정보보호개론

제5장 암호프로토콜 기초 설계 기법

1. 여분 정보

여분 정보 기법을 이해하기 위해서는 블록 암호의 특성을 이해하고 있어야 한다. 블록 암호의 특성에 대해서는 4장에서 이미 간단히 설명한 바 있지만 여분 정보 기법에 대한 이해를 높이기 위해 조금 더 구체적으로 살펴보고자 한다. 참고로 인증 암호화가 보편화되면서 여분 정보의 필요성은 줄어들고 있다.

1.1 블록 암호의 특성

블록 암호는 항상 블록 크기 단위로 메시지를 암호화하며, 입력과 출력의 크기가 같다. 따라서 암호화할 데이터가 블록 크기보다 작으면 채우기가 필요하고, 블록 크기보다 크면 암호화 모드를 사용해야 한다. 이와 관련 더 자세한 내용은 8장에서 설명한다. 이 절에서는 2장에서 언급한 ECB 모드를 이용하여 암호화한다고 가정하고 설명한다.

크기가 블록 크기와 같은 임의의 서로 다른 두 메시지 M과 M'을 다음과 같이 대칭키 K로 암호화하였다고 하자.

$$C = E.K(M), C' = E.K(M')$$

이 경우 C와 C', M과 M' 간에는 어떤 상관 관계도 존재하지 않는다. 또 C나 C'을 $K'(\neq K)$로 복호화하면 그 결과는 예측할 수 없다. 즉, 잘못된 키로 복호화하였을 때 그 결과는 예측할 수 없다. 암호화된 이후 C와 C'이 조작된 경우(일부 비트가 변경된 경우), 이를 K로 복호화한 결과도 예측할 수 없다.

M이 랜덤한 값이면 K로 C를 복호화하더라도 그 결과로부터 C가 K를 이용하여 생성한 암호문인지 알 수 없다. 하지만 M의 일부가 복호화한 사용자가 확인할 수 있는 값이고, 복호화한 후에 그것이 확인되면 C가 K를 이용하여 생성한 암호문인지 확인할 수 있으며, 복호화하여 얻은 M의 무결성에 대해 확신할 수 있다. 그 이유는 확인해야 할 정보가 일정한 비트 크기 이상이면 그것이 우연히 기대한 값과 일치할 확률은 무시할 수 있으며, 블록 암호의 특성 때문에 일부분만 올바르게 복호화될 수 없기 때문이다. 이처럼 암호문을 생성할 때 사용한 암호키를 확인할 수 있게 해주는 평문의 요소를 **여분 정보**라 한다.

블록 크기보다 큰 메시지를 ECB 모드를 사용하여 암호화하고자 하면 메시지를 먼저 블록 크기로 나누어야 한다. 이때 메시지가 정확하게 블록 크기로 나누어지지 않으면 마지막 블록은 채우기를 하여 블록 크기로 만들어 암호화한다. 보통 표준 채우기 방법을 사용하면 정확하게 블록 크기로 나누어지더라도 채우기를 한다. 이에 대해서는 8장에서 다시 자세히 설명한다. 여기서는 크기가 정확하게 블록 크기의 2배인 메시지 $M(= M_1||M_2)$을 채우기 없이 다음과 같이 ECB 모드로 암호화하는 경우를 생각하여 보자.

$$E.K(M = M_1||M_2) = C = C_1||C_2 = E.K(M_1)||E.K(M_2)$$

M_1과 M_2에 복호화한 사용자가 확인할 수 있는 여분 정보가 있어야 C_1과 C_2가 모두 K로 암호화하여 만들 블록임을

확신할 수 있다. 둘 다 여분 정보가 있어도 복호화한 M_1과 M_2를 연결할 수 있는 정보가 없으면 C_1과 C_2가 하나의 메시지 M을 암호화하여 만든 것임을 확신할 수 없다. 실제 통신으로 전달되는 메시지는 공격자에 의해 다양하게 조작될 수 있다. 일부 비트를 조작할 수 있고, 일부 블록을 다른 블록으로 교체할 수 있다.

예를 들어 다음과 같은 암호문을 생각하여 보자.

$$\{B||K_{AB}\}.K_{AS} = C_1||C_2, \{D||K_{AD}\}.K_{AS} = C_1'||C_2',$$

여기서 사용자 식별자, 대칭키의 길이는 16byte이고, 사용하는 대칭 암호알고리즘의 블록 크기도 16byte라 하자. Alice는 수신한 암호문에 B 식별자를 기대한다고 하였을 때, 다음과 같은 4 종류의 메시지를 받았다고 가정해 보자.

(1) $C_1||C_2'$

(2) $C_1'||C_2$

(3) $C_1||X$, X: C_2의 몇 비트를 조작한 것

(4) $Y||C_2$, Y: C_1의 몇 비트를 조작한 것

(1)과 (3)의 경우 Alice는 기대하는 B를 얻게 되지만 (2)와 (4)는 기대하는 식별자를 얻지 못하므로 문제가 있다는 것을 알 수 있다. 하지만 (1)과 (3)의 경우도 두 번째 암호문 블록이 조작 또는 교체되었다는 것은 알 수 없다. 수신한 것이 원래 하나의 메시지를 암호화한 것인지 확신할 수 없기 때문에 어떤 암호 블록을 복호화하여 확인한 내용이 다른 블록을 해석하는데 아무런 영향을 줄 수 없다. 예를 들어 (2)의 경우 Alice는 K_{AB}를 B가 아닌 D와 사용하기 위한 키로 해석하게 된다. 즉, 공격자의 조작에 의해 바인딩이 올바르게 해석되지 않을 수 있다는 것을 알 수 있다.

또 다른 예를 생각하여 보자.

$$\{N_A||B||K_{AB}\}.K_{AS} = C, \{N_A||K_{AB}||B\}.K_{AS} = C', \{K_{AB}||N_A||B\}.K_{AS} = C''$$

여기서 난스의 크기는 8byte이고, 나머지는 이전 예와 동일하다고 가정하자. Alice는 각 암호문을 복호화하여 N_A와 B를 확인하였고 채우기(채우기도 여분 정보에 해당함)가 문제가 없었다고 하였을 때, 3개의 암호문 중에 K_{AB}의 무결성을 확인할 수 있는 것이 있는지 생각하여 보자. 평문 크기는 모두 40byte이므로 마지막 평문 블록에 8byte 채우기를 추가하여 암호화한다. 따라서 3개의 암호문은 총 3개의 암호 블록으로 구성된다. C''에서 M_1이 K_{AB}이므로 나머지 2개 평문 블록의 확인은 M_1 해석에 아무런 영향을 줄 수 없다. C의 경우 K_{AB}가 M_2와 M_3에 나누어져 있고, C'의 경우에는 M_1과 M_2에 나누어져 있다. Alice는 N_A와 B를 확인하였고, 채우기도 확인하였기 때문에 C에서 C_2와 C_3, C'에서 C_1과 C_2는 조작되지 않았다고 생각할 수 있다. 조작되지 않았기 때문에 K_{AB}의 무결성을 확신할 수 있다고 생각할 수 있다. 하지만 그렇지 않다.

다음을 생각하여 보자.

$$\{N_A||B||K_{AB}\}.K_{AS} = C = C_1||C_2||C_3, \{N_A'||B||K_{AB}'\}.K_{AS} = X = X_1||X_2||X_3$$

공격자가 $C_1||C_2||X_3$를 C 대신에 Alice에게 주었다고 생각하여 보자. 그러면 여전히 N_A, B를 확인하게 되며, 채우기도 문제 없는 것으로 확인하게 된다. 하지만 얻게 되는 키는 K_{AB}의 왼쪽 반과 K_{AB}'의 오른쪽 반을 결합한 키를 얻게 된다. C'의 경우도 비슷한 공격이 가능하다. 이와 같은 문제는 ECB 모드 때문제 더 악화된 측면이 있지만 평문의 구성에 따라 다른 암호화 모드를 사용할 경우에도 나타날 수 있는 문제이다. 8장을 학습한 후에 그 장에서 소개한 암호화 모드에서 유사한 문제가 발생하는지 실제 살펴보면 암호화 모드를 이해하는데 큰 도움이 될 것이다.

다음과 같이 메시지를 인증 암호화하면 이와 같은 문제가 여전히 존재하는지 살펴보자.

$$C = E.K_1(M), \mathrm{MAC}.K_2(C)$$

인증 암호화를 하면 MAC 값이 확인된 경우에만 복호화를 시도한다. 또 MAC 값이 확인되면 C를 복호화하지 않아도 상대방을 신뢰할 수 있다면 C가 어떤 키로 암호화된 것인지 확신할 수 있다. 프로토콜에서 각 메시지에 사용할 MAC키에 대응되는 암호화키가 정해져 있으므로 MAC이 확인되고 전송자를 신뢰할 수 있으면 그 MAC의 입력인 C도 프로토콜에서 정한 키로 암호화되어 있을 것이다. 또 MAC 값이 확인되면 C가 전송 과정에서 조작되지 않았다는 것을 확신할 수 있다. 앞서 살펴본 두 가지 예에 있는 것과 달리 인증 암호화를 하면 공격자의 조작은 MAC 값 때문에 모두 발견할 수 있다. 인증 암호화를 하면 메시지의 일부를 조작, 교체하는 것은 가능하지 않지만 전체를 과거나 다른 세션에 있는 것으로 교체를 할 수 있다. 따라서 인증 암호화를 한다고 모든 보안 문제가 해결되는 것은 아니다.

지금까지 살펴본 블록 암호의 특성을 요약하여 보자. 인증 암호화를 하지 않는 경우 여분 정보가 없으면 해당 암호 블록이 어떤 키로 암호화되었는지 알 수 없다. 또 전송 과정에서 조작된 것을 발견할 수 없으므로 바인딩을 위해 함께 암호화된 것이 실제 바인딩되었다고 확신하기 어렵다. 반면에 인증 암호화를 하면 전송 과정의 조작은 발견할 수 있다. 상대방을 신뢰할 수 있다면 여분 정보 없이 암호문이 어떤 키로 암호화되었는지 확신할 수 있고, 조작되지 않았으므로 바인딩에 대해서도 확신할 수 있다. 상대방에 대한 신뢰가 필요한 이유는 상대방이 애초에 메시지를 엉뚱하게 구성할 수 있기 때문이다. ECB 모드가 아닌 다른 암호화 모드를 사용하면 해석이 조금 달라질 수 있지만 큰 틀은 변하지 않으며, 전체 메시지가 조작되지 않았다는 것을 확인하기 위해서는 여전히 인증 암호화가 필요하다.

이 장에서 여러 기법을 소개하면서 인증 암호화를 한다고 구체적으로 제시하지 않고 암호문을 제시할 수 있다. 이 경우 앞서 설명한 것처럼 조작 공격에 의해 바인딩을 해석할 때 문제가 발생할 수 있지만 바인딩에 문제가 없다고 가정하고 설명한다. 이렇게 하는 것이 기법을 이해하는데 복잡성을 줄일 수 있기 때문이다.

1.2 여분 정보 기법

평문 블록에 복호화한 사용자가 기대하거나 알 수 있는 요소가 없다면 이 암호 블록이 원래 어떤 키로 암호화된 것인지 알 수 없다. 반대로 복호화하였을 때 사용자가 확인할 수 있는 요소가 있고 그것이 확인되면 맞는 키로 복호화한 것이 되며, 얻은 평문 블록은 그 암호 블록에 대응된 블록이 맞다는 것을 확신할 수 있다. 우리는 이와 같은 요소를 여분 정보라 한다. 여분 정보라 하면 불필요한 정보라고 생각할 수 있으며, 인증 암호화를 하면 실제 필요 없을 수 있다. 하지만 오직 여분 정보 역할만 하기 위해 포함하는 요소는 거의 없다. 키의 용도나 최근성 등 메시지 의미를 정확하게 하기 위해 꼭 필요한 요소인데, 그 값 자체의 특성이나 사용하는 방법에 의해 여분 정보 역할까지 하고 있는 것이다.

여분 정보는 크게 명백한(explicit) 여분 정보와 함축적(implicit) 여분 정보로 구분한다. 명백한 여분 정보란 사용자 식별자처럼 누구나 확인할 수 있는 정보를 말한다. 따라서 명백한 여분 정보는 암호 해독을 용이하게 해주는 부작용이 있다. 그러므로 가급적 명백한 여분 정보를 사용하지 않는 것이 바람직하지만 키의 용도와 같이 다른 측면의 안전성을 위해 필요하므로 명백한 여분 정보를 전혀 사용하지 않도록 프로토콜을 설계하는 것은 어렵다.

무결성을 제공하기 위해 $\{M||H(M)\}.K$처럼 평문의 해시값을 함께 암호화할 수 있다. 이 경우 이 해시값은 명백한 여분 정보이다. 따라서 누구나 복호화의 정확성을 확인하기 위해 이 해시값을 활용할 수 있다. 명백한 여분 정보로 활용할 수 없도록 MAC을 사용하는 것을 생각해 볼 수 있다. 물론 M에 여분 정보가 포함되어 있다면 이와 같은 시도는 의미가 없다.

그런데 $\{M||\mathrm{MAC}.K(M)\}.K$처럼 암호화키와 MAC키가 같으면 여전히 이 MAC값은 명백한 여분 정보이다. 키를 추측하여 복호화한 공격자는 얻은 M과 추축한 키를 이용하여 MAC을 계산하여 얻은 MAC과 비교하여 추측이 맞았는지 확인할 수 있다. 서로 다른 두 개의 독립적인 키를 암호화키와 MAC키로 사용하면 앞서 살펴본 추측 공격이 가능하지 않다. 하지만 보통은 독립적인 키이지만 하나의 키로부터 계산된 키를 사용하는 경우가 많다. 이 경우에는 MAC값이 여전히 명백한 여분 정보에 해당한다. 공격자는 하나의 키를 추측한 후에 이로부터 두 개의 키를 계산하고 이들을 이용하여 자신의 추측이 맞았는지 확인할 수 있다. 보통 메시지의 무결성을 제공하고 싶으면

해시값이나 MAC값을 함께 암호화하는 것보다 encrypt-then-mac 방법의 인증 암호화를 사용하는 것이 더 일반적인 방법이다.

함축적 여분 정보는 송신자와 수신자만이 확인할 수 있는 여분 정보이다. 예를 들어 A와 B가 비밀정보 I_{AB}를 공유하고 있을 때, A가 B와 공유한 K_{AB}를 이용하여 $\{H(I_{AB})||...\}.K_{AB}$를 B에게 전송하면 $H(I_{AB})$는 B만 확인할 수 있는 함축적 여분 정보이다. 하지만 나머지 부분에 명백한 여분 정보가 들어 있다면 이 값은 아무런 효과를 발휘하지 못한다. 여기서 비밀 정보 대신에 그것의 해시값을 사용하는 이유는 해시값이 노출되어도 일방향성 특성 때문에 비밀 정보는 노출되지 않도록 하는 효과가 있으며, 비밀 정보가 명백한 여분 정보인 경우에도 함축적 여분 정보로 사용할 수 있도록 해주는 효과도 있다.

2. 명명 기법

명명 기법이란 **메시지의 의미를 명확하기 위해 참여자의 식별자를 암호문 내에 포함하는 기법**을 말하며, 이미 3장에서 제시한 키 확립 프로토콜의 세 번째 시도 프로토콜부터 확립하는 세션키의 용도를 나타내기 위해 사용한 기법이다. 명명 기법을 사용하는 이유는 메시지를 통해 전달하고자 하는 의미는 해당 메시지에 명백하게 나타나야 하기 때문이다[1]. 만약 나타나지 않으면 그 부분을 공격에 활용할 가능성이 높아진다.

4장 그림 4.3에 제시된 프로토콜에서 신뢰 기관이 메시지 2의 첫 번째 암호문 $\{B||K_{AB}\}.K_{AS}$를 통해 A에게 전달하고 싶은 것을 글로 표현하면 "A와 B가 사용하기 위한 새로운 키 K_{AB}를 비밀로 A에게 전달함"과 같다. 이 글에 포함된 새롭다는 것을 제외하고 암호문으로 나타내면 $\{A||B||K_{AB}\}.K_{AS}$와 같다. 이 메시지는 키 K_{AS}로 암호화되어 있기 때문에 A를 위한 것이 명백하므로 이 메시지에서 A는 생략할 수 있다. 어떤 암호키를 사용하여 생성한 암호문의 경우 해당 암호키로부터 유추할 수 있는 것은 평문에 포함하지 않아도 된다.

$\{A||B||K_{AB}\}.K_{AS}$를 공개키만 사용하여 만들면 다음과 같다.

$$\{\{A||B||K_{AB}\}.{-K_S}\}.{+K_A}$$

공개키는 비밀성만 제공할 수 있고, 개인키는 인증만 제공할 수 있으므로 신뢰 기관이 생성한 것임을 나타내기 위해 신뢰 기관의 개인키와 비밀성을 제공하기 위해 A의 공개키를 사용한 것이다. 보통 공개키를 이용하여 인증과 비밀성을 동시에 제공하기 위해서는 메시지를 개인키로 먼저 암호화하고 그 결과를 다시 공개키로 암호화하는 방식을 사용한다. 이때 비밀키를 이용한 암호문처럼 A의 식별자를 생략할 수 있다고 생각할 수 있다. 하지만 A의 식별자를 A의 공개키로 직접 암호화하고 있지 않으므로 생략할 수 없다. 다시 말하면 데이터를 직접 암호화하는 키로부터 유추할 수 있는 것만 생략이 가능하다. 실제 생략을 하면 공격자 C는 B와 비밀 통신하기 위한 키를 받은 후에 해당 메시지의 내부 암호문만 A의 공개키로 암호화하여 A가 자신이 알고 있는 키를 사용하도록 다음 메시지를 전달하여 공격할 수 있다.

$$\{\{B||K_{BC}\}.{-K_S}\}.{+K_A}$$

3. 메시지 최근성 제공 기법

암호프로토콜에서 참여자가 메시지를 수신하였을 때 보통 많이 하는 검사 중 하나가 메시지의 최근성(freshness, timeliness)을 확인하는 것이다. 여기서 메시지는 오해의 소지가 있는 용어의 사용이다. 오히려 이 절에서는 메시지를 암호문으로 바꾸어 이해하는 것이 더 정확할 수 있다. 메시지의 평문 부분은 조작이 가능하기 때문에 암호프로토콜에서 최근성은 암호문의 최근성을 확인하는 것이며, 더 나아가 암호문 내에 있는 세션키와 같은 특정 요소의 최근성을 확인하고 싶은 것이다.

메시지의 최근성을 검사한다는 것은 메시지가 이전 또는 다른 세션에서 사용한 메시지가 아니라 현재 프로토콜 수행을 위해 새롭게 작성한 메시지임을 확인하는 것을 말한다. 메시지의 최근성은 과거 세션에서 사용한 것들만 배제하는 것이 아니라 현재 병행으로 수행되고 있는 다른 세션에서 사용한 것도 배제할 수 있어야 한다. 메시지의 최근성은 메시지의 구성요소로부터 유추해야 하며, 어떤 메커니즘을 사용하든 간에 재전송 메시지에는 포함할 수 없거나 재전송 메시지와 현재 메시지를 구별할 수 있어야 한다. 메시지와 특정 프로토콜 수행과의 연결은 일반적으로 시간적인 관계(temporal relationship) 또는 인과 관계(causal relationship)를 통해 이루어진다. 시간적 관계 또는 인과 관계는 일반적으로 관계를 증명할 수 있는 식별자를 메시지에 포함하므로써 형성된다.

메시지의 최근성을 보장하기 위한 식별자로 가장 널리 사용하는 것은 타임스탬프와 난스이다. 타임스탬프 기반 기법은 시간적 관계를 통해 메시지의 최근성을 보장하는 기법이며, 이 기법은 메시지에 타임스펨프라고 하는 메시지 작성 시간을 포함하여 최근성을 보장한다. 난스 기반 기법은 인과 관계를 통해 메시지의 최근성을 보장하는 기법이며, 이 기법은 어떤 메시지 M에 처음으로 사용한 값을 그 메시지의 응답에 포함하여 응답이 메시지 M 이후에 생성한 것임을 증명함으로써 메시지의 최근성을 보장한다. 타임스템프 기반 기법은 시간적 관계를 이용하기 때문에 병행으로 수행하는 프로토콜에는 같은 시간 정보가 포함될 수 있으므로 타임스템프만으로는 다른 프로토콜 수행과 완벽하게 구분하기는 어렵다.

메시지 최근성을 보장하기 위한 관계를 형성하는 과정에서 참여자는 크게 다음의 3가지 역할을 한다.

- 제공자(supplier): 식별자를 제공하는 참여자
- 입증자(prover): 제공된 식별자를 메시지에 포함하는 참여자
- 검증자(verifier): 메시지에 포함된 식별자를 통해 메시지의 최근성을 확인하는 참여자

제시한 세 역할을 모두 다른 참여자가 하는 것은 아니다. 방식에 따라 제공자와 입증자가 같은 참여자일 수 있고, 제공자와 검증자가 같은 참여자일 수 있다.

어떤 특정한 값이 최근성 식별자로서 적합하기 위해서는 반드시 사용하는 값이 예측할 수 없어야 하는 것은 아니다. 타임스탬프는 본질적으로 예측이 가능한 값이고, 난스 기법의 경우에는 사용하는 방식에 따라 불예측성이 필요할 수 있고 필요하지 않을 수 있다. 하지만 난스의 경우에는 예측이 가능하더라도 반드시 매번 다른 값을 사용하여야 한다.

3.1 타임스탬프 기법

타임스탬프 기법은 다음과 같이 암호문에 현재 시각을 추가하여 암호문의 최근성을 보장하는 기법이다.

$$A \rightarrow B: \ \{T_A||...\}.K_{AB}$$

B는 위 암호문을 수신하면 그것을 복호화한 후에 자신의 지역 시간의 현재 값 T_B와 T_A를 비교하여 두 시각의 차이 $|T_A - T_B|$가 허용할 수 있는 범위 내에 있으면 암호문이 최근에 생성한 것으로 인식하게 된다. 따라서 두 사용자의 지역 시간이 어느 정도 동기화되어 있어야 한다. 보통 두 사용자의 시간을 동기화하기보다는 각 사용자가 절대 시간과 동기화하여 사용자 간에도 동기화하는 방식을 사용한다. 타임스탬프를 유효한 값으로 허용하는 범위를 허용 윈도우(acceptance window)라 하며, 메시지 통신 지연과 시간 동기화(synchronization) 메커니즘에 의해 결정된다.

타임스탬프 기법에서 암호문에 타임스탬프를 포함하는 사용자가 제공자인 동시에 입증자이며, 암호문을 수신하는 사용자가 검증자다. 제공자와 입증자가 같은 사용자이기 때문에 하나의 메시지만을 이용하여 해당 메시지에 포함된 암호문의 최근성을 보장할 수 있다.

타임스탬프 기법의 취약점은 다음과 같다.

- 취약점 1. 허용 윈도우라는 것이 있기 때문에 허용 윈도우 내에는 항상 재전송하는 공격이 가능하다.
- 취약점 2. 타임스탬프의 형태는 공개된 것이므로 제공자는 쉽게 특정 순간을 가리키는 타임스탬프를 메시지에 포함할 수 있다.
- 취약점 3. 시스템 시간이 보호해야 하는 중요한 자산이 된다.

취약점 1을 방어하기 위해서는 메시지 중복 검사가 필요하다. 이를 위해 허용 윈도우의 길이가 t초이면 지난 t초 내에 수신한 모든 메시지를 보관하고 있어야 한다. 취약점 2를 방어할 수 있는 수단은 많지 않다. 전자서명 중재 방식처럼 별도 신뢰 기관을 사용하여 타임스탬프 값을 확인하는 방식을 생각할 수 있지만 모든 상황에 적용할 수 없을 뿐만 아니라 하나의 메시지로 최근성을 보장할 수 있다는 타임스탬프의 장점이 없어지는 문제점도 있다. 따라서 검증자는 제공자가 정직하게 현재의 타임스탬프 값을 메시지에 포함한다고 신뢰할 수 있어야 한다. 하지만 이 신뢰 관계는 보통 성립하지 않는다. 따라서 타임스탬프를 이용하는 프로토콜에서는 이 값의 고의적 변경이 어떤 문제를 야기할 수 있는지 살펴보아야 한다.

취약점 3 때문에 다음과 같은 문제가 발생할 수 있다. 타임스탬프에서 제공자나 검증자의 시간이 오류나 공격에 의해 현재 시간과 다를 수 있다. 검증자의 시간이 미래 시간인 경우와 제공자의 시간이 미래 시간인 경우를 생각하여 보자. 전자를 선행(predated) 메시지라 하고, 후자를 사후(postdated) 메시지라 한다[2]. 두 경우 모두 검증자는 수신한 메시지가 허용 윈도우를 벗어나기 때문에 해당 메시지를 거부한다. 하지만 사후 메시지는 해당 미래 시간이 되면 유효한 메시지가 되기 때문에 공격자가 이를 보관한 후 메시지가 유효해지는 시점에서 사용하는 공격에 대해서는 방어할 방법이 없다. 물론 사용자의 시간을 미래 시간을 모두 바꾸면 방어가 가능하지만 사용자의 시간은 다른 응용들 때문에 절대 시간과 동기화해야 하므로 이와 같은 방법은 사용하기 어렵다.

이와 같은 취약점이 있음에도 불구하고 타임스탬프은 제공자와 입증자가 같기 때문에 하나의 메시지만을 이용하여 그 메시지의 최근성을 보장할 수 있다는 장점이 있다. 타임스탬프의 단점은 앞서 살펴본 바와 같이 사용자 간의 시간 동기화가 필요하며, 이 동기화는 절대 시간과의 동기화를 통해 이루어진다. 이 때문에 시스템 시간이 매우 중요하게 보호되어야 하는 자산이 된다. 특히, 사후 메시지를 해결할 방법이 없으므로 공격자들이 시스템에 침입하여 시간을 변경할 수 없도록 하여야 한다.

3.2 난스 기법

난스 기법은 4장 키 확립 프로토콜 네 번째 시도부터 사용한 기법으로 제공자가 메시지에 난스를 포함하여 입증자에게 전달하면 입증자는 최신성을 보장하고자 하는 암호문에 해당 난스를 포함하여 회신하는 방식이다. 이 방식에서 제공자와 검증자는 같고 이들과 입증자는 다르다. 따라서 특정 메시지의 최근성 보장을 위해서는 2개의 메시지가 필요하다. 이와 같은 방식으로 진행하기 때문에 난스 기법을 시도-응답(challenge-response) 방식을 사용하는 기법이라 한다. 시도-응답 방식이란 메시지 M에 대한 응답 메시지는 오직 M의 내용을 알고 있을 때만 생성할 수 있도록 하는 기법을 말한다. 따라서 시도와 응답 간의 인과 관계가 성립하며, 두 메시지를 바인딩하는 역할을 한다. 보통 난수를 난스 값으로 많이 사용하지만 사용 방식에 따라서는 타임스탬프나 카운터도 난스 값으로 사용이 가능하다. 타임스템프를 난스 값으로 사용하면 그 값을 시스템 시간과 비교하지 않고 이전 메시지에 포함된 값이 회신 메시지에 포함되어 있는지 확인하는 형태가 된다.

난스 기법은 크게 다음과 같은 3가지 방식으로 사용할 수 있다. 방식 1은 아래와 같이 시도를 평문으로 전달하면 입증자는 암호문 내에 난스를 포함하여 암호문의 최근성을 보장하는 방식이다.

$$\begin{aligned} &\text{Msg 1.} \quad A \to B: \quad N_A \\ &\text{Msg 2.} \quad B \to A: \quad \{N_A || ...\}.K_{AB} \end{aligned}$$

방식 2는 아래와 같이 제공자가 난스를 암호문에 포함하여 전달하면 입증자는 난스를 평문으로 보내는 방식이다.

$$\begin{aligned} &\text{Msg 1.} \quad A \rightarrow B: \quad \{N_A||...\}.K_{AB} \\ &\text{Msg 2.} \quad B \rightarrow A: \quad N_A \end{aligned}$$

방식 2는 메시지 2의 최근성을 보장하기보다는 시도 메시지 1의 암호문을 입증자가 최근에 복호화하였다는 것을 입증하는 방식이다. 방식 3은 아래와 같이 제공자도 입증자도 난스를 암호문에 포함하여 교환하는 방식이다.

$$\begin{aligned} &\text{Msg 1.} \quad A \rightarrow B: \quad \{N_A||...\}.K_{AB} \\ &\text{Msg 2.} \quad B \rightarrow A: \quad \{N_A||...\}.K_{AB} \end{aligned}$$

방식 3에서는 두 암호문의 구조나 내용이 동일하면 여러 가지 공격이 가능할 수 있기 때문에 암호문의 구조와 내용을 다르게 하는 것이 중요하다. 방식 3에서 시도 메시지가 암호화되어 있는 이유는 난스의 비밀성을 보장하기 위한 것은 아니고 난스 기법을 활용하는 프로토콜의 필요에 의해 암호화한 것이다. 난스 기법은 대칭, 비대칭 암호알고리즘 외에 다른 암호알고리즘에서도 사용이 가능하다.

위 3가지 방식에서 방식 2의 경우에만 난스의 예측 불가능성이 필요하다. 하지만 계속 강조하지만 난스는 매번 절대 사용한 적이 없는 새로운 값을 사용해야 한다. 실제 응용에서는 의사난수(pseudo-random)를 생성하여 사용하는 경우가 많다. 이렇게 할 경우 중복될 수 있다고 생각할 수 있지만, 범위가 충분하고 균일하게 생성하면 우연히 같은 값을 다시 사용하는 것에 대해 걱정할 필요가 없다. 물론 프로토콜 수행에 참여하는 횟수가 많아지면 우연히 같은 값을 사용할 수 있는 확률은 높아진다. 이 문제는 시기적절하게 사용하는 장기간 키를 갱신하는 것이다. 다른 키를 사용하면 같은 난스를 다시 사용하여도 문제가 되지 않는다.

난스 기법의 최대 장점은 타임스탬프 기법과 달리 입증자를 신뢰할 필요가 없으며, 시간 동기화도 요구하지 않는다. 하지만 메시지의 최근성을 보장하기 위해서는 2개의 메시지가 필요하다. 참고로 연속적인 n개의 메시지의 최근성이 필요하면 필요한 메시지 개수는 $n+1$이므로 필요한 메시지 수가 계속 2배로 증가하는 것은 아니다.

3.3 기타 메시지 최근성 보장 기법

타임스탬프 대신에 카운터를 이용하여 메시지의 최근성을 보장할 수 있다. 카운터는 타임스탬프와 매우 유사하다. 사용자는 자신의 카운터 값을 1 증가한 다음 그 값을 암호문 내에 포함하면 수신자는 암호문 내에 포함된 카운터가 자신의 카운터보다 크면 메시지가 최신에 생성한 메시지로 판단하는 방식이다. 이 기법은 타임스탬프와 마찬가지로 제공자와 입증자가 같은 사용자이며, 검증자가 다른 사용자이다. 따라서 단일 메시지를 통해 메시지의 최근성을 보장할 수 있다.

두 사용자가 카운터를 이용하기 위해서는 두 사용자 모두 카운터를 유지하여야 하며, 사용자 쌍마다 다른 카운터를 사용해야 한다. 한 사용자가 n명과 통신하면 그 사용자는 n개의 카운터를 유지해야 한다. 따라서 일반 사용자 간의 카운터를 사용하기에는 유지해야 하는 카운터의 수 때문에 적절하지 않지만 신뢰 기관은 사용자마다 어떤 정보를 유지하고 있기 때문에 사용자마다 카운터를 하나 추가로 유지하는 것은 큰 문제가 되지 않는다. 따라서 신뢰 기관과 사용자 간에는 카운터 기법의 이용이 충분히 가능하다.

타임스탬프는 앞서 설명한 바와 같이 절대 시간을 활용하여 동기화하기 때문에 사용자마다 하나의 시간만 유지하면 되지만 카운터는 그렇지 못하다. 반대로 타임스탬프는 절대 시간에 구속되기 때문에 사후 메시지 문제가 발생할 수 있지만 카운터는 항상 큰 값으로 동기화가 가능함으로 사후 메시지 문제는 발생하지 않는다. 공격자가 유지하는 카운터 값을 불법적으로 바꾸면 최근성 보장이 정상적으로 이루어지지 않기 때문에 타임스탬프 방식과 마찬가지로 유지하는 카운터가 중요한 보호 자산이 된다.

카운터를 사용할 경우 제한적 크기의 카운터를 사용할 수밖에 없으므로 무한한 카운터가 아니다. 하지만 어느 정도 길이의 비트 값을 사용하면 응용의 수명 시간 내에 카운터 값이 소진되어 반복된다는 걱정은 하지 않아도 된다.

또 키를 바꾸면 다시 카운터를 초기화하여 사용할 수 있다.

지금까지 살펴본 타임스탬프, 난스, 카운터는 메시지에 어떤 값을 포함하게 되며 그 값이 검증자가 기대하는 값이면 최근에 생성한 것으로 믿게 된다. 사용자가 어떤 값이 최근에 생성된 것으로 믿고 있으면 그 이후에 해당 값을 사용한 암호문도 최근에 생성한 것으로 믿을 수 있다. 예를 들어 어떤 키가 최근에 생성한 것이라 믿고 있을 때 이 키로 암호화된 메시지를 수신하면 사용자는 이 암호문의 생성 시점을 예측할 수 있다.

4. 신뢰 관계

4장에 제시된 키 확립 프로토콜의 네 번째 시도에서 S가 A에게 $\{B||N_A||K_{AB}\}.K_{AS}$를 전달한다. A는 이 암호문을 K_{AS}로 복호화하는데 성공하고 바인딩에 문제가 없다고 가정하면 이 메시지는 서버 S가 생성한 메시지임을 확신할 수 있다. 또 암호문 내에 있는 N_A를 확인하면 이 암호문이 최근에 생성한 암호문임을 확신할 수 있다. 하지만 여기서 최근성을 보장하고 싶은 것은 암호문 자체가 아니라 암호문 내에 포함된 세션키이다. 그러나 N_A를 통해서는 암호문 내에 함께 포함된 세션키의 최근성을 직접 확인할 수 없다. 이를 위해서는 신뢰 관계를 활용하여야 한다. 타임스템프 기법을 이용하여도 마찬가지이다.

참여자가 다른 참여자를 신뢰한다는 것은 해당 참여자는 프로토콜에서 정한 약속대로 정직하게 진행한다는 것을 믿을 수 있다는 것을 말한다. 4장에 제시된 키 확립 프로토콜의 네 번째 시도의 약속 중 하나는 서버 S는 사용자의 요청을 받으면 새로운 세션키를 생성하여 준다는 것이다. 사용자가 S를 신뢰하고, 난스를 통해 $\{B||N_A||K_{AB}\}.K_{AS}$가 이번 세션을 위해 생성한 것을 확인하게 되면 이 암호문에 포함된 K_{AB}는 서버가 이번에 생성해 준 세션키임을 확신할 수 있다.

신뢰와 관련된 주장(또는 가정)은 잘못되었다고 말할 수 없지만 적절하지 않다고 말할 수는 있다. 이 때문에 프로토콜에서 신뢰 관련하여 어떤 가정을 사용하는지 분석하고 관계들이 부적절하면 해당 프로토콜을 사용하지 말아야 한다. 보통 프로토콜에 참여하는 이해 당사자들은 서로를 신뢰하기 어렵다. 이 때문에 신뢰 기관과 같은 제3의 중재자를 활용하는 것이다. 하지만 프로토콜에 따라 실제 현장에서는 중재자 역할을 할 기관을 찾는 것이 어려울 수 있다. 따라서 정부가 직접 운영하거나 승인한 기관을 신뢰 기관으로 많이 사용한다.

이와 관련하여 하나의 응용을 살펴보자. 현재 컴퓨팅 환경의 변화로 클라우드 서비스를 활용하는 경우가 많아지고 있다. 특히, 자신의 데이터를 자신의 컴퓨터에 유지하는 것이 아니라 클라우드에 유지하는 경우가 늘어나고 있다. 이때 사용자들은 자신에 데이터에 대해 클라우드를 운영하는 기관을 신뢰할 수밖에 없다. 이 신뢰가 부적절하다고 생각하면 편리성에도 불구하고 자신의 데이터를 외부에 유지하지 않을 것이다.

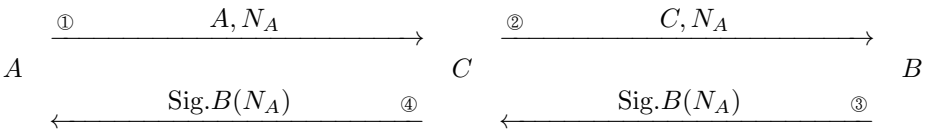
5. 개체 인증

암호프로토콜에서 수신한 암호문을 생성한 참여자의 확인이 중요할 수 있다. 예를 들어 4장에서 살펴본 키 확립 프로토콜의 경우 일반 참여자는 세션키가 포함된 암호문을 서버가 생성한 것인지 확인할 수 있어야 한다. 보통 암호문을 생성할 때 사용한 암호키를 통해 그것을 생성한 참여자를 확인한다. 암호프로토콜에서 참여자의 인증은 크게 강한 개체 인증과 약한 개체 인증으로 구분할 수 있다. 약한 개체 인증은 인증 대상 참여자가 현재 온라인 상태라는 것을 검증자가 확인한 경우를 말하며, 강한 개체 인증은 인증 대상 참여자가 현재 온라인에서 활동하고 있을 뿐만 아니라 검증자를 통신의 상대방으로 인식하고 있다는 것을 확인할 수 있는 경우를 말한다.

개체 인증을 위해 가장 널리 사용하는 암호기술은 전자서명이다. 그림 5.1에 제시된 프로토콜에서 A는 메시지 2의 전자서명 값을 통해 B가 현재 온라인 상태라는 것은 확인할 수 있다. 하지만 자신을 상대방으로 인식하고 메시지를 보낸 것인지 알 수 없다. 예를 들어 그림 5.2과 같은 공격이 이루어졌다고 하자. 이것이 실제 공격이라고

Msg 1. $A \rightarrow B$: A, N_A
Msg 2. $B \rightarrow A$: Sig.$B(N_A)$

<그림 5.1> 약한 개체 인증 프로토콜



<그림 5.2> 약한 개체 인증 프로토콜에 대한 공격

보기 힘들 수 있다. 공격자 C는 A가 전송하는 메시지 1을 차단하고 자신이 대신 이 메시지를 B에게 전달한다. 이 때문에 B는 수신한 난스가 A가 아닌 C가 보낸 것으로 착각하게 된다. 결과적으로 B는 C와 통신하고 있는데 A는 B가 자신과 통신하고 있다고 생각하게 된다.

이와 같은 공격은 강한 개체 인증을 제공하고 있지 않기 때문에 가능한 것이다. 하지만 프로토콜에 대한 간단한 분석으로 이와 같은 허점을 제거할 수 있다. 2자간 프로토콜의 참여자는 프로토콜을 시작하는 참여자(개시자)와 그것에 대해 반응하는 참여자(반응자)로 구분할 수 있다. 이때 프로토콜이 안전하기 위해서는 개시자가 서로 다른 두 명의 반응자와 프로토콜을 수행할 때 두 수행을 구분할 수 있어야 하며, 반대로 반응자 입장에서 서로 다른 두 명의 개시자와 프로토콜을 수행할 때 두 수행을 구분할 수 있어야 한다.

그림 5.1과 그림 5.3을 비교하여 보자. A는 메시지 2의 서명값을 통해 누구와 프로토콜을 진행하는지 구분할 수 있다. 하지만 그림 5.1과 그림 5.4를 비교하면 B 입장에서는 프로토콜을 구분할 수 없다. A와 할 때 보내는 메시지 형태나 내용이 C와 할 때 보내는 것과 차이가 없다. 이를 구분하기 가장 쉬운 방법은 명명 기법을 사용하는 것이다. 그림 5.1의 메시지 2에서 서명값에 상대방 식별자를 포함하면 서로 다른 개시자와 수행하는 프로토콜을 쉽게 구분할 수 있다.

Msg 2. $B \rightarrow A$: Sig.$B(A||N_A)$

실제 위와 같이 프로토콜을 진행하면 A는 B가 온라인에 있다는 것은 물론 B가 자신을 상대방으로 인식하고 있다는 것을 확신할 수 있다. 따라서 위와 같은 메시지를 사용하면 강한 개체 인증이 된다.

프로토콜에서 두 참여자가 서로를 인증할 수 있다면 상호 인증(mutual authentication)을 제공한다고 말하며, 한 참여자만 다른 참여자를 인증할 수 있다면 일방향 인증(unilateral authentication)을 제공한다고 말한다. 이 절에서 제시한 프로토콜은 일방향 인증만 제공하고 있다. 상호 인증과 일방향 인증 중 어느 것을 제공하는지는 응용에서 필요에 따라 결정하면 된다.

6. 공개키 사용 원리

암호프로토콜을 설계할 때 공개키 방식을 사용하면 다음 원리[3]를 충분히 숙지하고 설계해야 한다.

Msg 1. $A \rightarrow C$: A, N_A
Msg 2. $C \rightarrow A$: Sig.$C(N_A)$

<그림 5.3> 약한 개체 인증 프로토콜에 대한 개시자 입장 분석

$$\begin{array}{ll} \text{Msg 1.} & C \rightarrow B: \quad C, N_C \\ \text{Msg 2.} & B \rightarrow C: \quad \text{Sig.}B(N_C) \end{array}$$

<그림 5.4> 약한 개체 인증 프로토콜에 대한 반응자 입장 분석

$$\text{Msg 1. } A \rightarrow B: \ \{\text{Sig.}\text{–}K_A(M)\}.\text{+}K_B$$

(1) 개인키 후 공개키 방법

$$\text{Msg 1. } A \rightarrow B: \ \text{Sig.}\text{–}K_A(\{M\}.\text{+}K_B)$$

(2) 공개키 후 개인키 방법

<그림 5.5> 개인키, 공개키 동시 사용 방법

첫째, 공개키 방식으로 인증과 비밀성을 모두 제공하고 싶으면 개인키로 먼저 암호화한 다음 상대방의 공개키로 암호화하여야 한다. 여기서 개인키로 암호화한다는 것은 첨부 형태 전자서명하는 것을 의미한다. 즉, 그림 5.5.(1)에 기술된 프로토콜은 실제 다음과 같다.

$$\text{Msg 1. } A \rightarrow B: \{M\}.K, \{K\}.\text{+}K_B, \text{Sig.}\text{–}K_A(H(M))$$

여기서 K는 A가 사용한 랜덤키다. 보통 비용 때문에 공개키를 이용하여 일반 메시지를 암호화하지 않는다는 것을 생각해야 한다. 그림 5.5.(2)의 경우 $C = \{M\}.K$일 때, 위 식에서 서명이 $\text{Sig.}\text{–}K_A(H(C))$로 바뀌는 것으로 생각하면 된다.

그림 5.5에 제시된 두 가지 방법을 비교하여 보자. 두 방법은 모두 오직 B만 M을 얻을 수 있다. 즉, 비밀성은 둘 다 보장된다. 하지만 방법 1에서 B는 A가 M을 알고 있다고 확신할 수 있지만 방법 2는 확신할 수 없다. 반면에 방법 2에서 B는 A가 이 메시지를 전송하였다고 확신할 수 있다. 따라서 서로 장단점이 있지만, A가 M을 알고 있다는 것이 보통 더 중요하기 때문에 개인키로 먼저 암호화한 후에 상대방의 공개키로 암호화하는 방법을 더 많이 사용한다.

둘째, 개인키로 서명하거나 복호화할 때 상대방이 자신을 오라클로 활용할 수 있으므로 주의해야 한다. 그림 5.6에 제시된 프로토콜은 공격자가 오라클로 활용할 수 있는 프로토콜이다. 취약점을 살펴보기 전에 프로토콜의 내용을 살펴보면 다음과 같다. B는 난스를 생성하여 A의 공개키로 암호화하여 A에게 전달하면 A는 이것을 복호화하여 회신하는 프로토콜로서 B는 A가 자신의 요청에 대해 최근에 응답하였다고 확신할 수 있다. 이 프로토콜은 사용자의 복호화 능력을 이용하는 프로토콜이다. A의 공개키로 암호화된 것은 오직 A만 복호화할 수 있다.

이 프로토콜의 문제점은 공격자가 A의 공개키로 암호화된 임의의 암호문을 메시지 1로 전달하면 A는 이것을 인식하지 못한 상태로 복호화해줄 수 있다. 따라서 이 프로토콜은 A의 공개키로 암호화된 것을 무조건 복호화해주는 오라클로 활용할 수 있다. 이와 같은 문제는 수신한 메시지의 크기와 복호화하여 얻은 평문이 이 프로토콜의 형식에 맞는지 확인하면 극복할 수 있다.

$$\begin{array}{ll} \text{Msg 1.} & B \rightarrow A: \quad \{N_B\}.\text{+}K_A \\ \text{Msg 2.} & A \rightarrow B: \quad N_B \end{array}$$

<그림 5.6> 오라클에 취약한 프로토콜

실제 그림 5.1에 제시된 프로토콜도 서명 오라클로 활용할 수 있는 프로토콜이다. 해당 프로토콜은 상대방이 전달한 난스를 전자서명하여 회신하고 있다. 만약 공격자가 난스로 특정 메시지의 해시값을 전달하면 그것을 무조건 서명해 줄 수 있다. 이와 같은 오라클 문제는 공개키를 활용할 때만 발생하는 것은 아니다. 따라서 모든 프로토콜을 설계할 때 혹시 결과 프로토콜이 오라클로 활용이 가능한지 살펴보아야 한다. 오라클 문제는 프로토콜 설계 과정뿐만 아니라 소프트웨어 구현 과정에서도 충분한 고려가 있어야 한다. 소프트웨어가 허점 없이 올바르게 구현되어 있으면 프로토콜이 오라클로 활용이 가능해도 실제 활용하는 것이 어려울 수 있다.

참고문헌

[1] M. Abadi, R. Needham, "Prudent Engineering Practice for Cryptographic Protocols," IEEE Trans. on Software Engineering, Vol. 22, No. 1, pp. 6–15, 1996.

[2] Li Gong, "A Security Risk of Depending on Synchronized Clocks," ACM Operating Systems Review, Vol. 26. No. 1, pp. 49–53, Jan. 1992.

[3] Ross J. Anderson, Roger M. Needham, "Robustness Principles for Public Key Protocols," Advances in Cryptology, CRYPTO 95, LNCS 963, pp. 236–247, 1995.

퀴즈

1. 주어진 다음 프로토콜과 관련된 다음 설명 중 틀린 것은?

$$\begin{array}{lll} \text{Msg 1.} & A \rightarrow B: & A, N_A \\ \text{Msg 2.} & B \rightarrow A: & \text{Sig}.B(N_A) \end{array}$$

   ① 약한 개체 인증만 제공한다.
   ② 상호(양방향) 인증을 제공한다.
   ③ 서명 오라클로 활용할 수 있는 문제점이 있다.
   ④ Alice는 Bob의 인증서가 있어야 수신한 서명을 확인할 수 있다.

2. 난스 기법과 관련된 다음 설명 중 틀린 것은?

   ① 난스 기법은 상대방에 대한 신뢰가 필요 없다.
   ② 난스 기법은 시스템 간의 어떤 동기화도 필요 없다.
   ③ 난스 기법에서 사용하는 난스값은 무조건 예측이 가능하지 않아야 한다.
   ④ 난스 기법에서 한 사용자는 제공자와 검증자 역할을 하며, 다른 사용자는 입증자 역할을 한다.

3. 메시지의 최근성 보장 기법 중 타임스템프와 카운터 기법의 차이점을 설명한 다음 내용 중 틀린 것은?

   ① 타임스템프 기법은 시스템 간 클럭 동기화가 필요하지만 카운터 기법은 클럭 동기화가 필요하지 않다.
   ② 카운터 기법은 각 사용자 간의 별도 카운터가 필요하지만 타임스템프 기법은 모든 사용자가 절대 시간과 동기화된 하나의 클럭만 유지하면 된다.
   ③ 타임스템프 기법에서는 미래 시간이 메시지에 포함되면 나중에 해당 메시지를 다시 사용할 수 있는 문제가 있지만 카운터 기법은 현재보다 매우 큰 값을 사용하더라도 그 카운터 값으로 동기화할 수 있기 때문에 그와 같은 문제가 없다.
   ④ 타임스템프 기법에서는 시스템 클럭이 중요하게 보호되어야 하는 자산이지만 카운터 기법에서는 카운터가 중요한 보호 자산이 아니다.

4. 평문을 encrypt-then-mac 방법을 이용하여 인증 암호화하였을 때 특징을 설명한 다음 내용 중 틀린 것은?

   ① MAC 값이 확인되면 수신자는 암호문이 전송 과정에 조작되지 않았다는 것을 믿을 수 있다.
   ② MAC 값이 확인되고 상대방을 신뢰할 수 있다면 암호문 내에 포함된 여분 정보를 확인하지 않아도 암호문이 프로토콜에 정해진 암호키로 암호화되어 있다는 것을 확신할 수 있다.

③ MAC 값이 확인되면 수신자는 복호화한 평문이 조작되지 않았다는 것을 믿을 수 있다.

④ MAC 값이 확인되면 수신자는 평문의 내용을 확인하지 않아도 메시지의 최근성을 확신할 수 있다.

5. 평문이 두 개 블록 M_1, M_2로 구성되어 있다. 이 평문을 ECB 모드를 이용하여 대칭키 K를 이용하여 암호화한 결과를 C_1, C_2라 하자. 다음 내용 중 틀린 것은?

   ① M_1에는 여분 정보가 있고, M_2에는 여분 정보가 없을 때, 복호화한 후 M_1의 여분 정보가 확인되면 C_1과 C_2가 모두 K로 암호화된 암호문임을 확신할 수 있다.

   ② C_1을 복호화하여 여분 정보를 확인하였으면, C_1에 있는 나머지 정보의 무결성도 확신할 수 있다.

   ③ 공격자가 추측한 키로 암호문을 복호화하여 C_1에 있는 명백한 여분정보를 확인하였다. 이 경우 공격자는 자신의 추측이 맞았다는 것을 확신할 수 있다.

   ④ C_1과 C_2가 같은 키를 암호화된 암호문임을 확신할 수 있다는 가정하에 두 암호문 블록의 차이를 통해 대응되는 두 평문 블록의 차이를 예측할 수 없다.

연습문제

1. 이 문제는 1.1 절에서 설명된 내용에 대한 질문이다. 16byte(128bit) 블록 방식의 대칭 암호알고리즘을 이용하여 키 K로 $A||K_{AB}||padding$을 ECB 모드를 사용하여 암호화하였다고 하자. 여기서 A는 20byte, K_{AB}는 16byte이다. 수신자는 이 메시지의 내부 형태를 알고 있다고 가정하고, 채우기는 표준 채우기 이용하였다고 하자. 즉, 채워야 하는 12byte의 각 바이트를 정수 12(0x0C)로 채웠다고 하자. 실제 암호화된 평문을 C/C++의 구조체로 표현하면 다음과 같다.

   ```
   struct Msg{
       char id[20];
       char key[16];
   };
   ```

   수신자가 K로 복호화하였을 때 3개의 블록에 대해 무엇을 확신할 수 있는지 설명하시오. 여기서 id에는 널문자로 끝나는 C문자열이 유지된다. 이 문자열의 실제 크기가 답에 어떤 영향을 주는지 여부도 답에 포함해야 하며, 영향을 줄 경우에는 크기가 16보다 작은 경우와 16이상인 경우를 나누어 답하시오.

2. 1.2 절에서 둘 만이 공유한 비밀 정보 I_{AB}를 이용하여 함축적 여분 정보를 암호문에 포함하여 공유하는 방법을 소개하였다. A와 B가 소개된 방법을 이용하여 함축적 여분 정보를 포함하여 같은 키로 암호화된 여러 개의 암호문을 교환하였을 경우에는 어떤 문제점이 있는지 설명하시오. 해당 암호문에는 다른 여분 정보는 없다고 가정하고 답하시오.

3. 인증 암호화를 사용하면 명백한 여분 정보의 포함이 필요 없는 것인지 논하시오. 이를 위해 다음에서

   $$C = \{M\}.K_1, MAC.K_2(C)$$

   C가 K_1으로 암호화된 데이터인지 확인할 수 있는지 논하시오. 여기서 K_1과 K_2는 송신자와 수신자 간의 공유된 비밀키이다.

4. 메시지의 최근성을 보장하기 위해 타임스탬프 또는 난스 기법을 사용할 수 있다. 예를 들어 난스 기법을 사용하는 시스템에서 신뢰 서버가 A로부터 받은 난스 N_A를 이용하여 Alice에게 다음과 같은 메시지를 전달하였다고 가정하자.

   $$\{N_A||B||K_{AB}\}.K_{AS}$$

   이 메시지를 수신한 Alice는 이 암호문 전체를 최근에 생성한 것으로 믿을 수 있다. 그 근거는 무엇인지 설명하시오. 또 이때 K_{AB}의 최근성을 Alice가 믿을 수 있는지 설명하시오.

5. 메시지의 최근성을 보장하기 위해 타임스탬프 기반 기법을 사용할 수 있는데, 타임스탬프 기법은 사후 메시지 문제가 발생할 수 있다. 이 문제를 설명하고, 이 문제의 극복이 어려운 이유를 설명하시오.

6. 카운터는 타임스탬프 대신에 사용할 수 있는 기법으로 메시지에 포함된 카운터 값이 자신이 유지하는 카운터 값보다 클 경우에 최근 메시지로 인식하게 된다. 카운터를 이용한 기법의 가장 큰 문제는 각 사용자쌍마다 별도의 카운터가 필요하다는 것이다. 하지만 별도의 카운터를 유지해야 한다는 것이 항상 문제가 되지는 않는다. 그 이유를 설명하시오.

7. 본인이 사용하는 컴퓨팅 장치가 클럭 동기화를 어떻게 하고 있는지 간단히 조사하여 설명하시오.

8. 16비트 카운터(unsigned)를 사용할 때 1초에 1 증가한다고 가정하자. 그러면 이 카운터의 값이 순환되어 0이 될 때까지 소요되는 시간을 계산하시오. 32비트를 사용할 경우에도 동일한 시간을 계산하시오. 실제 컴퓨터 프로그래밍을 이용하여 소요되는 시간을 년, 일, 시간, 분, 초로 제시하시오.

9. 카운터를 이용한 메시지 최근성 보장 기법의 단점 중 하나는 각 사용자 쌍마다 다른 카운터를 유지해야 한다는 것이다. 만약 서버가 각 사용자마다 다른 카운터를 유지하지 않고 하나의 카운터만을 사용하는 것이 가능한지 논하시오. 사용 자체가 가능한 것인지, 사용하면 어떤 문제가 있는지 생각하고 본인의 생각을 서술하시오.

10. 타임스탬프 기반 기법에서는 보통 32비트 정수 값으로 시간(1970년 1월 1일 이후 경과된 초 값을 사용)을 나타낸다. 예를 들어 C언어에서는 `time_t` 타입을 사용한다. 이와 같은 타입은 2038 문제를 가지고 있다. 이것이 무엇인지 설명하시오.

11. 강한 개체 인증과 약한 개체 인증의 차이점을 설명하시오.

12. Bob이 Alice가 현재 온라인 상태인지를 확인하기 다음과 같은 프로토콜을 사용한다고 하자.

$$\begin{array}{ll} \text{Msg 1.} & B \rightarrow A: \quad N_B \\ \text{Msg 2.} & A \rightarrow B: \quad \text{Sig.}A(N_B) \end{array}$$

    이 프로토콜의 문제점을 제시하시오.

13. 다음과 같은 인증 프로토콜이 있다고 가정하자.

$$\begin{array}{ll} \text{Msg 1. } A \rightarrow B: & A, N_A \\ \text{Msg 2. } B \rightarrow A: & B, N_B, \text{Sig.}\text{–}K_B(N_B||N_A) \\ \text{Msg 3. } A \rightarrow B: & \text{Sig.}\text{–}K_A(N_B) \end{array}$$

    이 프로토콜은 강한 개체 인증을 제공하지 못한다. 강한 개체 인증을 제공하도록 수정하시오.

14. 문제 13에서 강한 개체 인증을 제공하도록 수정한 프로토콜에 대해 프로토콜의 시작자로써 A가 C와 하는 프로토콜을 작성하고, A 입장에서 두 프로토콜의 수행이 구분할 수 있는지 설명하시오. 또 프로토콜의 반응자로써 C와 B가 하는 프로토콜을 작성하고, B 입장에서 두 프로토콜의 수행이 구분할 수 있는지 설명하시오.