Step by Step

스텝 바이 스텝 3

네트워크 보안 기술의 원리와 응용

메시지 인증과 공개키 암호화

인터넷 환경에서 가장 중요한 것은 보안이다. 가정에서 개인용 PC로 인터넷 뱅킹이 가능하게 된 기술적 배경은 무엇일까? 바로 보안 알고리즘이 있었기에 가능한 일이다. 알고리즘이란 말에 언뜻 복잡해 보이지만, 의외로 원리가 간단하다는 사실에 놀랄 것이다. 연재를 통해 네트워크 보안 전문가로 거듭나 보자.

연 재 순 서
1회 | 2007. 11 | 메시지 인증과 공개키 암호화
2회 | 2007. 12 | 보안 애플리케이션, Kerberos와 PGP
3회 | 2008. 1 | 인터넷 보안, IPSec
4회 | 2008. 2 | 네트워크 보안 관리, SNMP

연 재 가 이 드
운영체제 | 윈도우, 리눅스, 유닉스, 매킨토시 등 모든 플랫폼
기초지식 | 네트워크 개론
응용분야 | 네트워크 보안 애플리케이션, 프로토콜 구현

서상원 smiler@ssm.samsung.co.kr | 필자는 2005년부터 삼성소프트웨어멤버십 15기로 활동 중이다. 최적의 개발을 위한 플랫폼솔루션과 개발 프레임워크에 큰 관심을 두고 있으며, 이에 기반 되는 아키텍처 설계에 고심을 하고 있다. 현재는 삼성전자에서 미디어 셋톱박스 관련 프로젝트를 수행하고 있다.

메시지를 인증하는 방법은 일반적으로 3가지 측면에서 정리할 수 있다. 첫 번째는 예전에 사용했던 방식인 메시지 전송측과 수신측에서 동일한 키(Key)를 공유하는 방식이다. 이 방식은 전송측에서 메시지를 키로 암호화시켜 수신측에 전송하고, 수신 측은 공유하던 자신의 키를 이용해 메시지를 복호화 하는 방식이다. 하지만 이 방식은 메시지의 단순한 인증만을 필요로 하는 경우에도 메시지를 모두 암호화하기 때문에 비용이 많이 든다. 그렇기 때문에 메시지 암호화 없이 인증하는 두 번째 방법이 제안됐다. 물론 이 방식도 단순한 원리를 갖고 있다. 각각의 메시지에 간단한 인증 태그를 붙이는 방식이다. 그러나 태그 생성 시 원문 메시지에 근거하지 않으므로 원문 메시지의 무결성(Integrity)까지는 체크할 수 없다는데 문제가 있다.

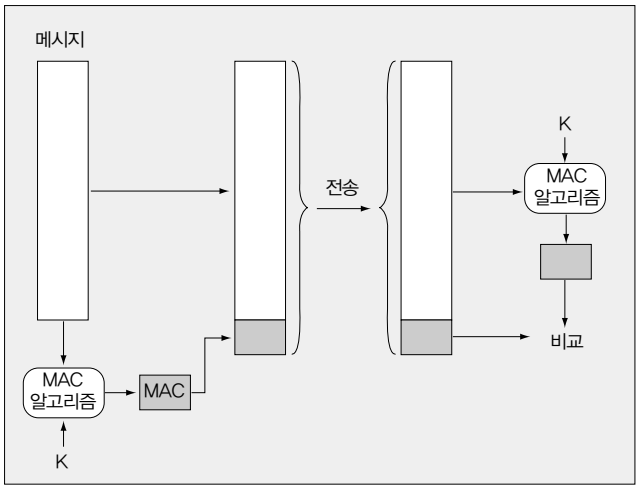
이는 중간에 메시지 내용이 변경될 수도 있다는 것을 의미한다. 그래서 만들어진 방법이 세 번째, 바로 MAC(Message Authentication Code)을 이용한 인증방법이다. MAC은 원문 메시지를 근거하여 생성된 비트코드를 의미하며 복호화가 불가능하다는 것에 의미를 둔다. 원문 메시지의 무결성을 완벽히 보장한다. 대표적인 메시지 인증 알고리즘으로 MD5, SHA 등이 있다. 이번 호에서는 이런 알고리즘에 대해 자세히 다뤄 볼 것이다. 더불어 인터넷뱅킹 사례를 예로 공개키를 사용하는 이유와 키를 생성하는 원리를 중심으로 전반적인 공개키 암호화 시스템에 대해 다룰 것이다. 대부분의 인터넷 사용자는 인터넷뱅킹을 이용하며, 공인인증서라는 것을 가지고 있다. 바로 이것이 CA(공인인증기관)에서 인증하고 배포한 공개키라는 것을 알고 있는가? 이번 호를 통해 궁금증을 풀어보자.

필자 메모

개발자라면 누구나 한번쯤은 암호화 라이브러리를 사용해본 경험이 있을 것이다. 특히 해쉬(hash)함수는 다양한 분야에서 응용되고 있다. 활용도가 높은 만큼, 어느 정도 구현원리를 이해하고 있다면 필요에 따라 해당 알고리즘을 변형하여 프로젝트에 적용할 수 있다. 4회에 걸친 연재를 통해 네트워크상에서 쓰이는 보안 알고리즘의 원리와 활용 예를 살펴볼 예정이다. 알고리즘을 나열하기 보다는 원리와 개념을 명확히 이해하는 데 주안점을 둘 것이다.

메시지 인증

〈그림 1〉을 분석해보자. 〈그림 1〉에서는 송신측과 수신측이 키(Key)를 공유한다는 가정이 필요하다. 송신측에서 K라는 키를 이용해 MAC 알고리즘의 입력으로 원문 메시지를 넣으면 출력으로 일정한 비트의 MAC을 얻는다. 이렇게 생성한 MAC을 원문 메시지에 붙여 전송(Transmit)한다. 수신측에서는 수신된 패킷을 메시지와 MAC으로 분리하는 작업을 가장 먼저 수행한다. 그 후에 메시지를 송신할 때와 동일하게 MAC 알고리즘의 입력



〈그림 1〉 일반적인 MAC을 이용한 메시지 인증

으로 원문 메시지를 넣으면 동일한 MAC을 얻게 된다. 이렇게 생성한 MAC과 수신된 MAC을 비교하여, 일치하면 인증절차를 완료한다. 즉 원문 메시지가 중간에 변형된 것이 없다는 것을 보장하고, MAC 알고리즘의 원리상 동일한 키가 아닌 경우는 MAC의 결과가 달라지므로 정확히 원하는 송신자로부터 패킷이 도착했다는 것을 의미한다. 또한 메시지가 시퀀스번호(X.25, TCP 등에서 사용됨)를 사용한다면 공격자는 순서번호를 임의로 변경할 수 없으므로 수신자는 정확한 순서라는 것도 확신할 수 있다.

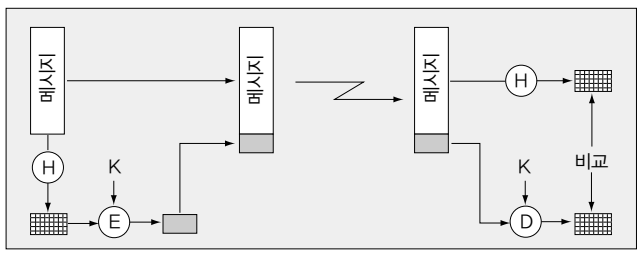
단방향 해쉬함수

메시지 인증 코드를 위한 또 다른 방법은 단방향 해쉬함수(One-way hash function)이다. 해쉬함수는 임의 크기의 메시지 M을 입력으로 받아들여 일정한 크기의 메시지 다이제스트(Message Digest)인 H(M)를 출력하는 함수이다. 단방향이라 함은 암호화는 되는데 복호화는 불가능하다는 것을 의미한다. 해쉬함수를 이용해 메시지를 인증하려면 인증된 상태의 메시지 다이제스트를 메시지와 함께 전송한다. 〈그림 2〉, 〈그림 3〉, 〈그림 4〉를 통해 단방향 해쉬함수의 세 가지 방법을 살펴보자.

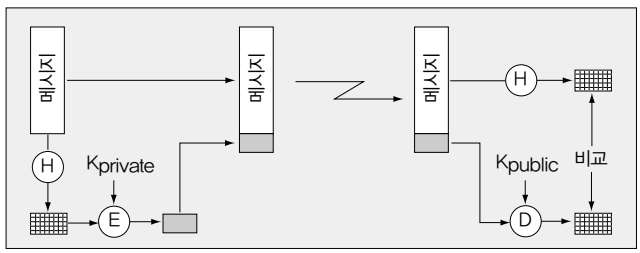
〈그림 2〉의 방식은 〈그림 1〉의 방식과 키를 공유하는 측면에서 비슷하다. 해쉬함수의 입력으로 원문 메시지를 넣으면 일정한 크기의 MD(Message Digest)를 만들고, DES나 IDEA와 같은 대칭키 알고리즘으로 키를 이용해 암호화 한다. 이때 암호화 된 값을 E(MD)라고 하자. 그 다음은 MAC을 이용한 인증방법과 동일하게 원문 메시지에 E(MD)를 붙여서 메시지를 전송한다. 이때 해쉬함수는 단방향이므로 복호화가 안 되기 때문에 인증을 위해서는 원문 메시지를 해쉬함수의 입력으로 넣어 MD를 얻고, 수신한 E(MD)를 복호화시켜 직접 만든 MD와 비교하는 절차가 필요하다. 해쉬함수 자체가 단방향이므로 생성한 MD가 원문 메시지로 복호화가 불가능한 것이고, E(MD)에 사용한 대칭키 암호화 알고리즘은 양방향이라는 것을 오해하지 않도록 하자.

〈그림 3〉의 공개키를 이용한 알고리즘도 〈그림 2〉의 방법과 거의 유사하지만, 두 가지 정도의 장점이 있다. 한 가지는 메시지 인증뿐만 아니라 디지털 서명도 제공한다. 다른 한 가지는 통신에 참여하는 양쪽 단말에 키를 분배할 필요가 없다는 것이다. 이해가 잘 안될 수도 있다. 이런 부분은 뒷부분의 공개키를 이용한 인터넷뱅킹 사례를 설명하면서 디지털 서명에 대해 다루겠다.

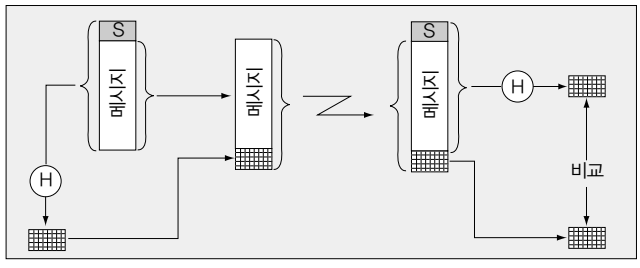
두 가지 방법을 써서 MD를 암호화하고 복호화 하는 알고리즘이 반드시 필요하다는 것을 확인했다. 하지만, 안전한 암호화를 위해 많은 계산량이 필요하고, 라이선스 비용 등의 예기치 않았던 문제점이 발생할 수 있다. 더구나 가장 널리 알려진 대칭키 알고리즘인 DES는 테러국가 등에는 수출제한이 걸려있다. 결국 암호화 알고리즘을 이용하지 않는 방식을 개발하기에 이르렀다. 바로 〈그림 4〉의 비밀 키를 이용한 방법이다. 이 방식은 양쪽 단말에 비밀 값(S)을 갖고 있다는 가정을 한다. 송신측에서는 전송할 메시지에 비밀 값(S)을 붙여 해쉬함수로 MD를 생성하고, MD를 원문 메시지에 붙여 전송한다. 수신측에서는 MD와 원문



〈그림 2〉 일반적인 해쉬함수



〈그림 3〉 공개키를 이용한 해쉬함수



〈그림 4〉 비밀키를 이용한 해쉬함수

메시지를 분리해 원문 메시지를 송신측과 동일한 방법으로 비밀값을 붙여 MD를 생성한다. 이렇게 생성한 MD와 전송된 MD를 비교함으로써 인증절차를 수행한다. 이 방법은 비밀 값만 누출하지 않으면 적은 비용으로 첫 번째, 두 번째 방법과 동일한 효과를 얻는다. 생각보다 원리가 단순하다는 것을 알 수 있다. 세 번째 기술의 변형으로 HMAC이라는 것이 있다. 후반부에 IP 보안을 다루면서 살펴볼 것이다.

안전한 해쉬함수의 조건

표준화 포럼에서는 몇 가지의 조건을 명시한다. 첫째는 어떤 크기의 원문 메시지(데이터블록)도 입력으로 가능해야 하며, 일정한 크기(Bit)로 정해진 출력을 생성해야 한다. 그렇기 때문에 각 해쉬함수의 버전에 따른 스펙에는 입출력 비트의 크기가 정의되어 있다. 다음으로 해쉬함수 내부의 계산량이 작아야 한다. 해쉬함수 자체가 복잡해서 성능에 문제를 초래하면 안 되기 때문이다. 또한 단방향성을 보장한다. Hash(x)=y의 경우, x를 역으로 찾아내는 것이 계산적으로 불가능해야 한다. 마찬가지로 Hash(x)=Hash(y)를 만족할 때 x와 y가 다른 경우를 허용하면 안 된다. 허용할 경우가 생기면 데이터 무결성을 보장하지 못하는 결과를 발생한다.

	SHA-1	MD5	RIPEMD-160
Digest length	160비트	128비트	160비트
Basic unit of processing	512비트	512비트	512비트
Number of steps	80(4 rounds of 20)	64(4 rounds of 16)	160(5 paired rounds of 16)
Maximum message size	264-1비트	제한 없음	제한 없음

〈표 1〉 해쉬함수 비교

〈표 1〉을 보면 3개의 기본적인 해쉬함수의 알고리즘 스펙을 확인할 수 있다. 원문 메시지를 M이라고 할 때, 해쉬함수의 출력물은 메시지 다이제스트(MD)라고 했다. 즉 Digest length는 출력물의 길이를 의미한다. 해쉬함수의 조건에 따라 일정한 길이를 가지고 있다. Basic unit of processing은 내부적으로 원문 메시지를 512비트라는 단위로 처리한다는 의미이다. 대부분의 해쉬함수가 512비트 단위로 메시지를 블록으로 나누어 처리한다. 나머지는 MD5를 살펴보며 이해하자.

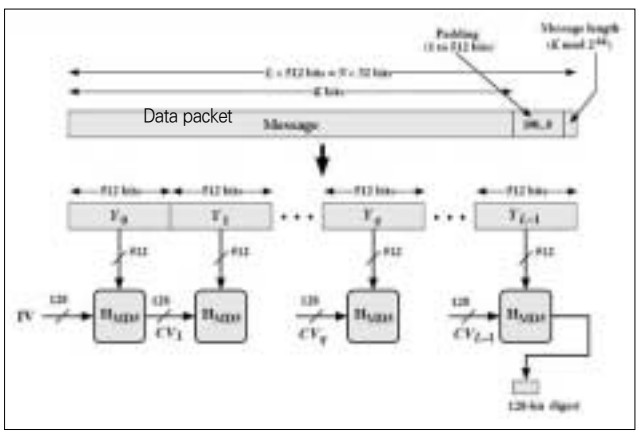
MD5 알고리즘

MD5(Message Digest) 알고리즘은 MIT의 Ron Rivest가 개발한 알고리즘으로 얼마 전까지 가장 널리 쓰이던 해쉬함수다

RIPEMD-160

RIPEMD-160의 개발은 유럽의 RIPE(RACE Integrity Primitives Evaluation)프로젝트 주관 하에 MD4와 MD5의 취약점을 분석했던 연구 그룹이 주도하였다. 이때 128비트-RIPEMD를 개발했다. 그러나 H.Dobbertin가 다시 취약점을 발견했고, 다시 160비트의 RIPEMD를 개발했다. 이것이 바로 RIPEMD-160이다. RIPEMD-256은 128비트 RIPEMD의 확장옵션 알고리즘이고 RIPEMD-320은 RIPEMD-160의 확장옵션 알고리즘이라고 이해하면 된다.
전반적인 구조는 SHA1, MD5와 거의 유사하며 〈표 1〉에서 기본적인 스펙을 확인할 수 있다. 자세한 내용과 소스 코드는 http://homes.esat.kuleuven.be/~bosselae/ripemd160.html를 방문하길 바란다.

(같은 계열의 MD2, MD4 등도 존재한다). 최근에는 전수공격과 암호해독에 대한 우려가 확산되는 추세이다. 그러나 MD5의 동작원리가 대부분의 다른 해쉬함수와 유사하므로 MD5를 이해하면 모두 이해할 수 있다. 이 알고리즘은 임의 길이의 메시지를 입력해서 128비트 길이의 메시지 다이제스트(DM)를 출력한다. 이때 원문 메시지를 한 번에 입력으로 하는 것이 아니라 512비트 단위로 나눠서 블록으로 처리한다.



〈그림 5〉 MD5를 이용한 메시지 다이제스트 생성

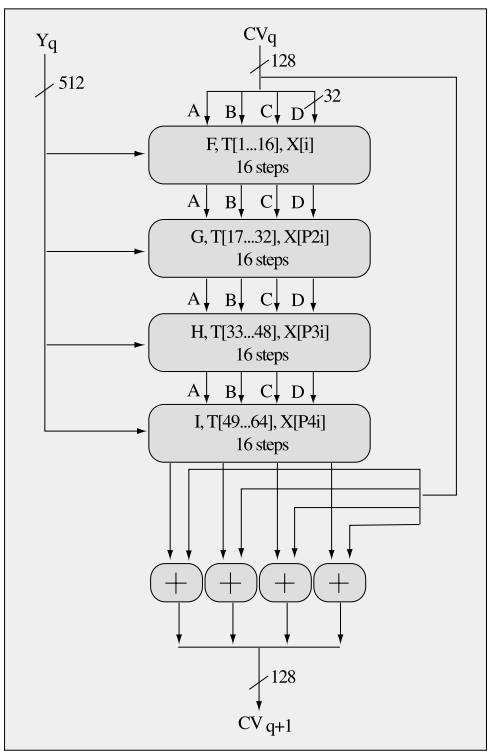
〈그림 5〉는 MD5 알고리즘의 전체적인 처리 흐름을 보여준다. 원문 메시지를 512비트 단위로 나누고 마지막 512비트를 맞추기 위해 패딩(Padding)을 한다. 예를 들어 512비트 씩 나누고 마지막에 남은 메시지가 200비트라면 0으로 채워진 312비트를 붙여서 512비트로 맞춰준다. 이렇게 하는 이유는 내부적으로 해쉬함수의 입력이 512 비트로 구성되어 있고 나머지 비트의 경우에 대한 예외처리가 없기 때문이다. 알고리즘의 단순화를 위해 대부분의 해쉬함수 알고리즘이 패딩정책을 사용한다. MD5의 알고리즘 입력으로 원문 메시지 이외에도 이전에 처리된 해쉬함수의 결과인 벡터 값이 입력으로 들어간다. 즉 512비트 단위로 생성한 해

쉬함수의 결과물이 다음 블록에 영향을 주어 연쇄모드(Chaining Mode)를 구성한다. 이때 가장 처음 블록의 경우에는 이전 결과물이 없으므로 IV(Initialize Vector)라는 초기 벡터 값을 사용한다. 초기 벡터 값은 임의로 정의할 수 있으며, 일반적으로 〈리스트 1〉의 값을 사용한다.

〈리스트 1〉 MD5의 IV(Initialize Vector) 예시

```
//Initialize variables:
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476
```

결국 블록(512비트)마다 128비트의 출력물을 생성한다. 다음 블록에 연속적으로 쓰이므로 결과적으로는 128비트의 일정한 메시지 다이제스트를 생성한다. 이제 전체적인 해쉬함수의 그림을 그렸으니 자세한 내부를 살펴보겠다.



〈그림 6〉 512비트 블록 한 개를 처리하는 MD5 해쉬함수 내부

〈표 1〉의 MD5를 보면 Number of steps 항목이 16 step의 4 round 인 것을 확인할 수 있다. 〈그림 6〉을 보면 16 steps 블록이 4개인데, 바로 이것이 Number of steps를 나타낸다. 즉 MD5는 4개의 라운드(round) 함수로 이루어졌고, 각 라운드 함수에는 16 단계의 처리과정이 있다고 이해하면 된다. 512비트의 메시지 블록이 입력으로 들어오면 4개의 라운드에 전달한다. 또한 〈그림 5〉에서 설명했듯이 이전 결과인 벡터 값 역시 입력으로 전달된다. 이때 각각의 라운드 수행은 동시에 진행될 수 없다. 이는 이전 라운드의 결과 값이 다음 라운드의 입력으로 필요하기 때문이다. 각각의 라운드는 512비트를 4개의 128비트로 나누어 처리하며, 최종 라운드의 결과를 초기에 전달받은 벡터 값과 XOR 연산을 통해 128비트의 최종 결과물을 생성한다.

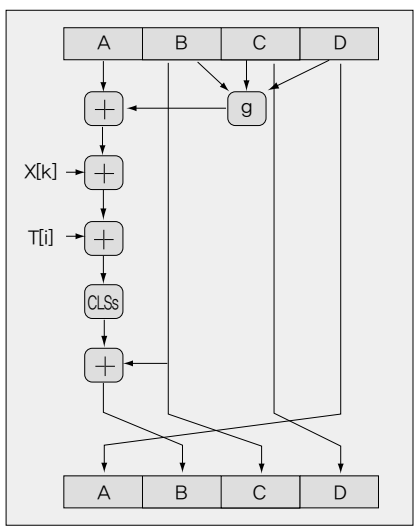
Round	Primitive function g	g(b, c ,d)
1	F(b, c, d)	(b∧c)∨(b∧d)
2	G(b, c, d)	(b∧d)∨(c∧d)
3	H(b, c, d)	b⊕ c⊕ d
4	I(b, c, d)	c⊕ (b ∨ d)

〈표 2〉 라운드 별 로컬함수 연산식

b	c	d	F	G	H	I
0	0	0	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	1	1	0
0	1	1	1	0	0	1
1	0	0	0	0	1	1
1	0	1	0	1	0	1
1	1	0	1	1	0	0
1	1	1	1	1	1	0

〈표 3〉 로컬함수 연산 진리표

〈그림 6〉의 각 라운드의 F, G, H, I는 라운드별 로컬함수를 일컫는다. 로컬함수의 기본 연산식은 〈표 2〉와 〈표 3〉의 진리표를 확인하면 된다. 〈그림 6〉의 T[]는 미리 계산된 사인 함수 테이블로 연산에 사용되는 미리 정의된 값이라고 이해하면 된다. 〈표 2〉에서 각 함수는 g로 표시했으며 인자 값으로 B, C, D를 사용했음을 잊지 말고 〈그림 7〉의 스텝 내부를 살펴보자. 〈그림 7〉에서는 128비트 단위로 나눈 A, B, C, D를 라운드함수 g와의 XOR 연산을 통해 처리되는 과정을 보여준다. g는 〈표 2〉에 명시된 데로 각 라운드 함수에 따라 처리 연산식이 다르다. 즉 모든 라운드



〈그림 7〉 라운드의 스텝(step) 내부 알고리즘

의 스텝은 〈그림 7〉의 알고리즘을 따른다.

〈그림 7〉은 'b ← b + ((a + g(b, c, d) + X[k] + T[i] ⋘ s)' 수식으로 나타낼 수 있다. 실제 알고리즘을 코드로 구현할 때는 이를 그대로 쓴다. 이로써 MD5의 내부까지 모두 살펴봤다. 생각보다 단순한 원리인 '반복'이라는 개념이 대부분이라는 것을 느꼈을 것이다. 실제 내부 알고리즘은 단순하다. 단순하다는 것도 좋은 해쉬함수의 조건이라는 것은 앞서 설명했었다. 단순하다고 해서 견고하지 못한 것은 아니라는 것도 이해하길 바란다. SHA나 RIPEMD와 같은 해쉬함수도 거의 동일한 원리로 동작한다. 크게 보면 라운드함수의 연산식이 다르며 반복의 횟수가 다를 뿐이다. 이제 MD5 알고리즘에 대한 설명은 〈리스트 2〉의 의사코드를 분석으로 마무리하겠다. 〈리스트 2〉의 주석을 참고하길 바란다.

〈리스트 2〉 MD5 의사코드 구현 예시

```
var int[64] r, k
r[ 0..15] := {7, 12, 17, 22,  7, 12, 17, 22,  7, 12, 17,
22,  7, 12, 17, 22}
r[16..31] := {5,  9, 14, 20,  5,  9, 14, 20,  5,  9, 14,
20,  5,  9, 14, 20}
r[32..47] := {4, 11, 16, 23,  4, 11, 16, 23,  4, 11, 16,
23,  4, 11, 16, 23}
r[48..63] := {6, 10, 15, 21,  6, 10, 15, 21,  6, 10, 15,
21,  6, 10, 15, 21}

// 연산에 사용하기 위한 미리 정의된 사인함수 생성
for i from 0 to 63
    k[i] := floor(abs(sin(i + 1)) × 2^32)

//Initialize variables: (IV 값 지정, 표준에 정의된 값은 없으며, 관용적으
로 아래의 값을 쓴다)
var int h0 := 0x67452301
var int h1 := 0xEFCDAB89
var int h2 := 0x98BADCFE
var int h3 := 0x10325476

//Pre-processing: 512비트 단위의 블록으로 나누고 마지막 블록은 패딩처리
append "1" bit to message
append "0" bits until message length in bits ≡ 448 (mod
512)
append bit length of message as 64-bit little-endian
integer to message


//Process the message in successive 512-bit chunks:
//각각의 블록의 라운드 함수 부분

for each 512-bit chunk of message
    break chunk into sixteen 32-bit little-endian words
w(i), 0 ≤ i ≤ 15

    //Initialize hash value for this chunk:
    var int a := h0
    var int b := h1
    var int c := h2
    var int d := h3

    //Main loop: <표 1>의 내용대로 64번의 처리과정이 있고 4개의 라운드를
      가지고 있다.
    //f는 <표 2>의 g에 해당하는 라운드 함수를 나타낸다.
    for i from 0 to 63
        if 0 ≤ i ≤ 15 then
            f := (b and c) or ((not b) and d)
            g := i
        else if 16 ≤ i ≤ 31
            f := (d and b) or ((not d) and c)
            g := (5×i + 1) mod 16
        else if 32 ≤ i ≤ 47
            f := b xor c xor d
            g := (3×i + 5) mod 16
        else if 48 ≤ i ≤ 63
            f := c xor (b or (not d))
            g := (7×i) mod 16

          // <그림7>의 b <- b + (( a + g(b, c, d) + X[k] +
            T[i] <<< s )를 코드화
         temp := d
         d := c
         c := b
         b := ((a + f + k(i) + w(g)) leftrotate r(i)) + b
         a := temp

         //end of main loop

    //Add this chunk's hash to result so far:
    h0 := h0 + a
    h1 := h1 + b
    h2 := h2 + c
    h3 := h3 + d

// 최종 128비트 메시지 다이제스트 생성
var int digest := h0 append h1 append h2 append h3
//(expressed as little-endian)
```

공개키 암호화

메시지 인증과 키 분배에 사용되는 가장 중요한 기술이 공개키 암호화다. 앞서 메시지 인증에 대한 내용을 언급하며 송수신 양단간에 키가 사용되는 것을 알아봤다. 이때 사용되는 것이 공개키-개인키 메커니즘이다. 여기서는 공개키 암호의 기본적인 개념을 설명하고, 디지털 서명과 관련된 인터넷뱅킹 사례를 살펴본다. 가장 핵심이 되는 알고리즘인 RSA와 Diffie-Hellman에 대해서는 자세히 분석해 본다.

공개키 암호의 구조

1976년 Diffie와 Hellman에 의해 최초로 제안된 공개키 암호는 수천 년을 이어져 내려온 암호학에 있어 혁명적인 발전을 가져왔다. 첫째, 공개키 알고리즘은 비트 패턴상의 단순한 조작이 아니라 수학적 함수에 근거해서 만들었다. 둘째, 더 중요한 사항

은 전통적인 대칭키 알고리즘과 대조적으로 서로 다른 두 개의 키를 이용하는 비대칭 방식이라는 것이다. 두 개의 키(공개키-개인키)를 이용하므로 기밀성, 키 분배, 인증 분야에서 매우 성능이 뛰어나다는 평가를 받고 있다. 이러한 맥락에서 어느 분야보다 보안이 중요한 인터넷뱅킹이 이 메커니즘을 이용해 보안 장치로 사용하는 것이다. 공개키 암호 구조는 〈표 4〉의 핵심요소로 설명할 수 있다.

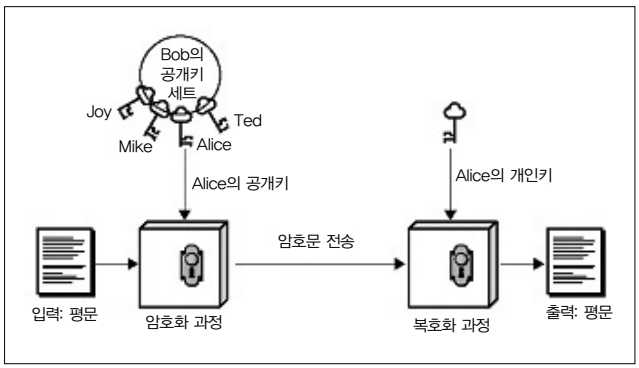
평문(Plain text)	사람이 읽을 수 있는 메시지나 데이터로서 알고리즘의 입력으로 사용된다.
암호 알고리즘	암호 알고리즘은 평문을 여러 가지 형태로 변환시킨다.
공개키와 개인키	한 쌍의 키로 이뤄졌고 한 개는 암호화에 사용되고 다른 하나는 복호화에 사용된다. 암호 알고리즘에 의한 변환은 입력으로 사용되는 공개키나 개인키를 이용해서 이뤄진다.
암호문(Cipher text)	출력으로 나오는 메시지이다. 암호문은 평문과 키에 의해 생성된다. 주어진 동일한 메시지에 대해 서로 다른 두 개의 키를 적용할 경우 출력되는 암호문은 서로 다르다.
복호 알고리즘	평문을 암호화 할 때 사용한 키에 대응하는 키를 이용하여 암호문을 원래의 평문으로 변환하는 알고리즘이다.

〈표 4〉 공개키 암호 구조의 핵심요소

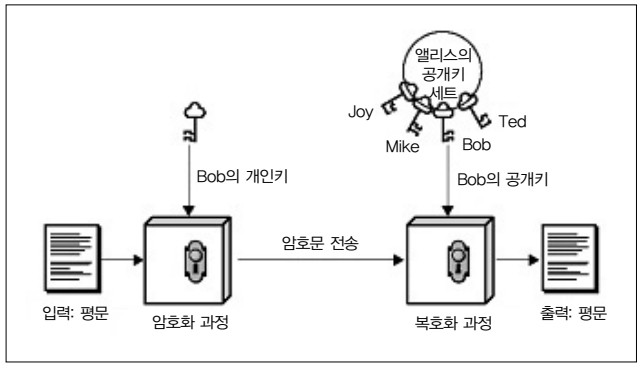
공개키와 개인키의 이름에서 알 수 있듯이 공개키는 다른 사람에게 공개하고, 개인키는 소유자만 알게 된다. 개인키(Private Key)와 전통적인 암호화에서 사용하는 비밀키(Secret Key)를 구분하기 바란다. 일반적인 공개키 암호 알고리즘에서는 이런 식으로 두 개의 비대칭 키를 사용하며, 한 키는 암호화에 사용하고 그 키에 대응하는 다른 한 키는 복호화에 사용한다.

공개키 암호 시스템의 범주

공개키 암호 시스템은 세 가지 범주로 구분한다. 첫째는 암호/복호 모델, 둘째는 디지털 서명, 셋째는 키 교환 이다. 〈그림 8〉을 보며 암호/복호 모델부터 살펴보자. 알고리즘으로는 RSA를 사용했으며 암호화(Encryption)시에는 여러 개의 공개키 중에 수신측 공개키인 Alice의 공개키로 암호문을 생성한다. 생성한 암호문을 Alice에게 전송하면 오직 Alice의 개인키로만 암호문을 평문으로 복호화 하는 시스템이다.



〈그림 8〉 공개키 시스템을 사용한 암호/복호 모델



〈그림 9〉 공개키 시스템을 사용한 전자서명(인증) 모델

다음은 두 번째 범주에 해당하는 디지털서명(인증)에 대해 〈그림 9〉를 보면서 살펴보자. 첫 번째 모델과는 반대로 개인키를 이용하여 암호화한다. 〈그림 9〉에서는 Bob의 개인키로 암호화를 했으며, 복호화를 할 수 있는 사람은 Bob 자신만이 아닌 본인이 공개한 키를 소유한 어떠한 사용자라도 복호화가 가능하다. 결과적으로 본인이 배포한 공개키를 소유한 사용자만 복호화가 가능하다는 것은 수신자에 대한 인증을 의미한다. 본인만의 개인키로 전자서명을 한 문서는 본인이 배포한 공인인증서를 발급받은 사람만 문서를 확인할 수 있다. 인터넷뱅킹의 전자서명 부분을 설명할 수 있는 모델이라고 볼 수 있다. 세 번째 범주인 키 교환에 대해서는 RSA와 Diffie-Hellman을 다루면서 정리하겠다. 〈표 5〉에는 알고리즘에 대한 모델의 지원 여부를 표시했다. RSA는 구현이 단순하고 모든 모델을 지원하기 때문에 가장 널리 쓰인다.

알고리즘	암호/복호 모델	디지털 서명 모델	키 교환
RSA	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No
타원곡선	Yes	Yes	Yes

〈표 5〉 알고리즘에 따른 공개키 시스템 지원 모델

실제 알고리즘을 분석하기 전에 Diffie와 Hellman이 주장한 공개키 알고리즘의 요구 조건을 정리하고 넘어가겠다. 먼저, 한 쌍의 키를 생성하는 것은 계산적으로 단순해야 한다. 또한 공개키를 알고 있는 공격자가 개인키를 알아내는 것이 계산적으로 불가능해야 한다. 어느 시스템에나 적용되는 기본적인 조건일 수 있지만, 기본적으로 적용되는 만큼 가장 중요하다.

인터넷 뱅킹 사례로 살펴본 공개키 암호화 시스템

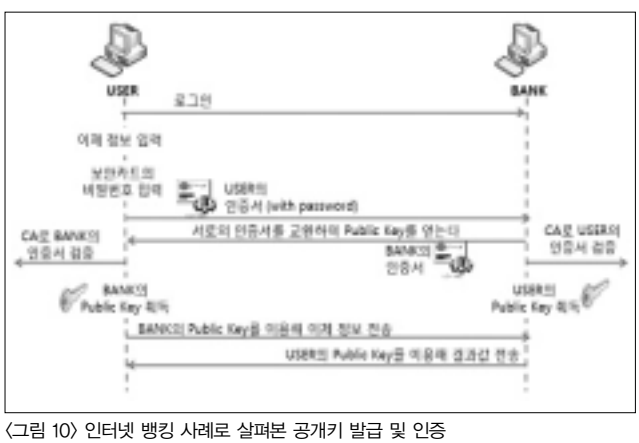
은행의 공인인증센터에서 발행하는 인증서는 X.509 표준을 따

른다. 〈표 6〉에서 인증서 구조를 확인해보자. 공인인증서의 등록 정보에 표시된 것과 동일한 구조임을 확인할 수 있다. Signature algorithm identifier 항목은 사용한 해쉬 알고리즘과 매개변수를 기록하며, Subject's public-key information 항목에는 사용한 공개키 알고리즘과 매개변수/공개키 등의 정보를 기록한다. Signature 항목은 다른 필드 전체로 메시지 다이제스트를 만들고 CA(공인인증기관)의 개인키(Private Key)로 암호화한 내용을 기록한다.

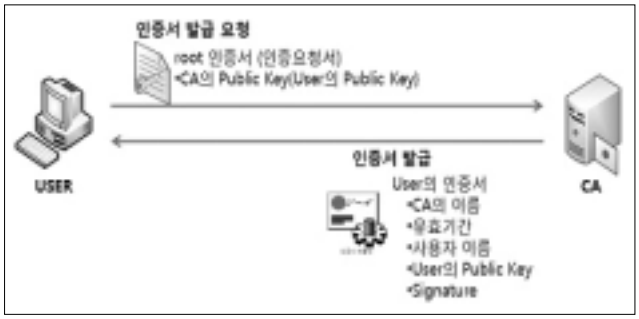
Version
Serial number
Signature algorithm identifier
Issuer name
Period of validity
Subject name
Subject's public-key information
Issuer unique identifier(Optional)
Subject unique identifier(Optional)
ExtensionsSignature

〈표 6〉
X.509 인증서 구조

〈그림 10〉을 살펴보며 인터넷 뱅킹의 인증서 요청 및 발급과정을 알아보자. 먼저 사용자(User)가 은행의 인터넷 뱅킹 웹사이트에 로그인을 한다. 로그인 후 계좌이체 절차를 진행한다. 이체에 필요한 정보를 입력하고 보안카드의 비밀번호를 입력한다. 이때, CA로부터 발급받은 사용자 공인인증서와 은행의 공인인증서를 상호 교환한다. 이와 같은 방식이 상호 인증을 처리하는 과정이다. 다음은 사용자와 은행의 인증서를 각각 CA(공인인증센터)에 검증을 의뢰한다. 이 과정에서 검증이 완료되면 은행과 사용자의 공개키를 획득하게 되어 거래 은행에 이체가 이루어진다. 인증서를 CA로부터 발급받는 과정을 좀 더 상세화하면 〈그림 11〉과 같다. 사용자와 CA의 관계에서 인증서를 요청하고 발급받는 절차를 명확히 하길 바란다. 이렇게 발급 받은 것이 공인인증



〈그림 10〉 인터넷 뱅킹 사례로 살펴본 공개키 발급 및 인증



〈그림 11〉 인증서 발급 절차

서이며 〈그림 10〉과 같이 은행의 공인인증서와 상호교환 인증이 이뤄진다. 단 〈그림 11〉에서 인증서를 발급 요청하는 부분에서 User의 Public Key 항목은 일반적인 은행거래에서는 공란(Empty)상태로 한다.

RSA 알고리즘

1977년에 MIT에서 Rivest 등에 의해 고안되었다. 공개키 시스템에서 가장 널리 쓰이는 알고리즘이기도 하다. 오일러(Euler)함수와 Mod 연산을 주로 사용한다. 간단한 예제를 통해 알고리즘을 살펴보자. 주로 수학적인 접근방식에 가깝기 때문에 의사코드는 생략한다.

〈리스트 3〉 RSA의 암호/복호 알고리즘

```
C(암호문) = Memod(n)
M(평문) = Cdmod(n)=Medmod(n)
공개키(KU) = {e,n}, 개인키(KR) = {d,n}
```

〈리스트 3〉을 보면 공개키와 개인키를 이용해 암호문과 평문을 어떻게 얻을 수 있는지를 보여준다. 송신측은 공개키를 이용해 암호문을 만들며, 수신측은 개인키로 암호문을 평문으로 복호화한다. 이때 사용한 연산은 mod 연산이다. 하지만 여기서 중요한 것은 어떻게 공개키와 개인키를 생성하느냐다. 가장먼저 할 것은 임의의 두 소수 p와 q를 선택하는 것이다. 〈리스트 3〉의 n은 p와 q의 곱으로 정해진다. 즉, n=pq 이다. 다음으로 n보다 작으면서 n과 서로소인 양의 정수의 개수인 오일러함수라고 알려진 $(n)을 계산한다. 결국 오일러공식에 의해 $(n)은 (p-1)(q-1)의 결과 값에 해당한다. $(n)를 계산했으면, 이 값과 서로소인 정수 e를 선택한다. 즉 $(n)과 e는 최대공약수가 1이다. 〈리스트 3〉에서의 e가 바로 공개키에 해당한다. 여기까지해서 공개키를 생성하는데 성공했다. 다음으로 $(n)을 mod로 하는 e의 곱에 대한 역원 d를 계산하면, 개인키도 구해진다. 수학적으로 증명하진 않았지만, 키를 생성하는 과정을 살펴보면 기본적인 원리 자체가 간단함을 알 수 있다. 이러한 알고리즘이 인터넷 뱅킹의 안전을

보장하게 될 줄은 생각도 못했을 것이다. 물론 위의 키 생성방법은 정수의 경우로 예시를 보인 거라 더 간단해 보일 수 있다. 그러나 실제 문자열의 경우도 아스키코드로 생각하면 이와 같은 원리로 간단하게 접근할 수 있다. 〈리스트 4〉에 키 생성과정을 식으로 정리해놓았다.

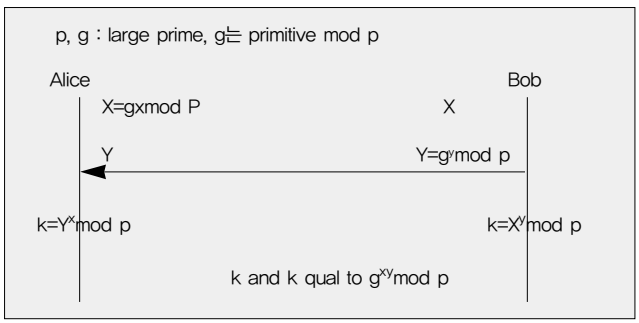
〈리스트 4〉 RSA의 키 생성 알고리즘

p,q 선정	p와 q는 모두 소수이다.
n = p x q	
$(n) = (p-1)(q-1)를 계산	오일러함수
정수 e를 선택	gcd($(n), e)=1; 1< e < $(n)
(gcd: 최대공약수)	
d를 계산	demod($(n)) = 1
공개키	KU = {e, n}
비밀키	KR = {d, n}

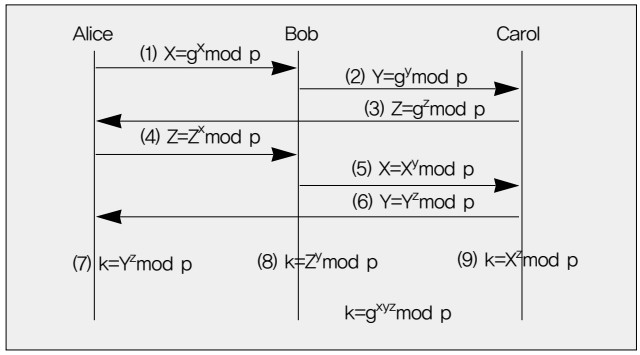
Diffie-Hellman 알고리즘

이 알고리즘은 주로 키 교환에 사용되며, Diffie와 Hellman에 의해 공개키 암호를 정의하는 논문에 처음으로 등장했다. 그 이후 다수의 상업용 제품들이 이 키 교환 기술을 이용한다. 이 알고리즘의 목적은 두 사용자가 키를 안전하게 교환해서 메시지를 암호화 하는데 쓰인다. 알고리즘 자체는 키를 교환하는데 국한되어서 사용됐다. 이 알고리즘은 이산대수문제에 근거하고 있는데 이 산대수문제를 요약하면 다음과 같이 설명할 수 있다. 우선 소수 p의 원시근(Primitive root)을 정의한다. 원시근이란 자신의 거듭제곱을 이용하면 1부터 p-1까지의 정수들을 모두 생성해 낼 수 있는 수를 의미한다. 즉 어떤 수 a가 수소 p의 한 원시근이라면, 다음 수들 {amodp, a2modp,...ap-1modp}은 서로 다르고 정수 1부터 p-1 까지의 수를 치환해 놓은 것과 같다. 즉 집합적으로 보면 같은 집합이고 p보다 작은 임의의 정수 b와 p의 원시근 a에 대해서 b=aimodp를 만족하는 유일한 지수 i를 찾을 수 있다. 너무 수학적인 내용이라 이해가 안갈 수도 있을 것이다.

〈그림 12〉를 보며 원리를 살펴보자. p와 g는 사전에 양쪽 단말의 사용자가 알고 있다고 가정해야 한다. 사실 이 부분이 Diffie-



〈그림 12〉 Diffie-Hellman의 키 교환 시퀀스(2명)



〈그림 13〉 Diffie-Hellman의 키 교환 시퀀스(3명 이상의 경우)

〈리스트 5〉 Diffie-Hellman 키 교환 예시

```
Alice와 Bob이 prime number p=23 and base g=5를 선택한다.
Alice는 비밀값으로 x값을 6 선택. Bob에게 X를 계산하여 전송 (X=gx mod p)
X: 56 mod 23 = 8.
Box은 y값을 15 선택, Alice에게 Y를 전송 (Y=gy mod p)
515 mod 23 = 19.
Alice 는 k를 구한다. k=(Yx mod p)
k: 196 mod 23 = 2.
Bob k'을 구한다. k'=(Xy mod p)
k': 815 mod 23 = 2.
결과적으로 k=k'
```

Hellman의 한계이기도 하다. Alice는 랜덤하게 p보다 크지 않은 x를 선택해 수식에 의해 X를 구한다. 이렇게 구한 X를 Bob에게 전송한다. Bob도 마찬가지로 랜덤하게 p보다 크지 않은 y를 선택해 수식에 의해 Y를 구해 Alice에게 전송한다. 이 과정에서 k와 k'라는 키를 구할 수 있다. Alice는 k라는 키를 갖게 되고, Bob은 k'라는 키를 갖는다. 결과적으로 두 개의 키는 같은 값이 된다. Alice와 Bob의 최종 키가 같은 것으로 두 사용자의 안전한 인증을 보장해줄 수 있다. 키 생성의 예제로 〈리스트 5〉에 정리해 보았다. Alice와 Bob이 p와 g를 각각 23과 5로 선택하고 비밀 값을 주고받으며 결과적으로 생성된 키가 동일하다는 것을 확인해 볼 수 있다. Diffie-Hellman 키 교환 인증방법은 두 사용자 간에만 국한된 것은 아니다. 〈그림 13〉은 3명 이상인 경우를 처리하는 방법을 제시한다.

이렇게 해서 메시지 인증과 공개키 암호화에 대한 기본적인 원리를 알아봤다. 이정도의 원리만 모두 이해했다면, 이를 이용한 애플리케이션을 이해하는 것은 쉬운 일이다. 다음 호에서는 이를 응용한 애플리케이션을 살펴본다.

참고 자료
1. Network Security Essentials 2 nd edition(William Stalling)
2. Cryptography and Network Security 4th edition(William Stalling)
3. Network Security :Private Communication in a Public World(PHPTR)
4. RIPEMD(http://homes.esat.kuleuven.be/~bosselae/ripemd160.html)