情報保護學會論文誌
第17卷 第2號, 2007. 4

고속 패킷 통신을 위한 패킷 암호 스킴과 Cryptoki 확장 방안

고 행 석,[1†] 박 상 현,[1†] 권 오 석[2‡]
[1]국가보안기술연구소, [2]충남대학교

A Packet encryption scheme and extension of Cryptoki for connectionless packet network

Haeng-Seok Ko,[1†] Sang-Hyun Park,[1†] Oh-Seok Kwon[2‡]
[1]National Security Research Institute(NSRI), [2]Chungnam National University

요 약

비 연결형 패킷 통신망에 Block Chaining 모드를 사용하여 통신패킷을 암호화하는 암호통신시스템에서 통신패킷의 송수신 순서가 달라지거나 통신패킷이 분실되면 암호 통신 성능 및 효율성이 저하된다. 이를 해결하기 위하여 통신패킷의 송수신 순서에 무관하게 암복호 처리 할 수 있는 고속 패킷통신을 위한 패킷 암호 스킴을 제안하였다. 제안한 패킷 암호 스킴은 키교환할 때 생성한 IV(Initial Vector)와 암호모듈 내부에서 난수로 만든 salt를 이용하여 새로운 IV를 만들고, 이 IV와 공통키를 사용하여 통신패킷별로 암호화한다.

패킷 암호 스킴을 적용하기 위한 패킷 암복호 함수 및 메커니즘을 추가한 확장된 Cryptoki를 제안하여 편리성과 성능을 개선하였다. 제안한 패킷암호 함수의 성능을 비교하기 위하여 암호모듈을 구현하였으며, 시험한 환경에서 패킷 암호 함수를 적용하여 20회 암호화하였을 때 기존 Cryptoki API에 비하여 약 1.5배에서 15.6배 정도의 성능 향상을 가져왔다.

ABSTRACT

In connectionless packet network, if a sender encrypts packets by block chaining mode and send it to receiver, the receiver should decrypt packets in encrypted order that is not received order. Therefore, the performance and efficiency are lowered for crypto communication system. To solve this problem, we propose packet encryption scheme for connectionless packet network that can decrypt the packets independently, even if the received order of packets are changed or packets are missed. The scheme makes new IV(Initial Vector) using IV that created by key exchange process and salt that made by random number.

We propose extended Cryptoki API that added packet encryption/decryption functions and mechanism for improving convenience and performance. We implement the scheme and get result that the performance increased about 1.5~15.6 times compare with in case of implementing using Cryptoki API in the test environment.

Keywords : *packet encryption, connectionless packet network, Cryptoki API*

접수일: 2006년 9월 15일; 채택일: 2007년 1월 3일

† 주저자, hsko@etri.re.kr
‡ 교신저자, oskwon@cnu.ac.kr

Ⅰ. 서 론

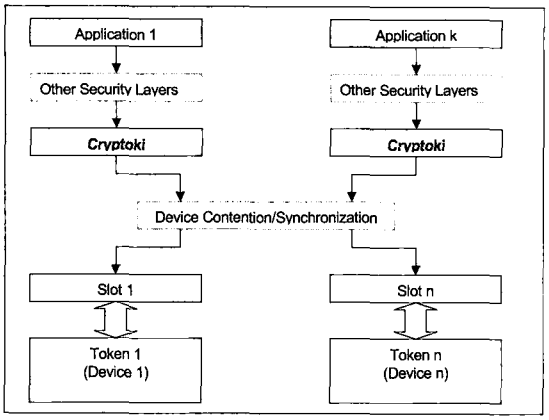
VPN(Virtual Private Network), 파일 암호장비, 암호전화기 등의 다양한 암호장비를 개발할 때 동일한 암호서비스를 제공하는 모듈에 대한 각 응용프로그램별 보안기능의 중복개발 및 자원의 중복 사용을 막기 위하여 표준화된 다양한 보안 API(Application Program Interface)가 사용되고 있다. 암호모듈은 암호장비에 장착되어 암호 서비스 및 메커니즘을 제공하는 모듈로 다양한 암호장비 및 응용 프로그램이 공통으로 사용하는 암호서비스 기능을 제공한다. 암호장비를 개발할 때 암호모듈을 사용하고 암호모듈로부터 암호 서비스를 제공받도록 설계하면 암호장비가 연동되는 시스템의 구축 및 제작이 용이해지므로 암호장비에 공통으로 사용되는 암호모듈을 적용하는 것이 일반적이다. 암호장비에서 암호모듈을 공통 적용하기 위하여 응용 프로그램에서 암호모듈을 사용하는 방법 및 절차를 기술한 표준화된 다양한 보안 API가 제시되었다. 대표적인 API로는 IETF(Internet Engineering Task Force)에서 제안한 GSS-API(Generic Security Services API), X/OPEN의 GCS-API(Generic Cryptographic Service API), 마이크로소프트사의 CryptoAPI, RSA Lab사의 Cryptoki, 인텔에서 제안한 CDSA-API(Common Data Security Service Architecture API)등이 있다.[1][2] 특히 Cryptoki API는 응용프로그램이 요구하는 라이브러리를 암호모듈(토큰)이 융통성 있도록 지원하고, 휴대성, 확장성, 일반성, 알고리즘 독립성 등을 고려하여 RSA Lab에 의하여 개발된 API로서 산업 및 학계에서 실질적 표준으로 사용하는 암호 API이다. 암호가 널리 사용되면서 개발자와 관계없이 상호연동성이 매우 중요한 요소가 됨에 따라 Cryptoki API는 응용과 스마트카드, PCMCIA카드 스마트디스켓과 같은 휴대형 암호 장치와 연동하기 위한 목적으로 고안되었다.[3] 그러나, [4]와 같이 국내 전자서명 표준인 KCDSA(Korea Certificate-based Digital Signature Algorithm) 등의 알고리즘에 대한 메커니즘이 제공되지 않아 추가하는 경우와 효율성을 증대시키기 위하여 통합된 기능을 추가하는 경우 등과 같이 Cryptoki API를 암호시스템 개발에 적용할 때, 표준을 그대로 사용하지 않고 함수 및 메커니즘을 추가해야할 필요가 있는 경우가 있다.

본 논문은 통신패킷의 수신 순서가 달라지거나, 통신패킷의 손실이 있는 패킷통신망을 사용하는 암호통신시스템에서 통신패킷별로 독립적으로 암호화하여 성능과 효율성을 제고할 수 있는 패킷 암호 스킴을 제안한다. 통신패킷을 암호화하는 암호통신시스템에서 송신자가 메시지를 Block Chaining 모드로 암호화하여 통신패킷으로 분할하여 전송할 때 수신자는 암호 순서와 다른 통신패킷이 수신되면 암호 순서에 맞는 통신패킷이 도착할 때까지 대기하여야 하므로 성능의 저하가 발생된다. 이 문제를 해결하기 위하여 제안한 패킷 암호스킴은 패킷별로 서로 다른 IV(Initial Vector)를 이용하여 독립적으로 암호화하기 때문에 전송 순서에 무관하게 복호화할 수 있다. 또한 패킷 암호 스킴을 암호모듈에 적용하기 위하여 Cryptoki API를 확장한 패킷 암호/복호 함수 및 메커니즘을 제안한다. 제2장에서는 Cryptoki에 대하여 간략하게 살펴보고, 패킷통신에서 발생하는 문제점과 이를 기존 Cryptoki API로 적용하는 방안에 대하여 살펴본다. 제3장에서는 패킷 암호 스킴과 Crypto-ki 확장 방안을 살펴보고, 제4장에서는 암호모듈을 구현 및 시험 결과를 제시하고, 제5장에서는 결론을 맺는다.

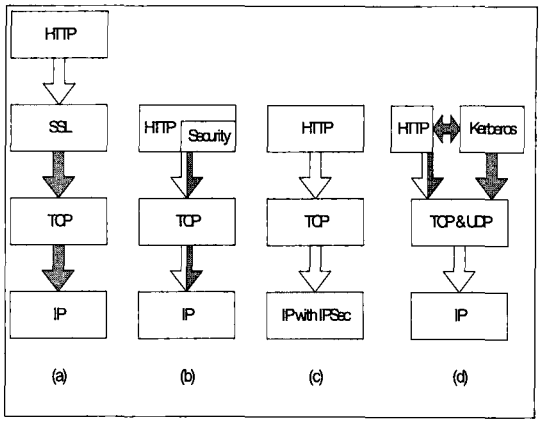
Ⅱ. 관련연구

2.1 Cryptoki

RSA Laboratories사에서 제안한 규격인 PKCS(Public Key Cryptography System)는 컴퓨터에 적용되는 공개키와 관련된 기술을 개발하는 개발자를 위하여 만들어졌다. PKCS 규격으로 인하여 개발을 빨리할 수 있을 뿐만 아니라 PKCS를 적용하는 제품 간의 상호운용이 가능하도록 PKI(Public Key Infrastructure)의 데이터 형식, 통신 프로토콜, API등에 대하여 규정하고 있다. PKCS는 #1에서 #15까지 규정되어 있으며, 암호모듈(토큰)의 정합규격인 PKCS#11은 Cryptoki(Cryptographic Token Interface)로 불린다. Cryptoki의 일반적인 모델은 [그림 1]과 같다. Cryptoki 모델은 암호 기능 수행이 필요한 1개 이상의 응용프로그램과 암호 기능을 수행하는 1개 이상의 암호 모듈로 구성된다. Cryptoki는 다수의 슬롯을 통하여 1개 이상의 암호모듈에 인터페이스를 제공한다. 암호모듈은 암호키 등의 중요정보를 저장하고, 암호 함수를 수행하는 장치이다. 슬롯은 암호모듈과 연결되는 리더에 해당된다. Cryptoki API는 일반목적함수, 슬롯과 토큰관리함수, 세션관리함수, 객체관리함수, 암호함수, 복호함수, 메시지 다이제스트



〔그림 1〕 일반적 Cryptoki 모델



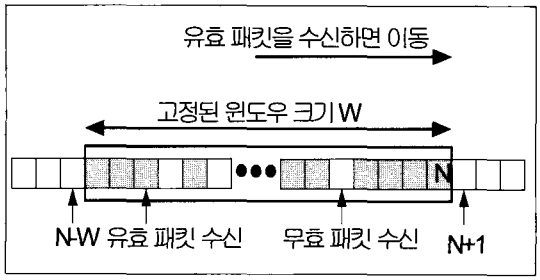
〔그림 2〕 네트워크 보안 접근 방법

함수, 서명/MAC(Message Authentication Code)함수, 검증함수, 이중목적함수, 키관리함수, 난수생성함수 등으로 나누어지고 이 함수들의 조합에 의하여 암호 기능을 수행하게 된다.[3]

2.2 TCP/IP 네트워크와 IPSec

인터넷을 통하여 사용자들이 신뢰할 수 없는 사이트와 연결하는 것을 허용하지 않고 기관에서 나가는 패킷은 암호화하고, 들어오는 패킷은 인증함으로서 인터넷을 통한 TCP/IP 네트워크를 안전하게 운용할 수 있다. 이러한 사용자의 요구를 달성하기 위하여 네트워크 보안 접근 방식이 몇 가지 제안되었다. 네트워크 보안 접근 방식은 SSL(Secure Sockets Layer)과 TLS(Transport Layer Security)와 같이 분리된 보안 프로토콜과 응용 프로그램에 보안 서비스를 직접 접목시킨 특정 응용 보안 방법(〔그림 2-b〕), IPSec과 같이 핵심 통신 프로토콜에 보안 기능을 내장한 방식(〔그림 2-c〕), MIT에서 개발한 Kerberos 프로토콜 등의 툴킷을 적용한 병렬 보안 프로토콜(〔그림 2-d〕)등이 있다.[5] 특히 IP 계층에서 보안을 구현하면 사용자는 보안 메커니즘을 갖는 응용 프로그램뿐만 아니라 보안기능이 없는 응용 프로그램에 대해서도 안전한 네트워크를 보장할 수 있다. 이러한 요구에 따라 IETF는 인터넷 레벨에서 보안 기능을 규정한 IPSec 보안 관련 제안 표준을 발표하였다.

일반적으로 많이 사용하는 대표적인 패킷 통신인 IP는 비 연결형이기 때문에 신뢰할 수 없는 서비스이므로 프로토콜은 패킷이 반드시 순서대로 전달된다는 보장을 할 수 없고, 모든 패킷이 손실 없이 전달된다는 보장도



〔그림 3〕 Anti-Replay 메커니즘을 이용한 유효패킷 확인

할 수 없다. 따라서 재전송 공격에 취약할 수 있으므로 이에 대응하는 메커니즘을 이용하여 유효 패킷을 확인하는 것이 필요하므로 IPSec 인증 문서에서는 그림3과 같이 수신자가 반드시 윈도우 크기 W를 구현해야 한다고 규정하고 있다.

〔그림 3〕에서 윈도우의 오른쪽 끝은 지금까지 수신한 유효 패킷의 가장 높은 순서 번호 N을 나타내고 N-W+1부터 N사이의 범위에 있는 순서 번호를 가진 모든 패킷이 정확하게 수신된 것이다.[6]

IPSec에서 ESP(Encapsulating Security Payload)는 메시지 내용의 기밀성과 제한된 트래픽 흐름의 기밀성을 포함한 기밀성 서비스를 제공한다. 페이로드 데이터, 패딩, 패딩길이 그리고 다음 헤더 필드는 ESP 서비스에 의하여 암호화된다. 페이로드를 암호화하는데 사용되는 알고리즘이 IV(Initial Vector)와 같이 암호학적 동기 데이터가 필요하다면, 이 데이터는 페이로드 데이터 필드의 시작 부분으로 전달된다. 이미 포함되어 있다면, 암호문의 일부로 간주되더라도 IV는 암호화되지 않는다. 암호 모드는 CBC(Cipher Block Chaining)를 사용하도

록 규정되어 있다.[6]

2.3 Cryptoki API를 사용한 패킷통신의 문제점

기밀성을 보장하기 위하여 메시지를 암호화할 때 Block Chaining모드를 사용하지 않고 ECB(Electronic Code Book) 모드를 사용하면 동일한 평문에 대하여 동일한 암호문이 나오기 때문에 패킷통신의 기밀성이 떨어지므로 ECB 모드는 사용되지 않는다. 따라서 패킷 통신에서 암호모드를 ECB모드가 아닌 Block Chaining 모드를 사용하는 것이 일반적이며 Block Chaining 모드로 암호화된 메시지를 통신패킷으로 분할하여 전송하면, 수신 순서에 따른 문제점들이 발생될 수 있다. Block Chaining 모드를 사용하여 페이로드를 암호화하는 패킷통신 시스템에서, 수신된 통신패킷의 순서가 암호화한 패킷의 순서와 다르면 복호화할 다음 순서의 통신패킷이 도착할 때까지 먼저 도착한 통신패킷은 복호화하지 않고 대기하여야 한다. 패킷이 분실된 경우는 송신자는 분실된 해당 패킷을 재전송하기 위하여 전송할 메시지를 처음부터 다시 암호화하여야 하므로 패킷 암호 통신 시간이 급격하게 증가되는 문제가 발생될 수 있다. 이경우를 대비하여 송신자는 송신한 통신패킷을 저장 및 관리하면 효율적으로 운용할 수 있다.

IPSec에서는 Window 크기에 따른 패킷을 관리하고, 페이로드 처음에 IV를 저장할 수 있도록 규정하고 있다. 하나의 세션에 하나의 IV만을 사용하면 패킷의 도착 순서에 따라 복호화 순서가 영향을 받으므로 복호대기 시간이 길어지는 것은 동일하다. IV를 페이로드에 첨부하여 보내면 IV의 노출 등으로 기밀성이 떨어지고 전달할 수 있는 데이터 크기가 줄게 된다. 특히 CBC모드를 사용할 때 공격자가 임의로 IV를 조작할 수 있으면 수신자에게 엉뚱한 암호문을 수신하게 할 수 있고, 평문 첫 블록의 선정된 비트들의 값을 임의로 바꿀 수 있다. 따라서 IV는 보안을 강화하기 위하여 암호키와 같은 수준으로 보호되어야 한다.[7][8][9][10]

암호화된 통신패킷이 장거리 라우팅을 거쳐 통신패킷의 도착순서가 달라지는 경우가 빈번한 통신시스템에서는 재전송에 많은 로드가 걸려서 전송속도가 저하된다. 암호화된 통신패킷에 에러가 발생한 경우도 동일하게 성능 저하의 원인된다.

본 논문에서는 IP 통신뿐만 아니라, 다양한 비 연결형 패킷통신시스템에서 암호화된 순서와 수신된 순서가 달라져 발생할 수 있는 문제를 해결하기 위하여 패킷의 도착 순서에 무관하게 기밀성을 보장할 수 있도록 통신패킷간의 암호화된 순서의 연관성이 없는 패킷 암호 방법인 고속 패킷 통신을 위한 패킷 암호 스킴과 이를 Cryptoki에 적용하는 방안을 제안한다.

Ⅲ. 고속 패킷통신을 위한 Cryptoki 확장 메커니즘

3.1 비 연결형 패킷 통신망에 적합한 패킷 암호 스킴

2.3절에서 살펴본 패킷 통신의 문제점을 해결하기 위하여 기밀성 및 효율성이 제고되는 패킷 암호 방안을 제안한다. 본 논문에서 제안하는 패킷 암호 및 복호 스킴은 다음과 같다.

송신자는 패킷 페이로드의 암호문을 생성한다.

- *salt = random number*
- *NewIV = IV xor (0's padding || salt)*
- *CipherText* = $E_{SK,NewIV}(PlainText)$

송신자는 페이로드에 salt와 암호문을 포함한 패킷을 생성하여 송신한다.

- *salt || CipherText*

수신자는 수신한 패킷의 페이로드에서 salt와 암호문을 추출하고 평문을 계산한다.

- *NewIV = IV xor (0's padding || salt)*
- *PlainText* = $D_{SK,NewIV}(CipherText)$

이때 salt의 길이는 IV길이보다 작은 크기를 사용한다. 난수로 생성한 salt가 반복될 확률은 해시함수의 충돌쌍을 찾는 확률과 동일하다. n이 salt의 비트수이고 $2^{n/2}$개의 salt값을 원소로 하는 두 집단에서 임의의 두 salt 값이 동일할 확률은 아래와 같다.

$$P(2^n, 2^{\frac{n}{2}}, 2^{\frac{n}{2}}) \approx 1 - e^{-\frac{2^{\frac{n}{2}} \times 2^{\frac{n}{2}}}{2^n}} = 1 - e^{-1}$$

$$\approx 0.63$$

IPSec에서 Sequence Number Counter를 32비트로 정의하고 있으므로 한 세션에서 최대로 만들어질 통신 패킷의 개수는 2^{32}이다. 따라서 salt의 길이를 64비트로 결정하면, 한 세션 내에서 충돌쌍이 발생할 최대 확률은 0.63이 된다. 그러나 하나의 세션에서 최대로 사용하는 통신 패킷

의 개수인 $2^{n/2}$가 줄어들면 위 수식에 의하여 충돌쌍이 존재할 확률이 줄어든다. 또한 암호모듈에서 서로 다른 salt를 생성하도록 처리하면, Sequence Number Counter 비트 크기로도 충돌쌍이 존재하지 않는 salt를 만들 수 있다.

위의 과정에서 CBC 모드에서는 임의로 salt를 조작할 수 있으면 [7][8][9][10]과 같이 보안의 취약성이 발생할 수 있으므로, 암호모듈 내부에서 난수발생기로 생성한 salt를 사용하고 CFB(Cipher FeedBack), OFB(Output FeedBack)등 안전한 암호 모드를 적용한다. 또한 암호키와 IV를 안전하게 저장 및 암호함수를 수행하는 암호모듈의 사용이 필요하다. 본 논문에서 제안하는 고속 패킷 통신을 위한 패킷 암호 스킴을 암호모듈에 적용한 방안은 그림4와 같다. 송신자와 수신자가 키교환 과정을 통하여 공통키(암호키)와 IV를 공유한다. 송신자와 수신자는 각각 암호키와 IV를 암호모듈에 저장하고 다음과 같은 연산을 수행한다.

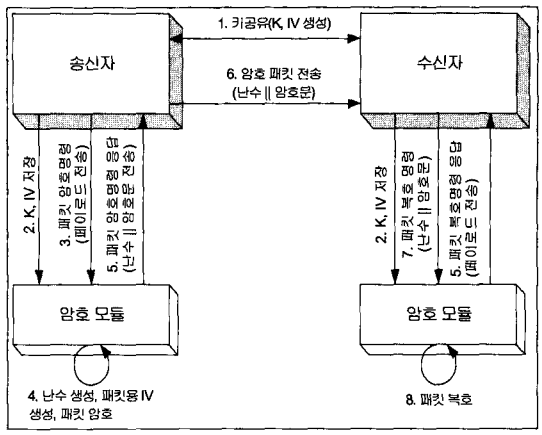
송신자는 n개의 암호화된 페이로드를 전송하기 위하여 단위크기의 패킷을 다음과 같이 계산한다.

```
{
  for i = 1 to n  loop
    salt <- random number;
    new IV <- IV xor (0's padding || salt)
    send {salt || E_Key, New IV(payload(i))}
  endloop
}
```

수신자는 i번째 암호화된 패킷을 수신하면 페이로드를 복호화하기 위하여 다음과 같이 계산한다. 이때 수신한 패킷은 암호화된 순서에 무관하게 해당 패킷을 복호화할 수 있다.

```
{
  for i = 1 to n  loop
    receive{salt || E_Key, New IV(payload(i))}
    new IV <- IV xor (0's padding || salt)
    D_Key, New IV(E_Key, New IV(payload(i)))
  endloop
}
```

위에서 살펴본바와 같이 패킷의 도착 순서에 무관하게 복호화할 수 있으므로 별도의 추가적인 버퍼링 및 부가적인 계산이 필요하지 않다. 또한 수신된 패킷에 에러가 발생하면 재전송 요구를 하고, 송신자는 새로 패킷을 암호화하여 전송하므로 해당패킷을 생성하기 위하여



〔그림 4〕 제안한 패킷 통신 암호 방안

처음부터 다시 계산을 하지 않는다.

제안한 고속 패킷 통신을 위한 암호 스킴을 적용하면 서로 다른 IV로 생성된 암호화 패킷은 패킷의 순서와 무관하게 패킷의 기밀성을 유지할 수 있게 된다. 따라서 안전하게 패킷통신을 하기 위하여 송신자와 수신자는 공통의 암호키와 IV를 저장하고, 패킷 암호함수를 수행하는 암호모듈을 사용해야한다.

3.2 고속 패킷통신을 위한 Cryptoki 확장 메커니즘

본 논문에서 제안한 패킷암호 스킴을 실질적 산업 표준인 Cryptoki API에 적용하면, 암호화 하는 절차는 다음과 같이 구현될 수 있다.

```
{
  C_FindObjectsInit;
  IV, Key <- C_FindObjects;
  C_FindObjectsFinal;
  for i = 1 to n  loop
    salt <- C_generateRandom;
    New_IV <- IV xor (0's padding || salt)
    C_CreateObject <- New_IV;
    C_EncryptInit;
    C_EncryptUpdate;
    C_EncryptFinal;
    C_DestroyObject;
    send packet;
  endloop:
}
```

패킷통신 복호 절차는 다음과 같이 구현될 수 있다.

```
{
  C_FindObjectsInit;
  IV, Key <- C_FindObjects;
  C_FindObjectsFinal;
  for i = 1 to n  loop
    receive packet and extract salt;
    New_IV <- IV xor (0's padding || salt)
    C_CreateObject <- New_IV;
    C_DecryptInit;
    C_DecryptUpdate;
    C_DecryptFinal;
    C_DestroyObject;
  endloop:
}
```

위와 같이 Cryptoki를 적용하면 패킷마다 IV가 달라지기 때문에 새로운 IV 객체를 생성하고 객체 핸들을 *C_EncryptInit* 및 *C_DecryptInit*에 인자로 전달하여야 한다. 따라서 패킷암호 스킴을 Cryptoki API로 구현하면 위의 loop내의 절차는 매 패킷마다 수행하여야한다.

본 논문에서는 기존의 Crytptoki API를 확장하여 고속처리에 적합한 패킷 암복호 명령과 메커니즘을 추가하였다. 새로 추가한 Cryptoki 함수는 *C_PacketEncryptInit, C_PacketEncryptUpdate, C_PacketEncryptFinal, C_PacketDecryptInit, C_PacketDecryptUpdate, C_PacketDecryptFinal* 이다.

C_PacketEncryptInit 및 *C_PacketDecryptInit*의 함수 정의는 〔그림 5〕와 같다. *hSession*은 C_*OpenSession*에 의하여 생성된 세션 핸들을 할당하고, 메커니즘은 패킷통신용 블록 암호 알고리즘 및 암호모드를 선택한다. *hKey*와 *hIV*는 키교환에 의하여 생성 및 저장된 암호키와 IV를 지정하는 핸들이다.

*C_PacketEncryptUpdate*명령은 *ulSaltLen*만큼의 난수를 생성하여 평문을 암호화하고 암호문과 함께 salt를 전달한다. *C_PacketEncryptUpdate*의 함수 정의는 〔그림 6〕과 같다.

*pPart*는 평문 데이터의 포인터를 지정하고, *ulPartLen*은 평문데이터 길이를 지정한다. *pEncryptedPart*는 암호문을 전달한 버퍼의 포인터를 지정하고, *pulEncryptedPartLen*은 암호문의 크기를 전달할 변수의 포인터를 지정한다. *pSalt*는 생성한 salt가 저장된 버퍼의 포인터를 지정하고, *ulSalt-Len*은 생성한 salt의 크기가 저장된다.

C_PacketDecryptUpdate 명령은 salt와 암호문을 입력받아 암호문을 복호화하고 평문을 전달한다. *C_PacketDecryptUpdate*의 함수 정의는 〔그림 7〕과 같다.

```
CK_DEFINE_FUNCTION(CK_RV, C_PacketEncryptInit){
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey,
    CK_OBJECT_HANDLE hIV
};
CK_DEFINE_FUNCTION(CK_RV, C_PacketDecryptInit){
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey,
    CK_OBJECT_HANDLE hIV
};
```

〔그림 5〕 *C_PacketEncryptInit, C_PacketDecryptInit* 함수 정의

```
  CK_DEFINE_FUNCTION(CK_RV,
C_PacketEncryptUpdate){
      CK_SESSION_HANDLE hSession,
      CK_BYTE_PTR pPart,
      CK_ULONG ulPartLen,
      CK_BYTE_PTR pEncryptedPart,
      CK_ULONG_PTR pulEncryptedPartLen,
      CK_BYTE_PTR pSalt,
      CK_ULONG ulSaltLen
  };
```

〔그림 6〕 *C_PacketEncryptUpdate* 함수 선언

```
  CK_DEFINE_FUNCTION(CK_RV,
C_PacketDecryptUpdate){
      CK_SESSION_HANDLE hSession,
      CK_BYTE_PTR pSalt,
      CK_ULONG ulSaltLen,
      CK_BYTE_PTR pEncryptedPart,
      CK_ULONG ulEncryptedPartLen,
      CK_BYTE_PTR pPart,
      CK_ULONG_PTR pulPartLen
  };
```

〔그림 7〕 *C_PacketDecryptUpdate* 함수 선언

```
CK_DEFINE_FUNCTION(CK_RV, C_EncryptFinal){
    CK_SESSION_HANDLE hSession
};

CK_DEFINE_FUNCTION(CK_RV,
C_PacketDecryptFinal){
    CK_SESSION_HANDLE hSession
};
```

〔그림 8〕 C_PacketEncryptFinal, C_PacketDecryptFinal 함수 선언

*C_PacketEncryptFinal*와 *C_PacketDecryp-tFinal* 함수 정의는 〔그림 8〕과 같다.

위에서 정의한 패킷 암호 함수 및 메커니즘을 사용하여 생성된 암호 패킷은 각 패킷간의 순서에 무관하다. 통신패킷을 암호화할 때 패킷마다 IV를 갱신할 필요가 없으므로 *C_PacketEncryptUpdate* 함수를 반복 호출하도록 다음과 같이 구현될 수 있다.

```
{
 C_FindObjectsInit;
  IV, Key <- C_FindObjects;
  C_FindObjectsFinal;
C_PacketEncryptInit;
  for i = 1 to n  loop
    C_PacketEncryptUpdate;
    send packet;
  endloop:
  C_PacketEncryptFinal;
}
```

통신패킷의 복호 절차는 다음과 같이 구현될 수 있다.

```
{
  C_FindObjectsInit;
  IV, Key <- C_FindObjects;
  C_FindObjectsFinal;
  C_PacketDecryptInit;
  for i = 1 to n  loop
    receive packet and extract salt;
    C_PacketDecryptUpdate;
  endloop:
  C_PacketDecryptFinal;
}
```

패킷 암호 함수를 이용하면 성능 측면에서 기존 Cryptoki 함수에 비하여 객체관리, 암호모드 설정 등이 필요 없고 *C_PacketEncryptUpdate* 및 *C_PacketEncrypt Update*의 반복으로 암호 및 복호를 할 수 있으므로 속도 측면에서 유리하다.

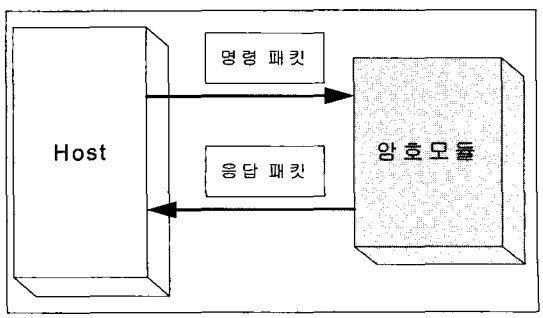
고속처리를 위하여 하드웨어 구현 관점에서 살펴보면 객체관리 및 암호모드 설정 등의 함수는 하드웨어로 구현하기가 어렵고, *C_PacketEncryptUpdate* 및 *C_PacketDecryptUpdate*는 하드웨어로 구현하기가 비교적 쉽다. 그러므로 *C_PacketEncryptUpdate* 및 *C_Packet DecryptUpdate* 함수를 하드웨어(암호칩)으로 구현하면 [13][14]와 같이 암호칩의 성능이 패킷 암호성능이 되므로 수백 ~ 수기가급의 패킷통신시스템에 패킷 암호 스킴을 적용할 수 있다.

보안 안전성 측면에서는 암호모듈에 저장된 암호키, IV 등의 객체를 SO(Security Officer)가 관리하여야 하나, Cryptoki 함수를 사용하여 패킷을 암호화하면 암호모듈 외부에서 사용자가 IV를 관리하므로 보안 취약점이 존재하게 된다.

Ⅳ. 설계 및 구현

4.1 암호모듈 설계 및 구현

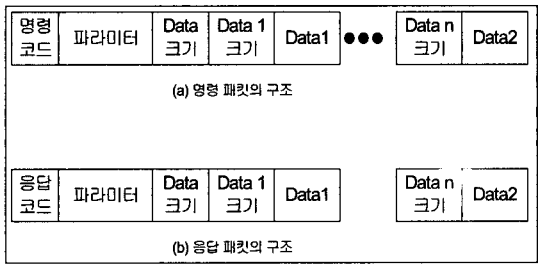
암호모듈은 일반적으로 암호장비 또는 전산장비 등의 호스트에 PCI, USB, PCMCIA등의 표준 인터페이스 또는 전용 인터페이스를 통하여 연동된다. Cryptoki API는 암호모듈과 응용프로그램간의 논리적 관점의 인터페이스이므로, 실질적으로 데이터를 전송하기 위하여 〔그림 9〕와 같이 명령패킷을 암호모듈 인터페이스로 전달하고, 응답패킷을 받는 구조로 암호모듈을 구현하였



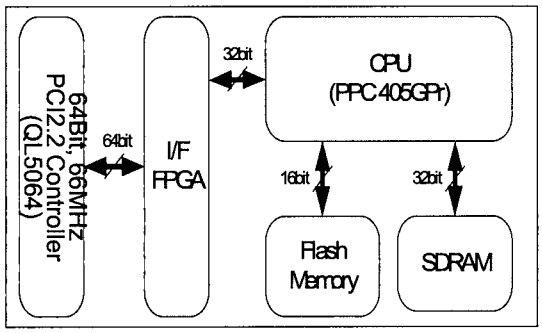
〔그림 9〕 호스트와 암호모듈간의 통신

다. 암호모듈은 응용 프로그램이 실행되는 호스트로부터 명령패킷을 받아 해당되는 명령을 수행하고 응답패킷을 보내는 구조이다. 명령패킷과 응답패킷은 [그림 10]과 같이 구성하였다. 명령패킷은 명령을 구분하는 명령코드, 전달되는 파라미터 데이터, 데이터 등으로 구성되어 있다. 응답패킷은 명령에 대한 응답코드가 들어 있고, 명령의 결과로서 처리된 데이터가 들어 있다.

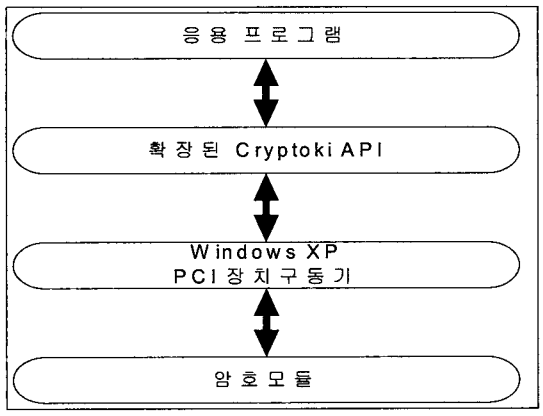
암호모듈과 응용프로그램간의 데이터 처리는 [그림 12]와 같이 응용프로그램에서 Cryptoki 함수를 호출하



[그림 10] 명령패킷과 응답패킷의 구성



[그림 11] 암호모듈 하드웨어 블록도



[그림 12] 소프트웨어 계층구조 모델

면, 확장된 Cryptoki API 계층과 장치구동기를 거쳐서 명령패킷이 암호모듈로 전달된다. 암호모듈은 명령패킷을 받아 해당되는 명령을 처리하고, 응답결과로서 응답패킷을 생성한 후 장치구동기가 읽어가도록 준비한다. 장치구동기는 응답패킷을 읽어서 API 계층으로 전달하고, API 계층은 Cryptoki API 형태로 응용프로그램으로 응답 결과를 전달한다.

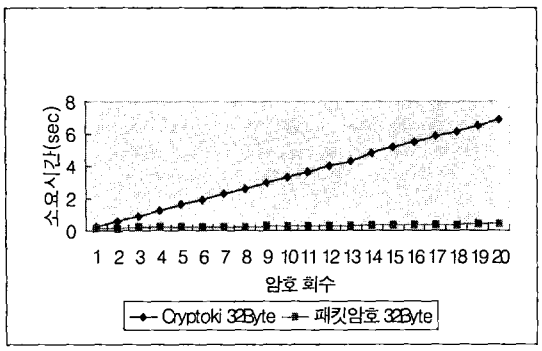
제안한 패킷 암호 스킴을 검증 및 성능 평가를 하기 위하여 [그림 11]과 같이 암호모듈을 설계 및 구현하였다. 호스트가 Cryptoki 함수를 명령패킷 형태로 암호모듈로 전달하면, 암호모듈은 해당 Cryptoki 함수를 수행하고 그 결과를 응답패킷으로 호스트로 전달하도록 설계하였다. 구현한 암호모듈은 호스트로 PC 및 Workstation을 사용할 수 있도록 P-CI 인터페이스를 적용하여 설계하였다. PCI 컨트롤러는 QuickLogic사의 QL5064를 사용하고, CPU로 IBM PPC 405GPr를 사용하였다. 암호 알고리즘은 AES256을 사용하였고, 패킷 암호 스킴에서 사용하는 salt의 크기는 64비트를 사용하였다.

제안한 패킷 암호 스킴과 패킷 암호 함수의 성능을 평가하기 위하여 Windows XP 환경에서 [그림 12]와 같은 소프트웨어 구조를 사용하여 구현하였다. 응용 프로그램은 본 논문에서 제안한 패킷 암호 스킴을 기존 Cryptoki를 이용하여 구현하는 경우와, 제안한 패킷암호 함수를 이용하여 구현한 경우를 비교하여 성능을 시험하는 프로그램을 구현하였다. 성능은 응용프로그램에서 clock 함수를 이용하여 msec 단위로 측정하였다. API는 DLL(Dynamic Link Library)형태로 구현을 하였으며 기존 Cryptoki API에 패킷암호 함수를 추가하였다. 장치구동기는 암호모듈과 고속으로 데이터를 송수신할 수 있도록 DMA 기능을 사용하였다.

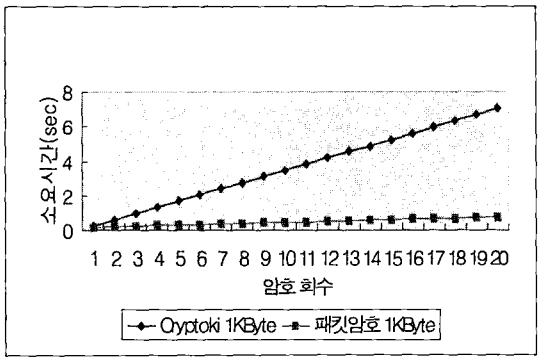
4.2 성능분석

[그림 13], [그림 14], [그림 15]에서는 32 byte, 1 Kbyte, 32 KByte 3가지 크기의 평문을 각각 1회에서부터 20회까지 Cryptoki 함수와 패킷암호함수를 이용하여 암호화하였을 때의 소요 시간을 나타낸다. [그림 13]은 32 Byte 데이터를 1회에서 20회까지 암호화할 때 소요된 시간으로 암호 횟수가 증가될수록 소요시간의 차이가 커지는 것을 볼 수 있다. 특히 20회 암호화 하였을 경우 약 15.6배의 성능 증가를 보인다.

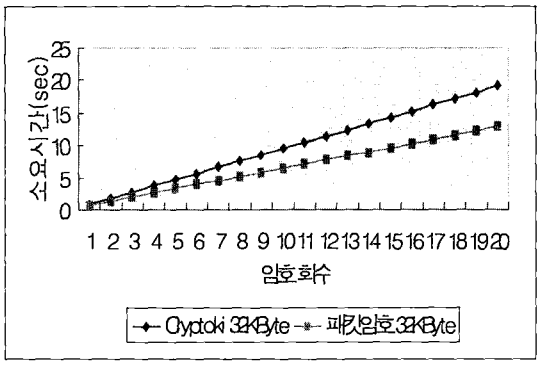
[그림 14]에서 1 KByte 데이터를 20회 암호화 하였



〔그림 13〕 패킷 암호 소요시간(32Byte)



〔그림 14〕 패킷 암호 소요시간(1KByte)



〔그림 15〕 패킷 암호 소요시간(32KByte)

을 경우 약 10.6배의 성능 증가를 보였고 〔그림 15〕에서는 약 1.5배 성능 증가를 보였다.

위의 결과를 보면 암호화할 패킷 개수가 증가될수록, 패킷 크기가 작을수록 성능향상이 되었으나, 〔그림 15〕에서와 같이 32 KByte 데이터를 암호화 할 경우 작은 데이터 크기를 암호화하는 경우에 비하여 상대적인 성능이 증가가 작은 이유는 암호알고리즘이 소프트웨어로 구현되어 객체관리, 암호초기화 등 다른 Cryptoki 함수에 비하여 암호화하는데 소요되는 시간이 크기 때문이다. 따라서 암호모듈에 구현된 암호알고리즘의 처리시간을 단축시키면 성능이 증가된다.

V. 결 론

본 논문에서는 Block Chaining 모드를 사용하여 통신 패킷을 암호화하는 암호통신시스템에서 패킷의 도착순서가 달라지거나, 패킷이 분실되면 패킷 암호 통신의 성능이 저하되는 패킷 암호 통신의 문제점에 대하여 살펴보았다. 패킷 암호 통신의 문제점을 해결하기 위하여 패킷별로 독립적으로 IV를 사용하는 패킷 암호 스킴을 제안하였으며, 암호 API의 실질적 표준인 Cryptoki API에 패킷 암호 스킴을 적용하기 위한 패킷암호 함수와 IV운용 메커니즘을 제안하였다. 패킷 암호 스킴을 구현할 때 제안한 패킷 암호 함수를 사용하면, 기존 Cryptoki API에 비하여 고속으로 통신 패킷을 암호화·복호화할 수 있으며, 보안 취약점을 제거할 수 있다. 또한 패킷 암호함수는 하드웨어로 구현하기가 용이하여 고속 네트워크의 통신 속도에 대응되는 암호 통신시스템 개발에 적용할 수 있다. 제안한 패킷암호 함수를 평가하기 위하여 암호모듈을 구현하였고 기존 Cryptoki 함수와 패킷 암호함수의 성능을 비교하였다. 32Byte에서 32KByte까지의 데이터를 대상으로 20회 전송 시험한 결과 제안한 패킷암호 함수를 적용하면 약 1.5배에서 15.6배정도의 성능향상을 가져왔다.

향후 연구과제로 고속 패킷 통신을 위한 패킷암호 스킴이 적용되고, Cryptoki 함수를 ASIC으로 구현한 하드웨어 기반 암호모듈을 설계하고 암호모듈 성능 평가 모델에 관련된 연구를 수행할 계획이다.

참고문헌

〔1〕 박수진, 신동명, 김학범, 최용락, "국제 보안 API 표준화 동향," *통신정보보호학회지*, 11(1), pp. 55-63, 2001.

〔2〕 주학수, 이언경, 김승주, "암호라이브러리 및 암호API 개발 현황," *정보보호학회지*, 12(4), pp. 94-103, 2002.

〔3〕 RSA Laboratories, "*PKCS #11 v2.2*," 2004.

〔4〕 김명희, 김은환, 전문석, "KCDSA 메커니즘을 제공하는 PKCS#11 설계 및 분석", *정보보호*

학회논문지, 14(5), pp. 141-151, 2004.

[5] Stephen A. Thomas, *SSL & TLS Essentials Securing the web*, Willey Computing Publishing, 2000.

[6] William Stallings, *Cryptography and Network Security*, Prentice Hall, 2003.

[7] Voydock V., Kent S., "Security Mechanisms in High-Level Network Protocols," *Computing Surveys*, pp. 135-171, J-une 1983.

[8] Steven M. Bellovin, "Problem Areas for the IP Security Protocols," *Proceedings of sixth annual USENIX Security Symposium*, pp. 205-214, July 1996.

[9] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway, "A concrete security treatment of symmetric encryption," *Proceedings of the 38th IEEE symposium on Foundations of Computer Science*, IE-EE, 1997, pp. 394 - 403.

[10] P. Rogaway, "Nonce-based symmetric encryption, Fast Software Encryption," *11th International Workshop, FSE 2004*, Delhi, India, February 5-7, 2004.

[11] Yoe-Sub Shin, Yang-Gyu Kim, Haeng-Seok Ko, Dong-Heyok Jan-g, Taejoo Chang, Oh-Seok Kwon, "A Study on the High Speed Crypto-System for the M-ultiSessions," *SAM03*, 2003.

[12] 이상한, 고행석, 장태주, 김영수, 양상운, 박상현, 구본석, "고속 세션 변경이 가능한 블록 암호화 장치 및 그 구동방법," 특허번호 10-0420555, 2004.

[13] 전신우, 정용진, 권오준, "Rijndael 암호 알고리즘을 구현한 암호 프로세서의 설계," *정보보호학회논문지*, 11(6), pp. 77-87, 2001.

[14] 안하기, 신경욱, "AES Rijndael 블록 암호 알고리즘의 효율적인 하드웨어 구현," *정보보호학회논문지*, 12(2), pp53-63, April, 2002.

〈著者紹介〉

고 행 석 (Haeng-Seok Ko) 정회원
1990년 2월: 충남대학교 컴퓨터공학과 졸업
1992년 2월: 충남대학교 컴퓨터공학과 석사
1992년~2000년: 국방과학연구소 선임연구원
2000년~현재: 국가보안기술연구소 선임연구원
<관심분야> 정보보호, 암호모듈, 컴퓨터 구조

박 상 현(Sang-Hyun Park) 정회원
1992년 2월: 충남대학교 컴퓨터공학과 졸업
1996년 2월: 충남대학교 컴퓨터공학과 석사
1996년~2000년: 국방과학연구소 연구원
2000년~현재: 국가보안기술연구소 선임연구원
<관심분야> 정보보호, 모바일 이동통신, 임베디드 시스템

권 오 석 (Oh-Seok Kwon) 정회원
1977년 2월: 서울대학교 전자공학과 졸업
1980년 2월: 한국과학기술원 전기 및 전자공학과 석사
1980년~현재: 충남대학교 전기정보통신공학부 교수
<관심분야> 정보보호, 컴퓨터통신, 퍼지시스템