]를 사용하면 23번의 곱셈으로 서명 요청을 처리할 수 있다. 이는 한 개의 원소를 사전에 계산하여 저장하는 경우만 고려한 것으로 추가적인 저장 공간의 사용을 허용하는 경우 더 큰 계산상의 개선을 기대할 수 있다. 또한 기본적인 묶음서명 생성기법을 변형하여 Merkle의 바이너리 해쉬 트리 기법을 적용하여 동시에 오는 서명 요청이 발생하는 경우 더 큰 효율성의 개선을 제공하는 묶음서명 생성기법을 제안한다. 본 논문에서 제안하는 유동적 묶음서명 생성기법들에서 서명 검증비용은 약간 증가하지만 서명 생성이 매우 효율적으로 수행되기 때문에 비대칭 통신 환경에 유용하게 사용될 수 있다. 비대칭 환경은 통신 두 주체의 연산 또는 통신 능력이 상이한 경우를 의미한다. 즉, 한 통신 주체가 상대적으로 큰 연산 또는 통신 능력을 가지는 환경을 의미한다. 비대칭 환경에는 두 통신 주체가 사용해야 하는 비용이 비대칭으로 발생하여 큰 능력을 가진 통신주체가 많은 비용을 사용하는 대신 적은 능력을 가진 통신주체가 적은 비용으로 통신에 참여할 수 있도록 구성함으로써 통신에 요구되는 비용을 능력에 준해 사용하도록 구성함으로써 전체 통신의 수행을 보다 효율적으로 구성하는 것이 중요하다. 예를 들어 살펴보자. 모바일 폰과 PDA와 같이 적은

연산능력을 가진 장비가 인증 등을 수행하기 위해 서명을 생성하여 등록된 서버에 전송하는 경우 서명 생성비용이 매우 적은 제안하는 기법이 유용하게 사용될 수 있다. 비대칭 환경의 또 다른 예로 다수의 클라이언트에게 서명을 생성해주어야 하는 서버의 환경을 고려할 수 있다. 다수의 클라이언트가 요청하는 많은 서명 요청을 처리함에 있어 제안하는 묶음서명 생성기법을 사용하면 거의 동시에 요청하는 서명 요청과 동시에 주어지지 않은 서명 요청들에 대해 모두 묶음서명 생성기법의 장점을 살려 효율적으로 서명을 생성해줄 수 있으므로 좀 더 효율적이고 신속하게 클라이언트들의 서명 요청을 처리할 수 있다.

II. 기존의 묶음서명 생성기법

본 장에서는 기존의 묶음서명 생성기법에 대해 간략히 살펴본다. 본 논문의 비교대상인 DLP 기반의 묶음서명 생성기법들[1,6]에 대해 기술한다. [1,6]에서 제안된 묶음서명 생성기법들은 일반적으로 설계되어 임의의 서명 기법을 사용할 수 있기 때문에 RSA와 같은 DLP기반이 아닌 서명 기법을 사용하여 구성할 수 있으나 본 논문에서는 DLP를 기반으로 설계하고 있으므로 동일한 연산 기반으로 구현하는 경우만 고려한다. 효율적으로 지수승 계산을 수행함으로써 효율적인 서명 생성을 제공하는 기법[4]은 목적이 유사하지만 묶음서명 생성기법은 아니므로 본 장에서의 논의에서 제외한다. 기존의 기법을 서명에서 다음과 같은 기호를 사용한다. $Sig(sk,m)$을 메시지 m의 개인키 sk에 대한 서명이라고 하자. 서명 알고리즘에 대응되는 검증함수 $Ver(pk,s,m)$는 메시지 m와 서명 알고리즘으로 생성된 m에 대한 서명 s와 공개키 pk를 입력으로 수행되며 서명이 올바른 경우 1을 출력하고, 올바르지 않은 경우 0을 출력한다. 서명 생성기법에서는 해쉬함수 $h()$을 사용한다.

1. 단순 묶음서명 생성기법[1] (SBS: Simple Batch Signature)

단순 묶음서명 생성기법 SBS에 대해 살펴보자. n개의 메시지 $m_1,\cdots,m_n$에 대한 서명이 요청되면 서명자는 다음과 같이 주어진 메시지에 대한 하나의 압축 값을 계산 한다: $HM=h(H(m_1)\|\cdots\|H(m_n))$. 계산된 압축 메시지를 임의의 DLP기반 서명기법을 사용하여 서명 값을 다음과 같이 생성 한다: $s=Sig(sk,HM)$. n개의 메시지 중에서 m_i에 대한 서명은 다음과 같다: (s,m_i,M_i). 이때, M_i는 각 메시지의 해쉬값 중에서 $h(m_i)$을 제외한 값이다. 즉, $M_i=(h(m_1),\cdots,h(m_{i-1}),h(m_{i+1}),\cdots,h(m_n))$이다. 이와 같이 계산된 서명이 검증자에게 주어지면 검증자는 m_i와 M_i를 사용하여 HM을 계산한다. 검증자는 계산된 HM와 HM에 대한 서명 s을 공개키로 검증 알고리즘을 사용하여 기반하고 있는 DLP기반 서명기법의 검증방법에 준하여 검증을 수행한다.

2. 트리 기반의 묶음서명 생성기법[6] (TBS: Tree-Based Batch Signature)

트리 기반의 묶음서명 생성기법 TBS는 SBS와 마찬가지로 다수의 메시지에 대한 해쉬 값을 생성하여 해당 값에 대한 서명을 생성한다. SBS와의 차이점은 해쉬 값을 생성하는 방법으로 Merkle의 바이너리 해쉬 트리를 이용하여 n개의 메시지 $m_1,\cdots,m_n$에 대한 해쉬 값을 생성한다. 압축 방식을 설명하기 위해 해쉬 트리와 관련된 두 가지의 개념을 정의한다. 루트(root)란 해쉬 트리의 최상위 부모노드로 정의 하고, 메시지에 대한 덮개 집합(CS: covering set)은 그 메시지와 함께 루트를 생성하기 위해 사용되는 값들을 모아놓은 집합이다. 그림 1에서 $h_{1,2,3,4,5,6}$은 루트이고

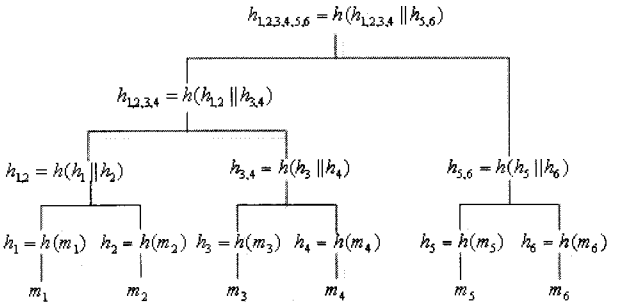


그림 1. 6개의 메시지에 대한 해쉬 트리의 구조
Fig. 1. Binary hash tree for six messages

$CS=\{h_{1,2}, h_4, h_{5,6}\}$는 메시지 m_3의 덮개 집합이다. m_3과 덮개 집합 CS로부터 루트는 다음과 같이 계산 할 수 있다. $h_{1,2,3,4,5,6}=h(h(h_{1,2},h(h(m_3),h_4)),h_{5,6})$. TBS에서는 이와 같은 방법으로 압축을 수행한 뒤, $m_1,\cdots,m_n$에 대한 루트 값 HM을 생성하고 이에 대한 서명을 계산한 뒤에 다음의 값을 서명으로 검증자에게 제공 한다: (s,m_i,CS). 검증자는 제공받은 서명 중에서 m_i와 CS를 사용하여 루트 HM를 복원하고 이후 서명 검증 과정을 SBS와 동일하게 수행한다.

Ⅲ. 새로운 묶음서명 생성기법

본 장에서는 기본적인 유동적인 묶음서명 생성기법과 Merkle 해쉬 트리를 적용한 기법을 제안한다. n개의 비동시적으로 요청되는 서명생성 요구에 대해 묶음서명을 유동적으로 생성할 수 있는 서명기법을 n-묶음서명 생성기법이라고 하자.

1. 유동적인 묶음서명 생성기법 1 (FBS1: Flexible Batch Signature 1)

본 절에서는 유동적으로 묶음서명을 생성하는 기법을 제안한다. 우선, 설명을 위해 2-묶음서명 생성기법을 기술한다. 즉, 2개의 비동시적으로 주어지는 서명 요청에 대한 묶음서명을 생성하는 경우를 보인다.

· **Setup(k)** : 길이가 k인 소수 q_1, q_2에 대하여 $q_1|p-1$, $q_2|p-1$인 소수 p를 선택한다. 위수가 q_1, q_2인 생성자 g_1, g_2를 생성한다. $k_1=q_2^{-1}(\mathrm{mod}q_1)$, $k_2=q_1^{-1}(\mathrm{mod}q_2)$ 일 때, $g=g_1^{k_1}g_2^{k_2}(\mathrm{mod}p)$라고 하자. 정의에 의하여 $g_1^{q_1}=1(\mathrm{mod}p)$, $g_2^{q_2}=1(\mathrm{mod}p)$이 됨을 알 수 있다. 또한, $g^{q_2}=(g_1^{k_1}g_2^{k_2})^{q_2}=g_1^{k_1q_2}g_2^{k_2q_2}=g_1^{k_1q_2}=g_1(\mathrm{mod}p)$이 만족하므로 $g_1=g^{q_2}(\mathrm{mod}p)$임을 알 수 있다. 이와 유사한 방법으로 다음이 만족함을 확인할 수 있다: $g_2=g^{q_1}(\mathrm{mod}p)$. $h_1: \{0,1^*\}\rightarrow Z_{q_1}$와 $h_2: \{0,1\}^*\rightarrow Z_{q_2}$는 암호학적 해쉬 함수로 정의한다. 주어진 안전성 파라미터 k에 대하여 시스템 파라미터는 $params=\{p,q_1,q_2,g_1,g_2,g,h_1,h_2\}$이다.

· **KeyGen($params$)** : 사용자는 두 개의 비밀정보 $x_1\in Z_{q_1}$와 $x_2\in Z_{q_2}$를 선택하여 $y_1=g^{q_2x_1}(\mathrm{mod}p)$와 $y_2=g^{q_1x_2}(\mathrm{mod}p)$를 계산한다. 사용자의 공개키는 $y=y_1^{k_1}y_2^{k_2}$이고, 개인키는 $sk=\{x_1,x_2\}$이다. 변수의 선택 방법에 의해 $y_1=g_1^{x_1}(\mathrm{mod}p)$가 만족한다. 따라서 $y_1^{q_1}=g_1^{x_1q_1}=(g_1^{q_1})^{x_1}=1(\mathrm{mod}p)$가 만족하고, 유사한 방법으로 $y_2^{q_2}=1(\mathrm{mod}p)$가 만족함을 알 수 있다. 결과적으로 $y^{q_2}=y_1(\mathrm{mod}p), y^{q_1}=y_2(\mathrm{mod}p)$이 만족하는 것을 확인할 수 있다.

· **BatchSign($params$, sk, m_1, m_2)** : 두 개의 메시지 m_1과 m_2에 대한 서명이 요청되는 경우를 고려한다. 제안하는 기법은 비동시적인 서명 요청에 묶음 서명을 생성할 수 있으므로 두 개의 서명 요청이 동시에 발생한다는 가정이 필요하지 않다. 두 개의 메시지 중 m_1가 서명을 위해 요청되면 서명자는 난수 $r\in\{0,1\}^k$을 선택하여 $\beta=g^r(\mathrm{mod}p)$를 계산한다. 서명자는 다음과 같이 α_1를 계산 한다: $\alpha_1=x_1h_1(m_1\|\beta)+r(\mathrm{mod}q_1)$. m_1에 대한 서명은 $\sigma_1=(\alpha_1,\beta)$이다. m_1에 대한 서명 요청이 처리된 이후 m_2에 대한 서명 요청이 발생하면 서명자는 m_1에 대한 서명 과정에서 사용한 β를 재사용하여 $\alpha_2=x_2h_2(m_2\|\beta)+r(\mathrm{mod}q_2)$를 계산한다. m_2에 대한 서명은 $\sigma_2=(\alpha_2,\beta)$이다.

· **Verify($params$, pk, σ_i, m_i)** : 메시지 m_i에 대한 서명이 $\sigma_i=(\alpha_i,\beta)$라고 하자. 사용자는 주어진 서명이 올바른 서명인지 검사하기 위해 다음 식을 확인하여 검증한다:

$$(y_i^{h_i(m_i\|\beta)}\beta/g^{\alpha_i})^{q_{3-i}}(\mathrm{mod}p)=1.$$

동일한 조건은 다음과 같은 식을 통해서도 확인할 수 있다: $(y_i^{h_i(m_i\|\beta)}\beta)^{q_{3-i}} = (g^{q_{3-i}})^{\alpha_i} (\mathrm{mod} p)$. 다음의 식을 통해 검증식이 주어진 서명을 검증하기 위해 올바르게 사용될 수 있음을 확인할 수 있다:

$$\begin{aligned}(y^{h_i(m_i\|\beta)}\beta)^{q_{3-i}} &= y_i^{h_i(m_i\|\beta)}(g^r)^{q_{3-i}} \\ &= (g_i^{x_i})^{h_i(m_i\|\beta)} g_i^r \\ &= g_i^{x_i h_i(m_i\|\beta)+r} \\ &= (g^{q_{3-i}})^{\alpha_i} (\mathrm{mod} p).\end{aligned}$$

앞에서 기술된 서명 과정은 6-묶음서명을 위한 형태로 확장 할 수 있다. p가 k_1-bit의 소수라고 하면, 서로 다른 생성자의 k_2-bit 소수 위수는 최대한 $\lfloor k_1/k_2 \rfloor$ 개 존재한다. 현재 많은 시스템에서 사용되고 있는 1024-bit 소수의 경우 최대 6개 (= $\lfloor 1024/160 \rfloor$)의 160-bit 소수를 선택할 수 있다. 즉, 널리 사용되는 변수의 크기에서는 최대 6-묶음서명 생성이 가능함을 의미한다. 6-묶음서명 생성을 위한 시스템 파라미터는 $params = \{p, Q, g, H\}$이다. 여기서 p는 1024-bit 소수이고, q_i는 $q_i | p-1$을 만족하는 160-bit 소수이며, $Q = \{q_1, \cdots, q_6\}$, $g = \prod_{i=1}^{6} g_i^{k_i} (\mathrm{mod}\ p)$, $k_i = ((\prod_{j=1}^{6} q_j)/q_i)^{-1} (\mathrm{mod}\ q_i)$, $H = \{h_1, \cdots, h_6\}$로 정의된다. 또한, g_i는 위수가 q_i인 생성자로 정의되며 $h_i: \{0,1\}^* \rightarrow Z_{q_i}$는 암호학적 해쉬 함수로 정의된다. 이와 같은 파라미터의 존재성은 V장에서 실험을 통해 하나의 예를 찾음으로써 보인다. 6-묶음서명 생성에서 서명자는 6개의 개인키 $x_i \in Z_{q_i}$를 선택하고, $y_i = (g^{(\prod_{j=1}^{6} q_j)/q_i})^{x_i}$와 $y = \prod_{i=1}^{6} y_i^{k_i}$를 계산하여 공개키 y를 생성한다. 서명자는 개인키 $\{x_1, x_2, x_3, x_4, x_5, x_6\}$로 6개의 메시지에 대해서 $\alpha_i = x_i h_i(m_i\|\beta) + r (\mathrm{mod}\ q_i)$와 $\beta = g^r (\mathrm{mod}\ p)$를 계산한다. 메시지 m_i에 대한 서명은 $\sigma_i = (\alpha_i, \beta)$이다. 서명의 검증은 다음식을 사용하여 수행한다: $(y_i^{h_i(m_i\|\beta)}\beta)^{(\prod_{j=1}^{6} q_j)/q_i} = 1 (\mathrm{mod}\ p)$. α_i가 다른 α_j와 β를 공통의 입력으로 사용함에도 불구하고 독립적으로 계산할 수 있기 때문에 β가 계산되면 동시적으로 요청되지 않는 서명 요청에 대해 α_i를 독립적으로 생성하여 응답할 수 있다.

2. 유동적인 묶음서명 생성기법 2 (FBS2: Flexible Batch Signature 2)

FBS1에 Merkle 해쉬 트리를 적용하여 동시에 다수의 서명 요청이 있는 경우를 보다 효율적으로 처리함으로써 효율성을 개선할 수 있는 묶음서명 생성기법(FBS2)을 제안한다. 여기서는 2-묶음서명 생성을 제공하는 경우를 기준으로 설명한다. FBS2도 FSB1과 마찬가지로 6-묶음서명 생성기법으로 확장 할 수 있으며, 이는 명백하므로 별도로 기술하지 않는다.

Setup(k) & KeyGen($params$) : FBS1과 과정이 동일하다.

· **BatchSign**($params$, sk, M_1, M_2) : 두 메시지의 집합 M_1과 M_2가 서명 요청으로 주어진다고 가정하자. 각 메시지 집합은 동시에 주어지는 것으로 가정하고 두 집합은 동시에 주어지지 않는다고 가정한다. 두 메시지의 집합을 $M_1 = \{m_{1,i_1} | i_1 \in [1,b_1]\}$와 $M_2 = \{m_{2,i_2} | i_2 \in [1,b_2]\}$라고 하고 b_1, b_2는 각각 M_1, M_2에 포함된 메시지의 개수라고 하자. 그림1에서 설명된 방법으로 서명자는 M_1과 M_2에 대한 해쉬 트리를 구성하여 각 해쉬 트리의 루트를 m_1, m_2로 계산한다. 서명자는 난수 $r \in \{0,1\}^k$를 선택하여 $\beta = g^r (\mathrm{mod}\ p)$를 계산하고 이를 사용하여 다음을 계산한다: $\alpha_1 = x_1 h_1(m_1\|\beta) + r (\mathrm{mod}\ q_1)$. $\alpha_2 = x_2 h_2(m_2\|\beta) + r (\mathrm{mod}\ q_2)$. 메시지 $m_{i,j}$에 대한 서명은 $\sigma_{i,j} = (\alpha_i, \beta, CS_{i,j})$ 이다. 이때, $i \in \{1,2\}$, $j \in [1,b_i]$이고, $CS_{i,j}$는 메시지 $m_{i,j}$에 대한 덮개 집합이다.

· **Verify**($params$, pk, $\sigma_{i,j}$, $m_{i,j}$) : 사용자가 메시지 $m_{i,j}$와 그 메시지의 서명 $\sigma_{i,j}$를 받으면 $m_{i,j}$와 $CS_{i,j}$를 이용하여 m_i를 계산한다. 사용자는 다음 식을 확인함으로써 주어진 서명의 정당성을 검증한다:

$$(y_i^{h_i(m_i\|\beta)}\beta / g_i^{\alpha_i})^{q_{3-i}} = 1 (\mathrm{mod}\ p).$$

위 등호가 만족하면 서명 $\sigma_{i,j}$은 유효한 서명이다.

Ⅳ. 분 석

이 장에서는, FBS1과 FBS2의 안전성과 효율성에 대해 분석한다. Schnorr 서명 기법과 묶음 지수승 계산 기법(batch exponentiation technique)을 각각 SCS(Schnorr Signature) 와 BEXP(Batch Exponentiation)로 표기한다. 제안하는 서명 기법의 안전성은 안전성이 검증되어 널리 사용되고 있는 SCS를 기반으로 분석한다.

1. 안전성

제안한 기법과 이전 기법들의 차이점 중에 하나는 동일한 난수를 다수의 서명을 생성하기 위해 사용한다는 점이다. SCS와 DSA 같은 서명의 경우, 동일한 난수 r을 다수의 서명에서 사용하면 공격자가 개인키를 계산할 수 있다.[1] 본 논문에서 제안하는 묶음서명 생성기법은 같은 서명 묶음에서 다른 메시지들에 동일한 난수를 적용하여 서명을 생성하지만, SCS와 DSA를 분석하는 방법으로 공격자가 개인키를 계산할 수는 없다. 6-묶음서명 생성기법의 경우를 살펴보자. 많은 요청을 했을 경우 그만큼 많은 정보가 드러날 수 있기 때문에 가장 많은 요청을 할 수 있는 6-묶음서명 생성의 경우를 고려한다. 6-묶음서명을 수행하는 경우에 공격자는 다음과 같이 동일한 난수 r로 생성된 정보들을 얻을 수 있다:

$$x_1h_1(m_1\|\beta)+r(\mathrm{mod}q_1),\ x_2h_2(m_2\|\beta)+r(\mathrm{mod}q_2)$$
$$x_3h_3(m_3\|\beta)+r(\mathrm{mod}q_3),\ x_4h_4(m_4\|\beta)+r(\mathrm{mod}q_4)$$
$$x_5h_5(m_5\|\beta)+r(\mathrm{mod}q_5),\ x_6h_6(m_6\|\beta)+r(\mathrm{mod}q_6)$$

여기서 $\beta=g^r(\mathrm{mod}p)$이다. SCS, DSA와 같은 서명 기법에서는 공격자가 같은 난수 r을 사용해서 생성된 서명을 위에서 전개된 것과 같이 방정식으로 구성하여 개인키를 얻을 수 있다. SCS, DSA 등의 서명 기법에서는 서로 다른 서명을 생성하기 위해 다른 난수를 사용함으로써 이러한 공격을 막고 있다. 제안된 FBS1, FBS2도 역시 위의 방정식을 풀면 개인키를 얻을 수 있다. 하지만 각각의 서명에 다른 개인키를 사용하고 있기 때문에 미지수의 개수보다 다항식의 개수가 적어 연립 방정식을 풀 수 없다. 또한, 각 방정식은 서로 다른 법(modulo)에서의 계산이다. 따라서 위의 연립 방정식을 계산하여 개인키 또는 난수 r을 알아낼 수 없다.

이전 서명기법과 제안한 서명 기법의 또 다른 차이점은 하나의 공개키 y에 대응되는 개인키의 수이다. 일반적으로 하나의 공개키에 하나의 개인키가 대응되어 사용된다. 본 논문에서 제안하는 서명 기법에서는 하나의 공개키에 최대 6개의 개인키가 대응된다. 그러나 주어진 공개키를 변형함으로써 각 개인키에 대응되는 공개키를 유일하게 결정할 수 있다. x_i에 대응되는 공개키는 $y_i=(g^{(\prod_{j=i}^{6}q_j)/q_i})^{x_i}$이다. $l\neq i$일 경우 $q_l|\gamma_i$와 $y_l^{q_l}=1\ (\mathrm{mod}\ p)$가 만족하기 때문에 $y_l^{\kappa_l\gamma_i}=1\ (\mathrm{mod}\ p)$이므로 $y=\prod_{l=1}^{6}y_i^{\kappa_i}$에서 각 y_i가 계산되는 것은 다음과 같이 확인할 수 있다: $y=\prod_{l=1}^{6}y_i^{\kappa_l\gamma_i}=y_i^{\kappa_i\gamma_i}=y_i$. 이때, $\gamma_i=(\prod_{j=1}^{6}q_j)/q_i$.이다. 하나의 공개키에서 각 개인키에 대응되는 공개키에 해당하는 값이 유일하게 결정되므로 제안한 기법에서 다수의 개인키에 하나의 공개키를 공개하여 사용하는 것은 문제가 되지 않는다.

2. 효율성

표 1에서 본 논문에서 제안하는 두 기법과 기존의 묶음서명 생성기법인 SBS와 TBS, 그리고 SCS를 비교한다. SBS와 TBS는 DLP 기반 서명기법을 기반으로 구현하는 경우를 고려한다. 묶음서명 생성기법과 유사하게 다수의 서명을 효율적으로 생성하기 위한 목적으로 제안된 BEXP기반의 서명도 비교하며 이를 BEXPS로 표기한다. 표 1에서는 지수승의 개수를 기준으로 효율성을 비교한다. 해쉬 함수의 계산, 두 수의 덧셈 및 곱셈 등은 지수승에 비해 매우 적은 연산량을 요구하므로 서명 기법들의 효율성 비교에서 배재한다. M은 곱의 연산 비용이고, E는 지수승의 연산 비

1) SCS와 DSA를 포함한 이전 기법에서 이러한 공격을 막으려면 다른 서명 값에 대하여 다른 난수를 사용해야 한다.

용(160-bit 지수의 경우)이으로 정의한다. 따라서 $160t$-bit 지수승의 경우 tE의 연산량이 요구된다. n은 서명의 요청수, n_b는 동시에 계산할 묶음의 크기로 정의한다. n_k는 BEXP의 효율성을 결정하는 상수이다. m은 6 이하의 자연수로 본 논문에서 제안하는 서명 기법에서 허용하는 묶음의 크기를 의미한다. 묶음서명 생성기법이 아닌 비교대상에 대한 유동성을 논하는 것은 무의미하므로 "-"로 명시하여 비교 대상이 아님을 밝힌다. 언급했듯이, BEXP기반의 서명은 효율적으로 다수의 서명을 생성하기 위한 기법이라는 측면에서 묶음서명 생성기법과 유사한 목적을 가지나 묶음서명 생성기법이 아니다. 따라서 유동성 측면에서 비교하지 않는다.

표 1. 효율성과 특징 비교
Table 1. Comparison

	Sign	Verification	Flexibility
SCS	nE	$2E$	-
SBS/TBS	n_bE	$2E$	X
BEXPS	n/n_kE	$2E$	-
FBS1	$\lceil n/m \rceil E$	$(1+m)E$	O
FBS2	$\lceil n_b/m \rceil E$	$(1+m)E$	O

m=6일 경우 FBS1은 SCS가 서명을 생성하기 위한 비용의 17%만으로 단일 서명을 생성할 수 있다. 한 번의 서명 생성 비용으로 여섯 개의 의 서명에 대한 응답을 수행할 수 있기 때문에 한 개당 평균 1/6(17%)의 서명 생성 비용이 사용되어 개별 서명이 평균 17%의 비용으로 생성된다. 마찬가지로, FBS2는 TBS가 사용하는 비용의 17%로 동일한 환경에서 서명 요청을 처리할 수 있다. FBS1이 6개의 비동시적인 요청에 대한 묶음서명을 생성하는 경우를 고려하면 1번의 지수승과 6번의 곱셈이 사용된다. 160-bit 지수승의 경우 1번의 지수승 계산에 240번의 곱셈이 요구되므로 평균적으로 $41=(240+6)/6$번 곱셈으로 한 개의 서명 요청을 처리할 수 있다.

효율성의 비교를 위하여 묶음 서명은 아니지만 효율적으로 서명을 생성하는 방법인 BEXP의 효율성을 살펴보자. BEXP의 효율성은 n_k에 의해서 결정된다. 표 2를 보면, 108

표 2. 묶음 지수승의 효율성(table2. [4])
Table 2. Efficiency of Batch Exponentiation(table2. [4])

크기	서명		Verification	n_k
Batch Exp	비용	저장공간	비용	
2	$141M$	316Byte	2E	1.70
3	$103M$	652Byte	2E	2.33
4	$85M$	1324Byte	2E	2.84
12	$58M$	3844Byte	2E	4.15
36	$49M$	11404Byte	2E	4.90
108	$46M$	34084Byte	2E	5.22

번 지수승을 동시에 수행하는 경우 34,084Byte의 저장 공간이 요구되며, 이때의 평균 서명 비용은 $46M$이다. 서명의 비용은 곱셈의 개수로 비교하고 검증의 비용은 지수승의 개수로 비교한다. 앞에서 언급되었듯이 160-bit 지수승의 경우를 고려하면 1번의 지수승 계산에 240번의 곱이 요구된다. 앞에서 언급되었듯이 제안하는 서명기법에서 지수승 연산을 위한 계산량은 평균적으로 $41M$이다. 그리고 제안하는 기법은 서명을 생성하는 과정에서 고정된 기저를 사용하므로 Comb-method[3]를 사용하여 효율성을 더욱 개선할 수 있다. 한 개의 원소를 미리 계산하여 저장하는 경우를 고려하면, 23번의 곱셈으로 서명 요청을 처리할 수 있다. 추가적인 저장 공간의 사용을 감안하면 더 큰 계산상의 개선을 기대할 수 있다. 따라서, 제안하는 묶음서명 생성기법은 BEXP기반의 서명 생성기법보다 훨씬 효율적이다. 참고로, BEXP는 Comb-method[3]를 적용할 수 없다.

3. 유동성

제안하는 서명 기법들은 기존의 묶음서명 생성기법과는 달리 유동성을 제공한다. 기존의 묶음서명 생성기법은 묶음서명 생성기법의 장점을 활용하기 위해 대기시간이 필요하다. 그리고 서명 요청자가 대기시간을 허용한다는 가정은 현실적이지 않으므로 기존의 기법은 유동적으로 묶음서명을 생성할 수 없다. 기존의 기법에서는 서명할 메시지에 의존적으로 서명이 생성된다. 즉, 다수의 메시지에 대한 압축 값을 계산하고 이에 대한 서명을 생성하기 때문에 메시지들에 대한 압축 값을 계산하는 단계에서 서명 대상으로 포함되지 않으면 묶음 서명으로 포함될 수 없다. 본 논문에

서 제안하는 서명기법은 각 메시지에 대한 서명이 공통적으로 포함하는 부분이 메시지에 의존적이지 않기 때문에 독립적으로 서명을 생성할 수 있다. 즉, $\beta = g^r \pmod p$가 한번 생성되면 이를 사용하여 6개의 서명을 생성할 수 있으므로 각 서명은 독립적으로 생성할 수 있고 이에 따라 유동성이 제공된다.

V. 파라미터 선택방법

본 장에서는 서로 다른 160-bit 소수 q_i에 대하여 $q_i | p-1$를 만족하는 소수 p를 생성하는 알고리즘을 제공한다. P_k를 $[2^k, 2^{k+1}]$ 범위 안의 소수들의 집합이라고 하자. $a \leftarrow b$는 a에 b를 입력하는 것을 의미하고, $a \leftarrow_R S$는 집합 S에서 a를 임의로 선택하는 것을 의미한다.

알고리즘 1 : 묶음서명 생성기법을 이용한 소수 생성 (6 묶음의 경우)

Step 1. for $(i=1; i<7; i++)$ $q_i \leftarrow_R P_{160}$

Step 2. $Q \leftarrow \prod_{i=1}^{6} q_i$

Step 3. $q \leftarrow_R Z \;\; s.t. \; |2qQ+1| = 1024$

Step 4. $Q \leftarrow 2q\prod_{i=1}^{6} q_i + 1$

Step 5. p가 소수일 경우 $q_1, q_2, q_3, q_4, q_5, q_6, q$를 출력하고, p가 소수가 아니면 Step1으로 간다.

제안하는 묶음서명 생성기법을 위한 파라미터를 생성하기 위해 Window XP, pentium 4, intel CPU 1.2 GHz, 1GB RAM의 환경에서 알고리즘을 실행했다. 알고리즘 1을 사용하여 1분 내에 묶음서명 생성기법에서 사용되기 위한 조건을 만족하는 변수를 찾을 수 있었다. 위 알고리즘은 확률적으로 동작하는 것이기 때문에 각 수행에서 다른 시간이 요구된다. 또한, 조건을 만족하는 변수의 존재성을 보이기 위한 구현이므로 최적화되지 않아 최적화하여 구현하는 경우에 비해 많은 시간이 소요되었을 것으로 예상된다. 따라서 위 알고리즘의 수행시간은 실험 환경에 따라 다르게 나타날 수 있다. 또한, DLP 기반 암호기법에서는 p와 같은 시스템 변수를 공유하여 사용할 수 있으므로 본 알고리즘 수행 결과의 의미는 본 논문에서 사용하고자 하는 변수의 존재성을 보이는 것에 있다. 다음은 알고리즘 1을 사용하여 생성한 변수이다.

$q_1 = 2^{160} + 23B54F501274F91B591D19A27A19FA237E4F59AB$

$q_2 = 2^{160} + 9A27E414F91BA19F5012754FD591A237F591C387$

$q_3 = 2^{160} + 5012754F4F91B59127A19F501274F91B591A23B7$

$q_4 = 2^{160} + C23B54FD19A2F50127EAF5914F91B591A2377A57$

$q_5 = 2^{160} + 4F5919A22741C23B54591A237EFDF91B7A19F643$

$q_6 = 2^{160} + 41274F91B9A27A19F5A237E4F595911C23B551AF$

$q = 19A27A19F2741FA5$

VI. 결 론

본 논문에서는 두 개의 유동적인 묶음서명 생성기법, FBS1와 FBS2, 을 제안했다. 제안하는 묶음서명 생성기법은 유동적이기 때문에 비동시적인 서명요청을 처리할 수 있고, 이는 비대칭 환경에 적합하게 사용될 수 있는 장점이 된다. 이와 같은 특성은 모바일 통신장비 등의 저전력 디바이스와 같이 적은 계산 및 통신 능력을 가지는 장비에서 인증 등을 수행하기 위한 목적으로 적합하게 사용될 수 있다. 예를 들어, 핸드폰이나 PDA와 같은 장비가 인증을 위한 서명 생성을 시도하는 경우, 서명 생성비용이 매우 적기 때문에 장비에서 소요되는 비용은 매우 적다. 해당 장비를 확인하기 위해 검증을 수행하는 서버는 저전력 장비에 비해 매우 큰 연산 능력을 갖고 있기 때문에 서명 생성 비용의 감소에 따라 발생하는 검증 비용의 증가가 부담이 되지 않는다. 이 외에도 다양한 비대칭 환경에서 유용하게 사용될 수 있을 것으로 기대된다.

참 고 문 헌

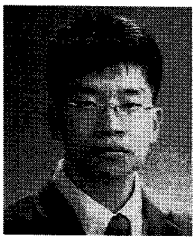
[1] W. C. Cheng, C-F Chou, and L. Golubchik, "Performance of Batch-based Digital Signatures," in Proc. of the 10th IEEE International Symposium on Modeling, Analysis, & Simulation of Computer & Telecommunications Systems (MASCOTS'02), pp. 77-85.

[2] A. Fiat, "Batch RSA", in Advances in Cryptology - Proceedings of Crypto'89, Lecture Notes in Computer Science, Vol. 435, pp. 175-185.
[3] C. Lim, and P. Lee, "Flexible Exponentiation with Precomputation", in Advances in Cryptology - CRYPTO'94, Lecture Notes in Computer Science 839, pp. 239-252, Springer-Verlag, 1994.
[4] D. M'Raïhi, and D. Naccache, "Batch Exponentiation: A Fast DLP-based Signature Generation Strategy", in Proc. of the 3rd ACM Conference on Computer and Communications Security (CCS'96), pp. 58-61.
[5] NIST, "Digital Signature Standard (DSS)," in FIPS Federal Register, Vol.56, No.169, 1991.
[6] C. J. Pavlovski, and C. Boyd, "Efficient Batch Signature Generation using Tree Structures," in Proc. of the International Workshop on Cryptographic Techniques and E-Commerce (CrypTEC'99), City University of Hong Kong Press, pp. 70-77.
[7] C.P. Schnorr, "Efficient identification and signatures for smart cards," in Advances in Cryptology - Proceedings of EUROCRYPT'89, Lecture Notes in Computer Science, pp. 239-252, 1989.

저 자 소 개

윤 택 영

- 2003년 2월 : 고려대학교 수학과 이학학사
- 2005년 2월 : 고려대학교 정보보호대학원 정보보호학과 공학석사
- 2005년 3월 ~ 현재 : 고려대학교 정보경영공학전문대학원 정보보호학과 박사과정
- 주관심분야 : 암호 이론, 정보보호 이론, 암호 프로토콜, 부채널 공격



이 세 원

- 2006년 2월 : 숭실대학교 수학과 이학학사
- 2006년 9월 ~ 현재 : 고려대학교 정보경영공학전문대학원 정보보호학과 석사과정
- 주관심분야 : 암호 이론, 정보보호 이론, 암호 프로토콜
- 주관심분야 : 암호 이론, 정보보호 이론, 공개키 암호 알고리즘

박 영 호

- 1990년 2월 : 고려대학교 수학과 이학학사
- 1993년 2월 : 고려대학교 수학과 이학석사
- 1997년 2월 : 고려대학교 수학과 이학박사
- 2002년 3월 ~ 현재 : 세종 사이버 대학교 조교수
- 주관심분야 : 정수론, 공개키 암호, 암호 프로토콜, 부채널 공격

김 창 한

- 1985년 2월 : 고려대학교 수학과 이학학사
- 1987년 2월 : 고려대학교 수학과 이학석사
- 1992년 2월 : 고려대학교 수학과 이학박사
- 2002년 2월 ~ 현재 : 세명대학교 정보통신학부 부교수
- 주관심분야 : 정수론, 공개키 암호, 암호 프로토콜

저 자 소 개

임 종 인

- 1980년 2월: 고려대학교 수학과 이학학사
- 1982년 2월: 고려대학교 수학과 이학석사
- 1986년 2월: 고려대학교 수학과 이학박사
- 1999년 2월~현재: 고려대학교 정보보호대학원 원장, CIST 센터장
- 주관심분야 : 암호 이론, 정보보호 정책