情報保護學會論文誌
第16卷 第4號, 2006. 8

안전한 RFID 프라이버시 보호 프로토콜을 위한 백엔드 서버의 태그 판별 시간 절감 기법*

여 상 수,[1†] 김 순 석,[2] 김 성 권[3‡]
[1]단국대학교, [2]한라대학교, [3]중앙대학교

Tag Identification Time Reduction Scheme of Back-End Server for Secure RFID Privacy Protection Protocol*

Sang-Soo Yeo,[1†] Soon-Seok Kim,[2] Sung Kwon Kim[3‡]
[1]Dankook University, [2]Halla University, [3]Chung-Ang University

요 약

RFID 기술은 바코드 시스템에는 없는 여러 가지 특징들 때문에 유비쿼터스 환경의 핵심기술로 평가되고 있다. 그러나 RFID 시스템은 정보 유출과 위치 추적 등과 같은 사용자 프라이버시 침해 문제를 가지고 있다. RFID 시스템에서 사용자의 프라이버시를 완전하게 보호하기 위해서는 기밀성, 불구분성, 전방 보안성 등의 3가지 필수 보안 요건을 만족시키는 RFID 프라이버시 보호 프로토콜이 필요하다. 기존에 제안된 프로토콜 중에서 이 3가지 필수 보안 요건을 만족하는 안전한 RFID 프라이버시 보호 프로토콜은 Ohkubo가 제안한 해시 체인 기반 프로토콜이다. 불행히도 이 프로토콜은 백엔드 서버에서 태그를 판별하는 시간이 매우 길다는 큰 단점이 있다. 본 논문에서는 Ohkubo가 제안한 프로토콜의 안전성은 그대로 유지하면서 백엔드 서버에서 태그를 판별하는데 걸리는 계산 시간을 단축하는 기법을 제안한다. 제안하는 기법은 백엔드 서버에서 계산 시간을 Ohkubo 프로토콜의 원래 기법보다 현저하게 단축시키는 결과를 보여준다.

ABSTRACT

RFID technology is evaluated as one of core technologies for ubiquitous environment, because of its various characteristics which barcode systems don't have. However, RFID systems have consumer's privacy infringement problems, such like information leakage and location tracing. We need RFID privacy protection protocols, that satisfy three essential security requirements; confidentiality, indistinguishability and forward security, in order to protect consumer's privacy perfectly. The most secure protocol, that satisfies all of the three essential security requirements, among existing protocols, is the hash-chain based protocol that Ohkubo proposed. Unfortunately this protocol has a big disadvantage that it takes very long time to identify a tag in the back-end server. In this paper, we propose a scheme to keep security just as it is and to reduce computation time for identifying a tag in back-end server. The proposed scheme shows the results that the identification time in back-end server is reduced considerably compared to the original scheme of Ohkubo protocol.

Keywords : *RFID, Privacy, Pre-computation, One-way hash function*

접수일: 2006년 2월 9일 ; 채택일: 2006년 7월 20일
* 본 연구는 한국과학재단 특정기초연구(R01-2005-000-10568-0) 지원으로 수행되었음.
† 주저자, ssyeo@dankook.ac.kr
‡ 교신저자, skkim@cau.ac.kr

Ⅰ. 서 론

현재 우리 주위에서 바코드를 이용한 자동인식 시스템은 유통, 물류 분야 등 곳곳에서 볼 수 있다. 바코드는 판독의 신뢰성 면이나 가격 면에서 다른 방식보다 유리한 점을 가지고 있기 때문에 자동인식 분야에서 많이 쓰이고 있다. 그러나 최근 들어서는 바코드 시스템보다 더 많은 장점을 가지고 있는 RFID 시스템에 대한 관심이 고조되고 있다. 바코드 시스템과 비교하여 RFID 시스템이 가지는 장점은 다량의 물품을 한 번에 처리가 가능하다는 점과 실시간으로 물품의 정보를 파악 가능하다는 점, 다양한 형태의 데이터 기록이 가능하다는 점, 그리고 넓고 광범위한 지향성을 가지고 있다는 점이다. 이처럼 RFID 시스템은 바코드 시스템에 비해 많은 장점을 가지고 있지만 대부분 태그의 가격이 고가였기 때문에 많은 분야에서 활용이 어려웠고, 바코드 시스템의 적용 분야보다는 작은 활용 분야를 가지고 있었다. 그러나 최근에는 RFID 태그 하드웨어에 대한 기술 개발과 대량생산 체제의 확보로 말미암아 RFID 시스템의 구축 비용이 하락하게 되었고, 활용분야가 점차적으로 넓어지게 되었다[1].

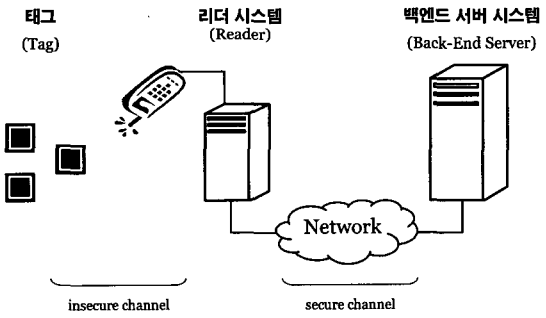


그림 1. RFID 시스템의 구조

RFID 시스템은 일반적으로 태그(tag), 리더(reader), 백엔드 서버(back-end server) 등으로 구성된다(그림1). 태그는 트랜스폰더(trans-ponder)라고도 불리며, 특정 제품에 부착된다. 태그는 특정 제품이 가지고 있는 고유의 정보 즉, 제품의 고유 *ID*(identifier)를 가지게 된다. 태그는 리더의 요청에 따라서 자신의 *ID*를 리더에게 전송하게 된다. 리더는 태그가 보내오는 *ID*를 수신한 후, 백엔드 서버에게 전달하게 된다. 백엔드 서버는 자체의 데이터베이스를 이용하여 태그의 *ID*를 판별하여 리더에게 알려주게 된다. 다시 말해서, 백엔드 서버는 합법적인 리더의 태그 *ID* 판별 요청을 받아서 태그를 판별한 후, 그에 해당하는 제품 정보를 리더에게 제공해 주는 역할을 하게 되는 것이다[1].

일반적으로 태그와 리더는 무선 통신을 하기 때문에 안전하지 않은 채널로 통신을 한다고 본다. 따라서 공격자에 의한 정보유출 가능성과 태그를 이용하는 사용자의 프라이버시 노출 문제에 대한 가능성을 가지고 있다. 뿐만 아니라 태그 하드웨어는 저가의 하드웨어이기 때문에 물리적인 공격(tampering)에 약하다. 결국 이러한 이유들로 말미암아 태그가 가지는 정보에 대한 보호 및 사용자의 프라이버시 보호를 위한 암호 프로토콜의 필요성이 대두 되었다[1]. 기존에 사용되고 있는 이동 통신 환경을 위한 그룹키 사용 기법을 RFID 시스템에 적용하여 이와 같은 문제를 해결할 수도 있겠지만[2-6], RFID 태그의 하드웨어적인 제약 사항은 이러한 해결방법을 적용하기 힘들게 한다.

일반적으로 RFID 시스템 환경에서는 다음과 같은 사항을 가정한다.

- 태그는 저가의 장치이고, 외부로부터 물리적인 공격을 받을 수 있다.
- 태그와 리더 간의 통신은 무선 통신으로 도청되어질 수 있다.
- 리더와 백엔드 서버 사이의 통신은 안전한 채널을 통해서 이루어진다.
- 백엔드 서버는 신뢰할 수 있는 개체로 본다.

이러한 가정 하에서 RFID 시스템은 다음과 같은 보안 문제가 발생하게 되며, 이를 해결하기 위한 암호 프로토콜에 대한 연구가 많이 진행되고 있는 중이다[7-14].

1.1 RFID 시스템에서 정보 노출 문제

현재 쓰이고 있는 RFID 시스템은 태그마다 고유의 *ID*를 가지고 있다. 태그의 *ID*는 리더의 요청이 있을 때마다 무선 통신을 이용하여 리더에게 전송된다. 태그에는 고유 *ID* 이외에도 제품명, 제품의 생산일, 회사명 등을 저장해 놓을 수 있다. 리더는 제품에 부착된 태그와의 통신을 통하여 제품의 정보를 알 수 있게 된다[1,7,8].

여기서 우리는 태그는 리더가 합법적인 리더인지 아닌지를 구분할 수 있는 능력을 가지고 있지 않다는

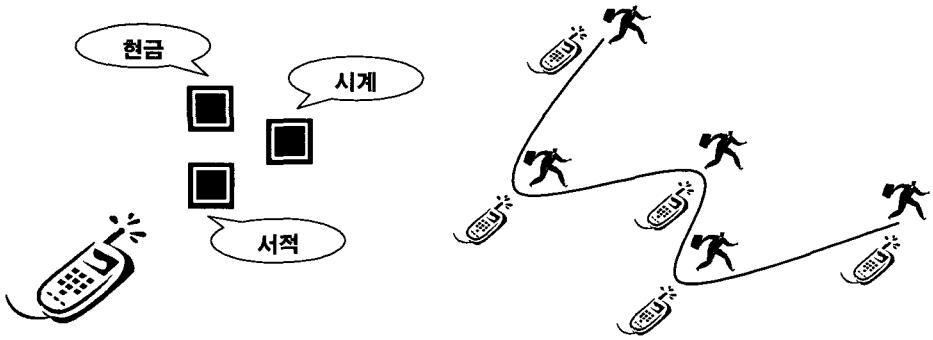


그림 2. RFID 시스템에서의 태그 정보 노출 문제(좌)와 위치 정보 노출 문제(우)

점에 주목해야 한다. 결국 비합법적인 리더가 정보를 요청할지라도 태그는 자신의 정보를 모두 알려줘야 한다는 문제가 발생한다[7-14]. 예를 들어서 A라는 사람이 태그가 부착 된 여러 가지 제품을 소지하고 있는 경우, RFID 리더를 가지고 있는 사람이라면 누구라도 A가 가지고 있는 제품의 정보를 읽을 수 있다는 것이다. 특히, 민감한 제품에 대한 정보가 리더에게 읽히게 된다면, 사용자의 프라이버시 노출이 심각해질 수 있다[7-9](그림2).

1.2 사용자 위치 정보 노출 문제

위치 정보의 노출 문제는 각각의 태그가 항상 동일한 값을 리더에게 전송하는데서 오는 문제이다. 태그는 고유의 *ID*를 동일하게 전송하기 때문에 리더를 소유한 공격자(추적자)는 특정 태그 소유자의 위치를 추적(location tracking)할 수 있다. 리더를 가진 공격자는 수 미터에서 수십 미터 거리를 유지하며 특정 태그의 소유자를 추적해 갈 수 있다(그림2). 공격에 가담하는 사람이 여러 명인 경우이거나 다수의 위치에 리더를 설치한 경우에는 추적이 더욱 용이해질 수 있다. 예를 들어서 특정 국가 기관이 도시 곳곳에 리더를 설치해 두고 특정 인물의 위치를 파악하는 경우처럼 매우 심각한 프라이버시 침해 문제로 공론화 될 수도 있다. 태그를 소유한 사용자의 입장에서 볼 때 자기가 방문한 곳의 정보가 타인에게 드러나면 난처해지는 경우가 있을 것이다. 그러한 경우에도 태그에 대한 추적으로 말미암아 소유자의 위치 정보가 노출될 수 있기 때문에 사용자의 위치 정보에 대한 프라이버시 침해를 가져온다. 결국 위치 추적은 정보 유출 문제만큼이나 심각한 프라이버시 침해 문제로 볼 수 있다[7-11].

1.3 사용자 프라이버시 보호를 위한 필수 보안 요건

현재까지의 연구들을 종합하여 볼 때, RFID 시스템에서 사용자 프라이버시를 보호하기 위해서는 다음과 같은 3가지 필수 보안 요건을 만족하는 프라이버시 보호 프로토콜이 설계되어야 한다[7-14].

- 기밀성(confidentiality): 모든 통신에서 어떠한 정보든지 외부에 노출되지 않고 비밀리에 전송되어야 된다는 조건. 이 조건이 만족되지 않는 경우에는 위에서 언급한 정보 유출 문제를 해결할 수 없다.
- 불구분성(indistinguishability): 태그가 리더에게 정보를 전송할 때 리더의 요청이 있을 때 마다 매번 똑같은 값을 전송하지 않아야 한다는 조건. 이 조건이 만족되지 않는 경우에는 위치 정보 노출 문제를 해결할 수 없다.
- 전방보안성(froward security): 태그의 비밀 값이 공격자에 의해 노출이 되었더라도 비밀 값을 이용하여 이전의 비밀 값을 알아낼 수 없어야 한다는 조건. 이 조건이 만족되지 않는 경우에는 공격받은 태그의 과거 위치 정보(location history)가 드러나게 되는 문제가 존재한다.

현재까지 제안된 프로토콜 중에서 위의 3가지 필수 보안 요건을 만족하는 안전한 프로토콜은 Ohkubo가 제안한 해시 체인 기반 프로토콜[13,14]이라고 볼 수 있다[7]. 본 논문에서는 Ohkubo의 프로토콜에 대해서 살펴보고, Ohkubo 프로토콜의 단점인 백엔드 서버에서의 효율성 측면을 향상 시킬 수 있는 기법을

설명한다. 결국 본 논문에서는 Ohkubo 프로토콜의 안전성은 그대로 유지하면서, 백엔드 서버의 태그 판별 시간을 절감할 수 있는 기법을 제안한다.

II. RFID 시스템 정의 및 관련 연구

2.1 RFID 시스템의 구성 요소

2.1.1 태그 (Tag)

RFID 태그는 주위의 사물 등에 태그를 부착하여 사물의 *ID*정보 및 주변 환경 정보를 인식하여 각 사물의 정보를 수집, 저장, 가공 및 추적함으로써 사물에 대한 원격처리, 관리 정보교환 등 다양한 서비스를 제공한다. RFID 태그는 칩(IC), 안테나 및 패키징으로 구성되고(그림3), 칩에는 고유의 *ID*정보를 저장하며 리더의 질의가 오면 상황에 따라 외부에 자신의 *ID*정보를 전송하거나 수신하는 역할을 한다. RFID 태그는 배터리를 사용하는 능동형 태그와 배터리를 사용하지 않은 수동형 태그로 구분된다. 능동형 태그는 자체 배터리를 가지고 있어 먼 거리까지 송신이 가능한 반면, 수동형 태그는 자체 배터리를 가지고 있지 않으며 태그를 동작할 수 있는 에너지를 리더로부터 공급 받는다. 태그와 리더의 접속을 위하여 사용되는 방식은 크게 상호 유도 방식과 전자기파 방식이 있다 상호 유도 방식은 코일 안테나를 이용한다. 리더의 안테나 코일은 주변지역에 강한 자기장을 발생하여 방출된 자기장의 일부분이 근접한 태그의 코일 안테나에 유도성 전압을 발생, 정류된 후 태그 IC 칩에 에너지를 공급한다. 전자기파 방식은 고주파 안테나를 이용해서 서로 무선 접속을 수행한다. 이러한 원리에 따라 상호 유도 방식의 태그는 거의 수동형으로 작동되며, 전자기파 방식의 태그는 IC 칩을 구동하기 위한 충분한 전력을 리더로부터 공급받지 못하므로 능동형으로 구현된다[7-11].

2.1.2 리더 (Reader)

RFID 리더는 수동형 태그가 동작할 수 있는 전력과 명령어를 무선 신호로 태그에 전송하고 태그로부터 응답을 수신하여 신호를 복원하는 기능을 수행한다. RFID 리더 시스템은 RF/아날로그부와 디지털 신호 처리 제어부로 구성되어 있다. 리더의 요소기술 중의 하나인 충돌 방지 기술은 하나의 리더에서 여러 개의 태그를 동시에 인식할 수 있도록 하기 위한 기법이다. 현재 RFID 시스템에서는 이진 탐색(tree-walking) 방식과 슬롯 알로하(slotted ALOHA) 방식을 기반으로 변형된 방법들이 널리 쓰이고 있다[7-10].

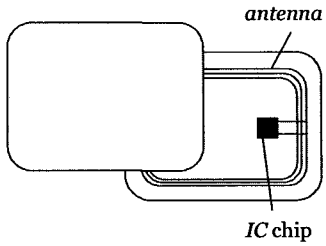


그림 3. RFID 태그의 일반적인 구조
(비접촉식 스마트 카드형 태그)

2.1.3 백엔드 서버 (Back-end Server)

리더에 의해 읽혀진 태그 및 센서의 데이터를 유무선 네트워크를 거쳐 데이터 처리를 담당하는 호스트 컴퓨터의 미들웨어로 전달하는데 이를 일반적으로 백엔드 서버라 불린다. 이러한 태그의 데이터는 객체에 대한 정적 데이터, 이력 데이터 및 실시간 데이터를 포함할 수 있기 때문에 애플리케이션은 이를 활용한 상황인식 기반의 지능화된 서비스를 제공할 수 있게 된다. 일반적으로 리더-백엔드 서버 간의 통신은 리더와 백엔드 서버 간의 상호 인증을 거친 후에 만들어지는 안전한 채널(secure channel)을 통해서 이루어진다고 가정한다[7-12].

2.2 용어 정리

이 절에서는 관련 연구와 제안 기법에 사용하게 되는 대부분의 기호와 용어를 정의하도록 한다[7].

- m : 태그들의 개수.
- n : 해시 체인의 최대 길이. 태그가 읽혀질 수 있는 최대 횟수.
- ID_t : 태그 T_t의 식별 정보. $t=1,2,\cdots,m$.
- $s_{t,1}$: 태그 T_t의 해시 시드 값.
- T_t : 태그 t. T_t는 해시 함수H와 G를 가지고 있으며, 고유의 해시 시드 값 $s_{t,1}$을 가지고 있다.
- R : 리더. R은 백엔드 서버와는 안전한(secure) 채널을 통해서 통신한다고 가정한다.
- B : 백엔드 서버. B는 모든 태그에 대한

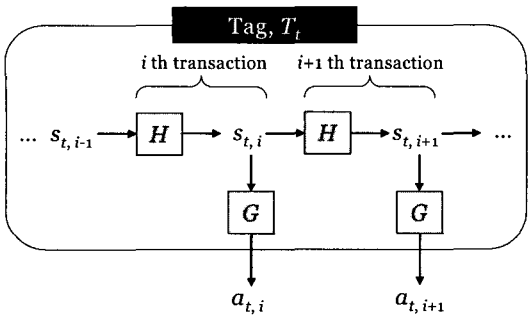


그림 4. 해시 체인 기반 기법에서 태그의 구조

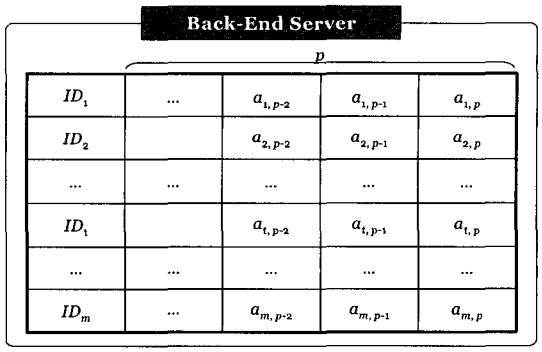


그림 5. 백엔드 서버 메모리에 저장되는 선행 계산 값들

$(ID_t, s_{t,1})$의 값에 대한 데이터베이스를 관리한다.

H : $\{0,1\}^* \rightarrow \{0,1\}^\ell$, 출력값의 길이가 ℓ인 일방향 해시 함수.

G : $\{0,1\}^* \rightarrow \{0,1\}^\ell$, 출력값의 길이가 ℓ인 일방향 해시 함수. H함수와는 서로 다른 분산(distribution)을 가진다.

2.3 관련 연구

Ohkubo가 제안한 프로토콜[13,14]에 대해서 살펴본다. 이 프로토콜에서는 태그가 출력 특성이 서로 다른 두 개의 일방향 해시 함수(one-way hash function)를 포함하는 것으로 가정한다. 일방향 해시 함수의 특징인 일방향성이 기밀성과 불구분성, 전방보안성을 보장해주게 되도록 설계되었다.

그림 4는 Ohkubo가 제안한 프로토콜을 간단하게 도식화 한 것이다. 프로토콜 수행 전 백엔드 서버는 모든 태그에 대한 고유의 시드 값과 ID를 데이터베이스에 저장하고, 각 태그는 해당 시드 값을 비밀 값으로 저장한다. 리더가 태그에게 요청할 수 있는 횟수 즉, 해시 체인의 최대 길이는 n으로 제한한다. 리더의 요청이 올 경우 태그는 자신의 비밀 값 $s_{t,i}$를 G해시 함수에 입력하고 $a_{t,i}$를 계산하여 리더에게 보내준 다음 다시 H해시 함수에 입력하여 $s_{t,i+1}$로 갱신한 후 태그 내의 메모리에 저장한다. 리더는 태그로부터 수신한 값 $a_{t,i}$를 그대로 백엔드 서버에 전달하고, 백엔드 서버는 리더에게서 받은 해시 연산 값으로 태그를 판별한다. 백엔드 서버는 모든 태그에 대한 시드 값 $s_{t,1}$, $(1 \le t \le m)$과 모든 해시 체인$(1 \le i \le n)$에 대해 $a'_{t,i} = G(H^{i-1}(s_{t,1}))$을 수행하여 리더로 부터 수신한 $a_{t,i}$와 일치하는 계산 결과 $a'_{t,i}$를 내는 시드 값 $s_{t,i}$를 찾는다. $s_{t,i}$를 찾게 되면 해당 ID_t를 찾을 수 있게 되고 ID_t는 안전한 채널로 리더에게 전송하면서 프로토콜을 종료하게 된다. 따라서 이 프로토콜은 백엔드 서버가 저장하고 있는 모든 태그의 시드 값을 이용해서 평균적으로 $\frac{mn}{2}$번의 해시 계산을 수행하여 몇 번째 태그의 해시 체인에 속했는지를 판단하는 것이다.

이 프로토콜은 태그가 저장하고 있는 비밀 값 $s_{t,i}$를 스스로 갱신함으로써 필수 보안 요건 중 기밀성과 불구분성을 만족시키게 된다. 이는 일방향 해시 함수의 특성을 근거로 하고 있다. 기존의 다른 연구에서는 리더를 인증하기 위한 과정을 거치기도 하지만, 이 프로토콜에서는 태그가 리더를 인증하는 과정을 거치지 않는다. 어떠한 리더라도 태그의 응답을 들을 수 있다. 하지만, 그것을 가지고 태그의 ID를 알아내기 위해서는 정상적으로 백엔드서버에 접근할 수 있어야만 한다. 결국 프로토콜은 단순해지고 리더를 따로 인증할 필요가 없게 된다. 또한 외부의 물리적인 공격에 의해 태그의 비밀 값 $s_{t,i}$가 드러났다 하더라도 일방향 해시 함수의 특성상 현재의 비밀 값 $s_{t,i}$를 이용하여 이전의 비밀 값들을 알아내기는 일방향 해시 함수의 특성상 매우 어렵기 때문에 전방보안성도 보장하게 된다. 따라서 이 기법은 기존의 연구 가운데 가장 안전한 프로토콜이라고 볼 수 있다.

Ohkubo의 프로토콜은 안전성에서 뛰어난 프로토콜이지만, 중요한 단점을 가지고 있다. 그것은 백엔드 서버가 태그를 판별하는 시간이 너무 오래 걸린다는 점이다. 백엔드 서버는 하나의 태그를 판별하기 위해서 모든 태그$(1 \le t \le m)$의 해시 체인$(1 \le i \le n)$에 대해서 $a'_{t,i} = G(H^{i-1}(s_{t,1}))$이라는 해시 계산을 수행해

표 1. 선행 계산 알고리즘의 요약

[Step 1]	백엔드 서버는 모든 태그의 시드 값에 대해 p번 트랜잭션만큼 해시 연산을 수행하여 선행 계산 집합(테이블) T_{pc}를 만든다. $T_{pc}=\{(a_{i,j},i,j)\mid a_{i,j}=G(H^{j-1}(s_{i,1})),\ \text{for } 1\le i\le m,\ 1\le j\le p\}$
[Step 2]	퀵 정렬 알고리즘을 사용하여 T_{pc}를 $a_{i,j}$값을 기준으로 오름차순 정렬하여 백엔드 서버의 주기억 장치에 저장한다.
[Step 3]	$p_i \leftarrow 1,\ \text{for } 1\le i\le m$

표 2. 백엔드 서버의 태그 식별 알고리즘의 요약

Input :	태그의 응답 값 a
Output :	응답 값 a에 해당하는 태그의 ID
[Step 1]	$a=a_{i,j}$의 조건을 만족하는 원소 $(a_{i,j},i,j)\in T_{pc}$ 가 존재하는지를 이진 검색을 이용하여 찾는다. $(a,i,j)\notin T_{pc}$ 이면, [Step 2]로 간다. $(a,i,j)\in T_{pc}$ 이면, $t\leftarrow i$, $c\leftarrow j$한 후 [Step 3]으로 간다.
[Step 2]	$a=G(H^{j-1}(a_{i,1}))$의 조건을 만족하는 원소를 $1\le i\le m$, $1\le j\le n$ 범위에서 찾는다(Ohkubo의 기존 방법과 동일). 조건을 만족하는 i, j가 없는 경우, $ID\leftarrow NULL$후 종료한다. 조건을 만족하는 i, j가 있는 경우, $t\leftarrow i$, $c\leftarrow j$.
[Step 3]	$ID\leftarrow ID_t$
[Step 4]	$p_t\le k\le p_t+p$ 범위에 속하는 $(a_{t,k},t,k)$원소들을 T_{pc}에서 삭제한다.
[Step 5]	$p_t\leftarrow c$
[Step 6]	새로운 집합 T_{Add}를 다음과 같이 만든다. $T_{Add}=\{(a_{t,l},t,l)\mid a_{t,l}=G(H^{l-1}(s_{t,1})),\ \text{for } p_t\le l\le p_t+p\}$
[Step 7]	퀵 정렬 알고리즘을 사용하여 T_{Add}를 $a_{t,l}$을 기준으로 오름차순으로 정렬한다.
[Step 8]	T_{pc}와 T_{Add}를 병합 정렬한 후 종료한다.

야 한다. 평균적으로 $\frac{mn}{2}$번의 해시 체인 연산을 수행해야 하는 것이다. 이는 $O(mn)$의 계산 복잡도로 표현할 수 있다. 결국 태그의 수와 해시 체인의 길이에 따라 백엔드 서버가 하나의 태그를 판별하는 시간이 비례해서 길어지게 되는 것이다. 간단한 예로서, 태그의 개수가 2^{20}개이고, 해시 체인의 최대 길이가 2^{10}으로 제한되어 있는 경우에도 백엔드 서버가 하나의 태그를 판별하는 데에는 평균적으로 약 64초가 소요된다[7,13,14]. 결과적으로 Ohkubo의 프로토콜은 안전성의 측면에서는 매우 우수하지만 확장성(scalability)은 약하다고 볼 수 있다.

Ⅲ. 백엔드 서버의 태그 판별 시간 절감 기법

이 장에서는 Ohkubo의 RFID 프라이버시 보호

프로토콜의 백엔드 서버 계산 시간을 절감하기 위한 새로운 기법을 소개한다. 일차적으로 핵심적인 절감 기법을 소개하고, 그 다음으로 추가 기법을 소개한다. 핵심 기법을 통해서 Ohkubo 프로토콜의 기존 백엔드 서버 계산 시간보다 훨씬 빨라진 태그 판별 속도를 낼 수 있게 된다. 추가 기법에서는 핵심 기법에서 발생할 수 있는 문제점과 그 해결 방안에 대해서 설명한다.

3.1 핵심 기법

핵심 기법에서의 태그와 리더 사이의 통신은 기존의 Ohkubo의 프로토콜[13,14]을 그대로 사용한다. 그러나 원래의 Ohkubo 프로토콜에서 사용되는 백엔드 서버의 계산 방법은 시간이 매우 오래 걸리기 때문에, 이를 개선하는 새로운 선행 계산 기법에 대해서 설명한다.

먼저 백엔드 서버는 그림 5의 가장 왼쪽 열에 표현된 것과 같이 메모리 내에 m개의 태그에 대한 각각의 ID를 저장하고 있다. 그리고 각각의 태그에 해당하는 해시 체인 값들을 $a_{t,i} = G(H^{i-1}(s_{t,i}))$의 계산을 통해서 p개씩 미리 계산하여 백엔드 서버의 메모리에 저장해 놓고 있다(그림 5). 즉 백엔드 서버의 메모리에는 $m \times p$개만큼의 결과값을 미리 계산하여 저장하고 있는 것이다($i \le p \le n$). 여기서 $m \times p$를 선행 계산 테이블이라 부르고, 각각의 $a_{t,i}$값은 $(a_{t,i}, t, i)$와 같은 형태의 자료구조를 저장하여, $a_{t,i}$값이 t번째 태그의 i번째 해시 체인 연산 값으로 판단할 수 있게 한다. 그 후 $a_{t,i}$을 기준으로 오름차순으로 정렬하여 저장한다(표 1). 선행 계산 테이블을 최초로 만드는 과정에서는 퀵 정렬(quick sort)을 이용하게 되고, 이후 태그의 판별이 이루어지는 과정에서 선행 계산 테이블을 갱신하게 될 때에는 병합 정렬(merge sort)을 이용하게 된다.

리더가 임의의 값 a를 전달해 주면, 백엔드 서버가 이진 검색(binary search)을 통해 리더에게서 받은 a값을 선행 계산 테이블에서 찾는 과정을 거친다. 만약 선행 계산 테이블에서 일치하는 해시 체인 연산 값을 찾게 되면 추가로 저장된 t와 i값을 통해서, 해시 결과 값이 t번째의 태그에 i번째의 해시 체인이라는 것을 알게 되는 것이다(표 2의 Step 1). 그 후 백엔드 서버는 해당 태그의 ID_t를 리더에게 전송하게 된다.

그러나 만약 선행 계산 테이블에서 a와 일치하는 값을 찾지 못하였을 경우에는 원래의 Ohkubo 프로토콜의 방법과 마찬가지로 $m \times (n-p)$개의 해시 연산을 모두 해서, 해당 태그를 찾는 과정을 거치면 된다(표 2의 Step 2). 이러한 경우에는 태그 판별에 걸리는 시간은 결론적으로 {(선행 계산 테이블 검색 시간) + ($m \times (n-p)$개의 해시 체인 계산 시간)}이 되지만, 원래의 Ohkubo 프로토콜 방법으로 계산했을 때 걸리는 시간 {$m \times n$개의 해시 체인 계산 시간}보다는 짧다고 볼 수 있다.

선행 계산 테이블에서 a와 일치하는 값을 찾지 못하는 경우로는 대표적으로 다음의 두 가지 상황을 생각해볼 수 있다. 첫 번째 경우는 가장 일반적인 경우로서 불법적인 리더에 의해 태그가 $p+1$번 이상 읽힌 경우이다. 불법적인 리더는 특정 태그로부터 의미 있는 정보를 뽑아내거나 특정 태그를 추적하여 사용자의 위치를 추적하려는 목표를 가지고 계속해서 태그를 읽으려고 할 것이다. 이 경우에는 백엔드 서버와의 인증된 정보교환 과정 없이 태그 내부의 값만 계

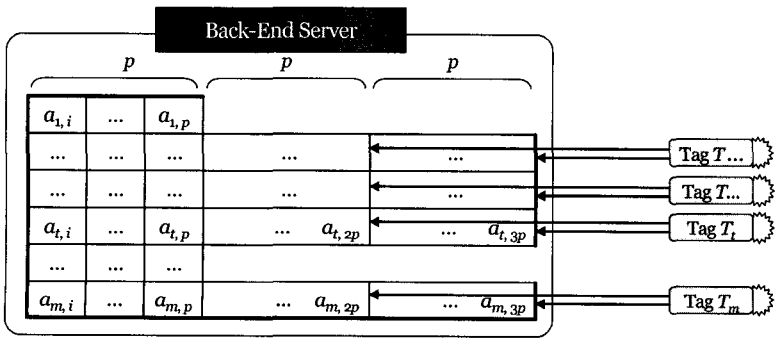


그림 6. 백엔드 서버에서의 메모리 문제

속해서 변경된다. 그렇게 되면 나중에 정상적인 리더에 의해서 읽혔을 경우에 백엔드 서버의 선행계산 테이블을 통해서 찾을 수 없는 결과가 발생할 수 있는 것이다. 두 번째 경우는 정상적인 리더에 의해서 읽혔지만 백엔드 서버와의 정보 교환을 하지 않고서 $p+1$번 이상 읽은 경우이다. 이런 경우는 드문 경우로서, 정상적인 리더가 읽기 시도를 하지만 태그나 리더 하드웨어의 장애 또는 통신 환경(air interface)상의 장애가 일정기간 계속됨으로 인해서, 태그는 읽힌 것으로 생각하고 자기 내부의 값을 계속 변경하지만 서버에는 이러한 상태가 반영이 되지 않은 경우이다. 이러한 두 가지의 경우를 포함해서 선행계산 테이블에서 a와 일치하는 값을 찾지 못하는 경우를 모두 "서버 동기화 문제"라고 정의한다. 물론 백엔드 서버의 메모리 상황에 따라서, p값을 충분히 여유 있게 설정해 준다면, 서버 동기화 문제를 줄일 수 있을 것이다.

선행 계산 테이블에서 찾은 경우와 Ohkubo 프로토콜의 원래 방법대로 찾은 경우에 모두 선행 계산 테이블은 갱신해 주어야 한다. 찾은 결과를 통해서 t값과 i값을 알 수 있으므로, 태그 t에 대한 선행 계산을 i번째부터 p번만큼 다시 한 후, 선행 계산 테이블에 병합정렬 시킨다(표 2의 Step 4~8).

3.2 추가적인 기법 (지속적으로 동기화 문제가 발생하는 태그에 대한 메모리 추가 할당 기법)

앞서 언급했던 바와 같이 서버 동기화 문제가 발생한 경우에는 {(선행 계산 테이블 검색 시간) + $(m\times(n-p)$개의 해시 체인 계산 시간)}이 걸리게 된다. 그러나 만약 하나의 태그가 긴 시간 동안 계속해서 서버 동기화 문제를 발생시킬 경우에는 제안한 서버 계산 시간 절감 기법의 효과가 절감될 수 있다.

이를 보완하기 위해서 자주 동기화 문제가 발생하는 태그에 대해서는 선행 계산 해시 값의 크기 p값을 더 늘려주는 방법을 사용할 수 있다. 각각의 태그에 대해서 선행 계산을 해두는 해시값의 개수를 p_t $(1\le t\le m)$라고 정의한다면, 초기에는 모든 p_t을 값을 p로 설정해 놓는다. 그 후 태그 t에서 서버 동기화 문제가 한 번 발생할 때마다, p_t의 크기를 증가시켜줌으로써, 서버 동기화 문제가 발생할 확률을 낮추어 줄 수 있다. 예를 들어, 서버 동기화 문제가 발생할 때마다 $p_t=p_t+p$의 형태로 p_t의 크기를 증가시켜줄 수 있을 것이다.

하지만 그림 6에서처럼 백엔드 서버의 메모리가 제한적이기 때문에, 동기화 문제가 발생하는 태그에 대해서 무작정 계속해서 메모리 공간을 추가로 할당할 수만은 없다. 이에 대한 부분적인 보완책으로서 다음과 같은 방법이 있을 수 있다. 계속해서 서버 동기화 문제가 발생하는 태그에 대해서는 p_t의 값을 크게 해주고, 서버 동기화 문제가 발생하지 않는 태그에 대해서는 p_t의 크기를 상대적으로 줄여주는 것이다. 예를 들어, 서버 동기화 문제가 발생하는 태그 t에 대해서는 $p_t=p_t+p$의 형태로 p_t의 크기를 증가시켜주고, 대신 서버 동기화 문제가 없이 정상적으로 읽히는 태그 s에 대해서는 $p_s=p_s\ /\ 2$의 형태로 p_s의 크기를 감소시켜줄 수 있을 것이다.

그러나 여기에서 p_t값을 얼마씩 증가시키고 감소시켜야 하는가는 서버의 메모리 보유량과 태그의 총 개수, 그리고 동기화 문제가 있는 태그의 수 등에 영향을 받는 요소이기 때문에, 위에서 언급한 예와 같이 직관적으로 정해주기 보다는 사전 시뮬레이션을 통해서 상황에 맞는 적절한 값으로 정해주는 것이 올바르다고 본다. 궁극적으로 이러한 메모리 관리 기법을 이용한다면 백엔드 서버에서 선행 계산 테이블이 차지하는 메모리를 효율적으로 관리할 수 있으리라고 본다.

Ⅳ. 시뮬레이션 및 분석

이 장에서는 제안 기법과 원래의 Ohkubo프로토콜의 방법을 실제 프로그램으로 구현하고, 이를 이용

표 3. 시뮬레이션을 위한 하드웨어 및 소프트웨어 환경

구성 요소	하드웨어	소프트웨어
태그 클라이언트	Pentium4 3.2GHz 2GB Memory	Microsoft Visual C++ 6.0 Microsoft Windows XP Crypto++ Library 5.2.1[15]
백엔드 서버		

해서 여러 가지 시뮬레이션 환경에서의 성능을 비교 분석한 결과를 보여준다.

4.1 시뮬레이션

4.1.1 시뮬레이션 환경

제안 기법의 성능 향상 정도를 평가하기 위해 Ohkubo 프로토콜의 방법[13,14]과 비교하여, CPU처리시간, 메모리 소비량, 적중률(hit ratio)에 따른 처리시간에 초점을 맞추어 실험을 수행하였다. 여기서 적중률은 임의의 태그의 출력값이 백엔드 서버의 선행 계산 테이블에 속해 있는 정도를 표현한 것이다. 적중률이 100%라는 것은 서버 동기화 문제가 전혀 발생하지 않은 상태를 의미하게 된다. 반면에 적중률이 70%라는 것은 백엔드 서버에 들어오는 태그 판별 요청 중에서 70%는 선행 계산 테이블 검색만으로 판별이 가능하고 나머지 30%는 원래의 Ohkubo 방법을 사용해서 판별을 해야 한다는 것을 의미한다.

표 3은 시뮬레이션에 사용된 소프트웨어와 하드웨어 환경을 설명한 것이다. 태그 클라이언트와 백엔드 서버 시스템은 같은 하드웨어를 사용하였고, 운영체제는 MS Windows XP를 사용하였다. 해시 연산에 이용되는 일방향 해시 함수는 공개 암호화 패키지 Crypto++에서 제공하는 160비트 해시 함수 SHA-1을 사용하였다[15]. 백엔드 서버에 필요한 정렬과 검색을 위해서는 퀵 정렬(quick sort), 병합 정렬(merge sort)과 이진 검색(binary search)을 이용하였다. 정렬 알고리즘과 검색 퀵 정렬과 이진 검색은 Visual C++에 포함되어 있는 표준 라이브러리 함수를 사용하였고, 병합 정렬은 직접 프로그래밍 하였다.

본 논문에서는 해시 시드 값 160비트(20바이트)와 추가적인 정보 데이터 2바이트를 갖는 샘플 데이터를 임의로 생성하여 시뮬레이션을 수행하였다. 전체 태그의 개수 m은 각각 10^6, 10^5, 10^4개씩으로 하였고, 최대 해시 체인의 길이 n은 각각 10^3, 10^2, 10^1으로 하였다. 또한 선행 계산에 필요한 p의 길이는 한정된 메모리 양에 따라 각각 다르게 주었다. 표 4는 실험에 필요한 태그의 수와 최대 해시 체인 길이, 선행 계산에 필요한 해시 체인 길이, 사용된 메모리 양을 나타낸 것이다. 각 조건에 해당하는 시뮬레이션은 100번씩 수행하여 평균 계산 시간을 산출하였다.

4.1.2 시뮬레이션 프로그램

그림 7~9에서 보는 바와 같이 시뮬레이션 프로그램은 태그 클라이언트(Tag Client)와 백엔드 서버(Back-End Server)의 두 가지 부분으로 구성된다. 그리고 백 엔드 서버 프로그램은 Ohkubo 프로토콜의 원래 방법대로 작동하는 프로그램과 제안 기

표 4. 시뮬레이션에 사용된 태그 데이터

비교 요소 / 비교 기법	적중률	태그 개수 (m)	최대 해시 체인 길이(n)	선행 계산 해시 체인 길이(p)	실제 사용된 메모리 양
Ohkubo 프로토콜의 백엔드 서버의 계산 방법	-	10^6	1×10^3 2×10^3 3×10^3	-	28MB
		10^5			2.8MB
		10^4			0.28MB
제안 기법	100% 90% 80% 70%	10^6	1×10^3	10	280MB + 28MB
			2×10^3	20	560MB + 2.8MB
			3×10^3	30	840MB + 0.28MB
		10^5	1×10^3	100	280MB + 28MB
			2×10^3	200	560MB + 2.8MB
			3×10^3	300	840MB + 0.28MB
	100%	10^4	1×10^3	1000	280MB + 28MB
			2×10^3	2000	560MB + 2.8MB
			3×10^3	3000	840MB + 0.28MB

법대로 작동하는 프로그램을 따로 구현하였다. 리더는 단순히 태그와 백엔드 서버 사이의 매개체 역할만 하기 때문에 태그 클라이언트 쪽으로 포함시켜서 구현하였다.

태그 클라이언트와 백엔드 서버 사이의 시뮬레이션 수행 과정은 다음과 같다. 먼저 백엔드 서버에서 전체 태그의 수 m과 최대 해시 체인의 길이 n를 입력 받게 된다. 제안 기법을 위한 서버 프로그램에서는 추가적으로 선행 계산을 위해 해시 체인의 길이 p를 입력 받게 된다. 그 후 태그 시드 값들을 생성하게 되고, p범위만큼 선행 계산 테이블을 생성한 다음 서버 소켓 서비스를 실행한다. 이 작업을 모두 마치게 되면 생성된 시드 값을 가지고 태그 클라이언트에서 원하는 t번째 태그의 해시 체인 길이만큼 해시 체인 연산을 수행하여 결과 값을 출력한다. 이후 태그 클라이언트와 백엔드 서버가 연결이 되면 태그 클라이언트에서 계산된 결과 값 a를 백엔드 서버에 보내고, 백엔드 서버는 선행 계산된 해시 체인 테이블에서 일치하는 해시 값 $a_{t,i}$을 찾고 그에 맞는 태그의 ID_t를 태그 클라이언트에 전송하게 된다. 적중률을 적용해서 Ohkubo 프로토콜의 원래 계산 기법과 제안 기법의 계산 시간을 비교해야 하기 때문에 적중률을 입력할 수 있으며, 반복적인 시뮬레이션을 실행할 수 있도록 태그 클라이언트를 구현했다(그림 7).

4.2 성능 분석

이 절에서는 시뮬레이션의 결과를 정리하여 Ohkubo 프로토콜의 원래 계산 방법과 제안 기법의 성능을 비교분석한다. 시뮬레이션에서 제안 기법은 적중률을 100%~70%로 나누어서 실험하였다. 현재 일반적인 서버 한대를 기준으로 장착할 수 있는 최대 메모리 용량은 4GB 정도이지만, 시뮬레이션의 수행 환경의 특성상, 선행 계산에 필요한 메모리 양을 1GB 이하로 제한하였다. 제안 기법의 선행 계산에 필요한 시간은 표 5에서 볼 수 있으며, 시뮬레이션의 결과는 그림 10과 표 6에서 확인할 수 있다. 여기서 평균 선행 계산 시간은 초(second) 단위로 측정하였고, 백엔드 서버의 평균 태그 판별 시간은 마이크로 초(microsecond) 단위로 측정하였다.

그림 10(a)와 표 6에 표시한 바와 같이 제안 기법의 적중률이 100%일 때에는 백엔드 서버의 평균 태그 판별 시간은 Ohkubo 프로토콜의 원래 방법보다 현저하게 줄어든 것을 볼 수가 있다. 이는 선행 계산 테이블의 해시 결과 값과 태그의 해시 체인 결과 값이 모두 일치하기 때문에 100%의 적중률에서는 당연한 결과로 볼 수 있다. 그림 10(b)는 90%의 적중률일 때 Ohkubo 프로토콜의 원래 방법과 제안 기법의 백엔드 서버의 계산시간을 비교한 것이다. 그래프에서도 볼 수 있듯이, 90%의 적중률을 갖는 제안 기법에서 선행 계산 테이블이 10번의 갱신을 했음에도 불구하고도 빠른 계산 시간을 보였다. 그림 10(c)는 적중률이 80%일 때 계산 시간을 나타낸 것이다. 전체적으로 90%의 적중률에 서 보인 계산 결과보다 다소

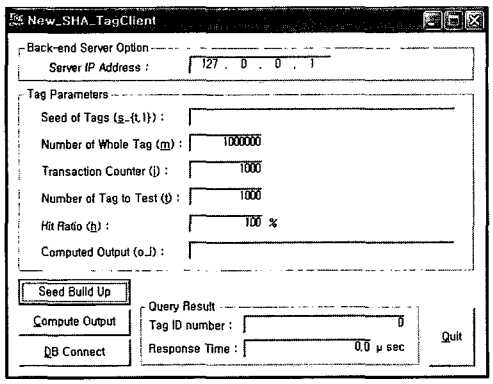


그림 7. 태그 클라이언트 프로그램의 실행 화면

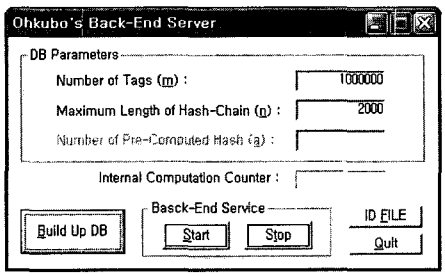


그림 8. Ohkubo의 기법의 백엔드 서버 프로그램의 실행 화면

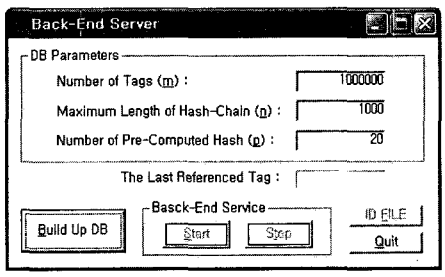
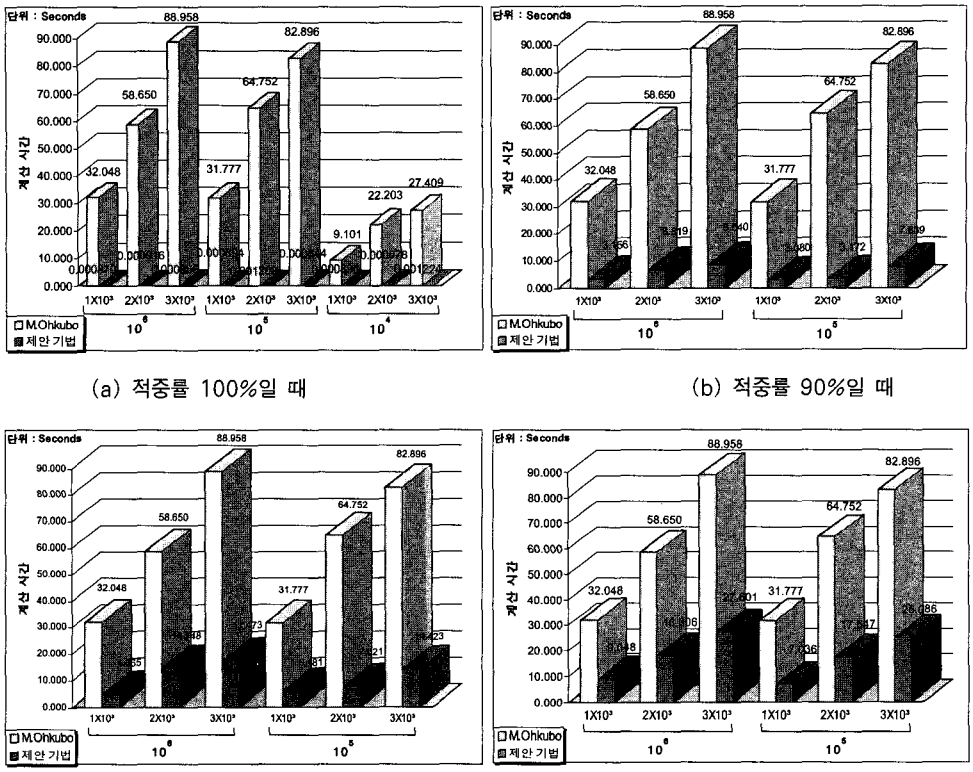


그림 9. 제안 기법의 백엔드 서버 프로그램의 실행 화면



(a) 적중률 100%일 때 (b) 적중률 90%일 때

(c) 적중률 80%일 때 (d) 적중률 70%일 때

그림 10. 백엔드 서버의 평균 계산 시간 비교 그래프

떨어지기는 했지만 Ohkubo 프로토콜의 원래 방법의 평균 계산 시간 보다 여전히 빠른 속도를 보이고 있다. 그림 10(d)은 적중률이 70%일 때 제안 기법의 평균 계산 시간을 나타낸 것이다. 여기서도 볼 수 있듯 이전의 높은 적중률에 비해 평균 계산시간이 떨어지기는 했지만 여전히 Ohkubo 프로토콜의 원래 방법보다 빠른 속도를 보이고 있다.

지금까지 시뮬레이션 결과로 볼 때 적중률에 따라서 제안 기법의 계산 시간은 일정한 차이가 난다. 적중률이 90%일 때는 Ohkubo 프로토콜의 원래 방법의 $\frac{1}{10}$의 속도로, 80%일 때는 $\frac{2}{10}$의 속도로, 70%일 때 $\frac{3}{10}$의 속도로 태그 판별을 하고 있다. 즉, 제안 기법을 사용하더라도 동기화 문제가 발생하는 태그의 수가 많아질수록 백엔드 서버에서 태그를 판별하는 시간은 점차 늘어나게 된다. 그러나 최악의 경우 모든 태그가 동기화 문제를 가지게 되어 태그의 해시 체인이 늘어나게 된다 하더라도 Ohkubo 프로토콜의 원래 방법보다는 항상 빠른 태그 판별 시간을 보장할 수 있다. 그렇지만, 일반적인 사용 환경에서 충분한 크기의 p값을 할당해 준다면, 적중률은 매우 높아질 것으로 보인다.

표 5. 백엔드 서버의 평균 선행 계산 시간

태그 개수 (m)	최대 해시 체인 길이 (n)	선행 계산 해시 체인 길이 (p)	선행 계산 시간
10^6	1×10^3	10	14.132 sec
	2×10^3	20	33.312 sec
	3×10^3	30	50.616 sec
10^5	1×10^3	100	14.482 sec
	2×10^3	200	30.636 sec
	3×10^3	300	50.777 sec
10^4	1×10^3	1000	14.459 sec
	2×10^3	2000	33.489 sec
	3×10^3	3000	53.872 sec

표 6. 적중률에 따른 백엔드 서버의 평균 계산 시간

태그 개수	최대 해시 체인 길이	적중률	Ohkubo의 방법	제안 기법
10^6	1×10^3	100%	32,048 *ms*	0.000870 *ms*
	2×10^3		58,650 *ms*	0.000916 *ms*
	3×10^3		88,958 *ms*	0.000866 *ms*
10^5	1×10^3		31,777 *ms*	0.000894 *ms*
	2×10^3		64,752 *ms*	0.001208 *ms*
	3×10^3		82,896 *ms*	0.000844 *ms*
10^4	1×10^3		9,101 *ms*	0.000912 *ms*
	2×10^3		22,203 *ms*	0.000678 *ms*
	3×10^3		27,409 *ms*	0.001224 *ms*
10^6	1×10^3	90%	32,048 *ms*	3,166 *ms*
	2×10^3		58,650 *ms*	6,819 *ms*
	3×10^3		88,958 *ms*	8,540 *ms*
10^5	1×10^3		31,777 *ms*	3,080 *ms*
	2×10^3		64,752 *ms*	3,172 *ms*
	3×10^3		82,896 *ms*	7,839 *ms*
10^6	1×10^3	80%	32,048 *ms*	4,865 *ms*
	2×10^3		58,650 *ms*	14,748 *ms*
	3×10^3		88,958 *ms*	17,473 *ms*
10^5	1×10^3		31,777 *ms*	6,181 *ms*
	2×10^3		64,752 *ms*	7,621 *ms*
	3×10^3		82,896 *ms*	14,423 *ms*
10^6	1×10^3	70%	32,048 *ms*	9,048 *ms*
	2×10^3		58,650 *ms*	18,306 *ms*
	3×10^3		88,958 *ms*	27,601 *ms*
10^5	1×10^3		31,777 *ms*	7,036 *ms*
	2×10^3		64,752 *ms*	17,547 *ms*
	3×10^3		82,896 *ms*	25,086 *ms*

V. 결 론

향후 RFID 시스템의 활용도가 높아짐에 따라서 여러 가지 사용 환경에서 RFID 시스템의 사용자 프라이버시 문제가 거론될 것으로 보인다. 본 논문에서는 RFID 시스템에서의 프라이버시 침해 문제와 이를 해결하기 위한 3가지 필수 보안 요건을 살펴보았다. 3가지 보안 요건을 모두 충족시키는 Ohkubo의 안전한 프라이버시 보호 프로토콜이 있지만, 백엔드 서버의 계산 시간은 확장성을 보장하지는 못한다. 본 논문에서는 Ohkubo 프로토콜의 보안 수준은 그대로 유지하면서, 백엔드 서버에서의 태그 판별 시간 단축을 위한 새로운 기법을 제안하였다. 제안한 기법에서는 백엔드 서버가 선행 계산을 한 후, 선행 계산된 해시 값들을 정렬시켜 이진 검색을 통해 빠른 검색을 할 수 있도록 하였다. 이로써 Ohkubo 프로토콜의 원래 방법보다 현저하게 빠른 태그 판별 시간을 얻을 수 있음을 시뮬레이션을 통해서 보여주었다.

참 고 문 헌

[1] K. Finkenzeller, *RFID Handbook*, John Wiley & Sons, 1999.

[2] 조태남, 이상호, "(2,4)-트리를 이용한 그룹키 관리", 정보보호학회논문지, 11권, 4호, pp. 77-64, 2001.

[3] 박영호, 이경현, "이동네트워크 환경에서 그룹키 관리구조", 정보보호학회논문지, 12권, 2호, pp. 89-77, 2002.

[4] 권정옥, 황정연, 김현정, 이동훈, 임종인, "일방향 함수와 XOR을 이용한 효율적인 그룹키 관리 프로토콜 : ELKH", 정보보호학회논문지, 12권, 6호, pp. 93-112, 2002.

[5] 이상원, 천정희, 김용대, "Pairing을 이용한 트리 기반 그룹키 합의 프로토콜", 정보보호학회논문지, 13권, 3호, pp. 101-110, 2003.

[6] 박영희, 정병천, 이윤호, 김희열, 이재원, 윤현수, "Diffie-Hallman 키 교환을 이용한 확장성을 가진 계층적 그룹키 설정 프로토콜", 정보보호학회논문지, 13권, 5호, pp. 3-15, 2003.

[7] S.-S. Yeo and S. K. Kim, "Scalable and Flexible Privacy Protection Scheme for RFID System", In Proceedings of the 2nd European Workshop on Security and Privacy in Ad hoc and Sensor Networks (ESAS 2005), vol. 3813 of LNCS, pp. 153-163, July 2005.

[8] S. Sarma, S. Weis and D. Engels, "RFID Systems and Security and Privacy Implications", In Proceedings of Workshop on Cryptographic Hardware and Embedded Systems 2002, vol. 2523 of LNCS, pp. 454-469, August 2002.

[9] D. Molnar and D. Wagner, "Privacy and Security in Library RFID Issues, Practices, and Architectures", In Proceedings of the 11th ACM Conference on Computer and Communications Security (ACM CCS 2004), pp. 210-219, October 2004.

[10] S. Weis, S. Sarma, R. Rivest and D. Engels, "Security and Privacy Aspects of Low-cost Radio Frequency Identification Systems", In Proceedings of the 1st International Conference on Security in Pervasive Computing, vol. 2802 of LNCS, pp. 454-469, March 2003.

[11] M. Feldhofer, "An Authentication Protocol in a Security Layer for RFID Smart Tags", In Proceedings of the 12th IEEE Mediterranean Electrotechnical Conference (MELECON 2004), pp.759-762, May 2004.

[12] D. Henrici and P. Müller, "Hash-based Enhancement of Location Privacy for Radio-Frequency Identification Devices using Varying Identifiers", In Proceedings of the 1st IEEE International Workshop on Pervasive Computing and Communication Security, pp. 149-153, March 2004.

[13] M. Ohkubo, K. Suzuki and S. Kinoshita, "Cryptographic Approach to "Privacy-Friendly" Tags", In RFID Privacy Workshop, MIT Media Laboratory, November 2003.

[14] M. Ohkubo, K. Suzuki and S. Kinoshita, "Efficient Hash-Chain Based RFID Privacy Protection Scheme", In Privacy Workshop at the Sixth International Conference on Ubiquitous Computing (UbiComp 2004), September 2004.

[15] *Crypto++ Library*, http://www.eskimo.com/~weidai/cryptlib.html

〈著者紹介〉

여 상 수 (Sang-Soo Yeo)
1997년 2월: 중앙대학교 컴퓨터공학과 졸업
1999년 2월: 중앙대학교 컴퓨터공학과 석사
2005년 8월: 중앙대학교 컴퓨터공학과 박사
2006년 3월 ~ 현재: 단국대학교 정보컴퓨터학부 강의전임강사
〈관심분야〉 정보보호, 암호프로토콜, 네트워크 보안, 생물정보학
email: ssyeo@dankook.ac.kr

김 순 석 (Soon-Seok Kim)
1997년 2월: 진주대학교 컴퓨터공학과 졸업
1999년 2월: 중앙대학교 컴퓨터공학과 석사
2003년 2월: 중앙대학교 컴퓨터공학과 박사
2003년 3월 ~ 현재: 한라대학교 컴퓨터공학과 조교수
〈관심분야〉 정보보호, 암호프로토콜
email: sskim@halla.ac.kr

김 성 권 (Sung Kwon Kim)
1981년 2월: 서울대학교 통계학과 졸업
1983년 2월: 한국과학기술원 전산과 석사
1990년 8월: University of Washington 전산과 박사
1991년 3월 ~ 1996년 2월: 경성대학교 전산통계학과 조교수
1997년 3월 ~ 현재: 중앙대학교 컴퓨터공학부 교수
〈관심분야〉 알고리즘, 정보보호, 암호프로토콜
email: skkim@cau.ac.kr