ⓒ2023, *sangjin@koreatech.ac.kr*

정보보호개론

제13장 고급 암호기술 1부: 비밀 공유 기법, 데이터 아웃소싱 보안

1. 개요

지금까지는 기본 암호알고리즘인 대칭과 비대칭 암호알고리즘, 해시함수, MAC, 전자서명을 살펴보았고, 키 확립 프토토콜을 중심으로 이들의 응용을 살펴보았다. 이 장에서는 전자서명과 암호화를 동시에 해주는 암호기법, 권한을 분산하고 싶을 때 사용할 수 있는 임계 기반 비밀 공유 기법, 클라우드 컴퓨팅 환경에서 사용할 수 있는 여러 암호기술을 살펴본다.

2. Signcryption

보통 함께 자주 사용하는 독립적인 연산들이 있으면 이를 효과적으로 결합하여 더 효율적으로 할 수 있는 방안을 찾아보게 된다. Signcryption은 이와 같은 시도의 대표적인 결과이다. Signcryption은 전자서명과 암호화를 동시에 해주는 알고리즘이다. 따라서 어떤 메시지에 대해 signcryption을 하게 되면 그 결과는 특정 수신자만 복호화할 수 있고, 이 수신자만 전자서명을 확인할 수 있다. 이 때문에 기존 전자서명과 달리 누구나 서명을 확인할 수 없다

A가 B에게 메시지 M을 signcrypt할 경우 다음 요구사항이 충족되어야 한다.

- B만 메시지를 복호화할 수 있어야 한다.
- B는 A의 서명을 확인할 수 있어야 한다.
- 전체 비용은 각각을 별도로 하였을 때보다 줄어야 한다.

세 번째 요구사항은 연산 비용과 결괏값의 길이 측면에서 별도로 하였을 때보다 효율적이어야 한다는 것을 의미한다.

전자서명과 암호화를 별도로 하는 방법은 보통 다음과 같이 진행된다.

- 단계 1. 메시지 M에 대해 전자서명을 수행하여 서명값 Sig.$A(M)$을 얻는다.
- 단계 2. 메시지 M과 Sig.$A(M)$을 랜덤 대칭키 K로 $\{M||\text{Sig}.A(M)\}.K$과 같이 암호화한다. 이때 서명값은 암호문에서 제외할 수 있다.
- 단계 3. K를 상대방 공개키로 암호화한 $\{K\}.+K_B$와 $\{M||\text{Sig}.A(M)\}.K$을 전달한다.

포함된 전자서명을 이용하여 메시지의 무결성을 확인할 수 있기 때문에 단계 2에서 인증 암호화를 할 필요는 없다.

ElGamal 전자서명의 변형 중 다음과 같은 서명 알고리즘이 있다. 이 알고리즘은 $\mathbb{Z}_p^*$의 부분군 $G_q = \langle g \rangle$를 이용한다. A의 서명키가 $x_A \in \mathbb{Z}_q^*$이고 공개키가 $y_A = g^{x_A} \mod p$일 때, 메시지 M에 대한 서명은 다음과 같이 계산된다.

- 단계 1. $w \in_R \mathbb{Z}_q^*$를 선택한 다음에 $r = H_q(g^w \mod p || M)$을 계산한다.
- 단계 2. $s = w(r + x_A)^{-1} \mod q$를 계산한다. 결과 서명은 M, r, s이다.

이 서명에 대한 확인은 다음과 같이 진행된다.

- 단계 1. $W = (y_A g^r)^s \mod p$를 계산한다.
- 단계 2. $H_q(W||M) \stackrel{?}{=} r$인지 확인한다.

이 전자서명을 이용한 signcryption은 다음과 같다[1].

- 단계 1. $w \in_R \mathbb{Z}_q^*$를 선택한 다음에 $(K_1, K_2) = H_q(y_B^w \mod p)$을 계산한다.
- 단계 2. $C = E.K_1(M)$과 $r = \text{MAC}.K_2(M)$을 계산한다.
- 단계 3. $s = w(r + x_A)^{-1} \mod q$를 계산한다. 결과 signcryption은 C, r, s이다.

Unsigncryption은 다음과 같이 진행된다.

- 단계 1. $(K_1, K_2) = (y_A g^r)^{s x_B} \mod p$을 계산한다.
- 단계 2. $M = D.K_1(C)$을 계산한 다음 $MAC.K_2(M) \stackrel{?}{=} r$인지 확인한다.

알고리즘 서술에서 알 수 있듯이 별도 세션키를 생성하지 않고 서명 과정에서 생성해야 하는 값으로부터 대칭키를 유도하고 있다. 이를 통해 연산 비용을 줄이고 있다.

서명과 암호화를 별도로 진행하는 것과 signcryption을 하는 것을 비교하여 보자. 서명과 암호화를 별도로 진행하면 ElGamal 전자서명에 제시된 것과 같이 서명은 r, s가 필요하고, 암호화는 메시지 M을 대칭키로 암호화한 암호문과 대칭키를 상대방 공개키로 암호화한 것이 필요하다. 이 경우, ElGamal hybrid 암호화 기법을 사용한다고 가정하자. 그러면 $g^\gamma, C = E.K(M)$가 필요하다. 여기서 $K = \text{KDF}(y_B^\gamma)$이다. 따라서 signcryption과 비교하면 별도 진행하는 것이 결괏값 측면에서 g^γ이 더 필요하다. 계산비용은 별도 진행하는 경우 ElGamal hybrid 암호화에서 필요한 g^γ, $K = \text{KDF}(y_B^\gamma)$가 추가로 더 필요하다.

3. 비밀 공유 기법

특정 권한을 특정 사용자가 홀로 가지고 있으면 해당 권한을 남용하기 쉽다. 이 문제는 보통 권한을 여러 사용자에게 나눔으로써 극복한다. 권한을 가지고 있는 여러 사용자가 동의 또는 협조할 경우에만 해당 권한을 행사할 수 있도록 장치를 만들게 된다. 그런데 n명에게 권한이 분산되었을 때 n명 모두의 협조가 필요하면 권한이 꼭 행사되어야 할 때도 행사를 못할 수 있다. 따라서 이와 같은 가용성 문제 때문에 보통 n보다 적은 수 t를 정해 그 이상의 사용자들이 협조하면 권한을 행사할 수 있도록 한다. 이를 (t, n) 임계(threshold) 방식이라 한다. 여기서 t는 보안변수로써 이 값이 클수록 안전성이 높아진다.

암호기술에서 권한은 그 권한을 행사하기 위한 암호키를 가졌는지 여부에 따라 결정된다. 예를 들어 인증기관의 인증서 발급 권한은 발급할 때 사용하는 서명키에 있다. 따라서 특정 암호키를 n명에게 나누고 이 중 t명 이상이 협조하면 암호키를 복원할 수 있도록 하여 권한 분산을 하고 있다. 이와 같은 기법을 암호기술에서는 비밀 공유 기법이라 한다. 이때 각 사용자가 유지하는 값을 그 사용자의 몫(share, shadow)이라 한다.

비밀 공유 기법은 공유할 값을 나누고 몫을 각 사용자에게 전달하는 신뢰 서버를 사용하는 경우와 서버를 전혀 사용하지 않고 참여하는 사용자 간의 값을 안전하게 나누는 경우로 분류할 수 있다. 전자의 경우 신뢰 서버는 해당 비밀(암호키)을 알고 있어 비밀 공유 기법의 원래 목적과 상충하는 측면이 있다. 또한 전자의 경우 각 사용자는 서버로부터 받은 몫이 유효한 몫인지 확인할 수 있어야 한다. 몫의 유효성을 확인할 수 있다면 이를 검증가능 비밀 공유(verifiable secret sharing) 기법이라 한다. 후자는 분산 비밀 공유 기법이라 한다.

3.1 간단한 비밀 공유 기법

대칭키 K를 n명에게 공유하기 위해 신뢰 서버가 대칭키와 동일한 길이의 랜덤한 S_i를 $n-1$개 생성한 후 $S_n = K \oplus S_1 \oplus S_2 \oplus \cdots \oplus S_{n-1}$을 계산하여 각 사용자 i에게 해당 사용자의 몫으로 S_i를 나누어 주었다고 하자. 이 경우 n명의 사용자가 K를 복원하고 싶으면 사용자의 모든 몫을 XOR하면 된다. 따라서 이 방식은 (n, n) 방식이 된다. 이 방식의 문제점은 다음과 같다.

- 문제점 1. 사용자의 몫을 생성해 줄 중앙 서버가 필요하다.
- 문제점 2. 모두 협력해야 복원이 가능하다.
- 문제점 3. 하나의 조각이 분실되면 복원할 수 없다.
- 문제점 4. 한 번 복원하면 비밀키가 노출된다.

문제점 2와 3은 (n, n) 방식이기 때문에 발생하는 문제점이며, 대칭키를 비밀 공유할 경우 문제점 4는 근본적으로 해결할 수 없다. 실제 대칭키를 비밀 공유하면 이것을 사용하는 것 자체가 어렵다.

3.2 Shamir의 비밀 공유 기법

Shamir는 $t-1$차 다항식을 이용하여 (t, n) 임계 기반 기법을 만들 수 있음을 보였다[2]. 예를 들어 $t = 2$일 때 대칭키 K를 공유하고 싶으면 K가 상수항이 되는 임의의 1차 다항식 $f(x) = ax + K$를 만든 후 각 사용자 i에게 $f(i)$ 값을 몫으로 나누어 주면 된다. k차 다항식의 기본 특성 때문에 다항식의 $k+1$개의 해를 알면 이 다항식의 계수들을 결정할 수 있으며, 상수항 값도 계산할 수 있다. 따라서 만들고 싶은 (t, n) 임계 기반 기법에 맞게 $t-1$차 다항식을 선택하면 원하는 수의 n명과 비밀을 공유할 수 있다.

이 기법은 다음과 같이 중앙 서버를 사용하지 않고 분산 방식으로 비밀 공유도 가능하다. 각 사용자가 동일한 차수의 다항식 $f_i(x) = a_i x + K_i$를 만든 후 각 사용자 i는 다른 사용자 j에게 자신의 $f_i(j)$ 값을 전달한다. 각 사용자는 받은 모든 값을 더하여 자신의 최종 몫으로 사용한다. 실제 이들이 최종적으로 사용하는 다항식은 각 사용자가 선택한 다항식의 합인 $f(x) = \sum a_i x + \sum K_i$가 되며, 비밀은 $K = \sum K_i$가 된다. 수학의 라그랑주 보간법(Lagrange interpolation)을 이용하면 t개의 점이 주어졌을 때 이를 지나는 독특한 $t-1$ 다항식을 쉽게 구할 수 있으며, 해당 다항식의 상수항을 구할 수 있다.

Shamir의 비밀 공유 기법을 공개키로 확장하여 보자. 중앙 집중 서버를 통해 개인키를 공유하는 것부터 살펴보자. $\mathbb{Z}_p^*$의 부분군 $G_q = \langle g \rangle$에서 (t, n) 임계기반 ElGamal 암호화가 가능하도록 하기 위해 서버는 다음과 같은 $t-1$차 다항식을 생성한다.

$$f(x) = a_{t-1}x^{t-1} + \cdots + a_1 x + s, \forall a_i \in_R \mathbb{Z}_q^*$$

여기서 시스템의 공개키는 $y = g^s$가 된다.

서버는 각 참여자에게 $s_i = f(i)$를 몫으로 전달하고, $y_i = g^{s_i}$와 $c_{t-1} = g^{a_{t-1}}$, ..., $c_1 = g^{a_1}$을 공개한다. 이산대수 문제 때문에 이들이 공개되어도 다항식의 계수와 상수항은 노출되지 않는다. 각 사용자는 $g^{s_i} \stackrel{?}{=} y \prod_{j=1}^{t-1} c_j^{i^j}$를

이용하여 자신의 몫을 확인할 수 있다. 또한 라그랑주 보간법에 따라 다음이 성립한다.

$$b_j = \prod_{1 \le k \le t, k \ne j} \frac{k}{k-j}, s = \sum_{j=1}^{t} s_j b_j$$

이를 나중에 복호화할 때 활용하게 된다.

시스템의 공개키 y를 이용하면 누구나 $(g^r, C = E.K_1(M), \text{MAC}.K_2(C))$을 계산하여 ElGamal 방식으로 암호화할 수 있다. 여기서 $K_1||K_2 = \text{KDF}(y^r)$이다. t명이 모이면 이 암호문을 다음과 같이 복호화할 수 있다.

- 단계 1. 각 참여자는 $w_i = (g^r)^{s_i}$를 계산하여 공개한다.
- 단계 2. 각 참여자는 $\prod w_i^{b_i} = \prod g^{rs_ib_i} = g^{r\sum s_ib_i} = g^{rs} = y^r$를 계산할 수 있으며, 이를 이용하여 K_1과 K_2를 계산하여 메시지를 복호화할 수 있다.

여기서 알 수 있듯이 특정 암호문을 복호화하더라도 s가 노출되지 않는다. 따라서 대칭키를 공유하였을 때와 달리 개인키를 이처럼 공유할 경우 공개키를 계속 사용할 수 있다.

4. 데이터 아웃소싱

데이터 아웃소싱(data outsourcing)이란 데이터를 내부 서버에 유지하는 것이 아니라 외부 서버에 위탁하여 관리하는 것을 말한다. 데이터 관리 비용을 줄이기 위해 데이터를 아웃소싱하는 기업이 증가하고 있으며, 클라우드 컴퓨팅의 확산으로 이와 같은 현상이 더욱 가속화되고 있다.

하지만 이와 같은 아웃소싱은 중요한 데이터 또는 사적 데이터를 자신이 통제할 수 없는 외부에 유지해야 하므로 정보의 노출, 정보의 조작, 정보의 손실 등의 문제가 발생할 수 있다. 이와 같은 문제점 중 정보의 노출 문제를 극복하기 위해 데이터를 암호화하여 유지하는 것을 고려할 수 있다. 하지만 데이터를 암호화하면 검색과 같은 서비스를 이용하기 힘들 수 있다. 정보의 조작이나 손실 문제는 무결성 검증 서비스를 통해 해결할 수 있으나, 아웃소싱하는 데이터의 규모와 외부 서버의 신뢰 문제를 고려하였을 때 효과적인 검증 방법을 만드는 것이 쉽지 않다.

데이터 아웃소싱의 요구사항들은 다음과 같다.

- 비밀성, 프라이버시 보장
- 접근제어 보장
- 효율적인 검색

비밀성과 프라이버시 보장은 제3자에게만 해당하는 것은 아니며, 아웃소싱 서버나 업체에 데이터를 노출하지 않고 서비스를 받을 수 있기를 원한다. 이를 위해 앞서 언급한 바와 같이 데이터를 암호화하여 유지하는 것이 필요할 수 있다. 여기서 접근제어는 데이터의 공유와 관련되어 있으며, 공유할 때 읽기, 쓰기 권한을 제어할 수 있어야 한다.

4.1 아웃소싱된 데이터의 보호

아웃소싱된 데이터는 소유자가 직접 관리할 수 있는 공간에 데이터를 유지하는 것이 아니기 때문에 데이터의 비밀성을 위해 데이터를 암호화하여 유지하는 것이 필요하다. 우리가 궁극에 원하는 것은 제3자는 물론 데이터를 유지하는 서비스 업체도 저장된 데이터의 내용을 볼 수 없는 것이다. 이것을 제공하기 위한 몇 가지 간단한 해결책을 살펴보자.

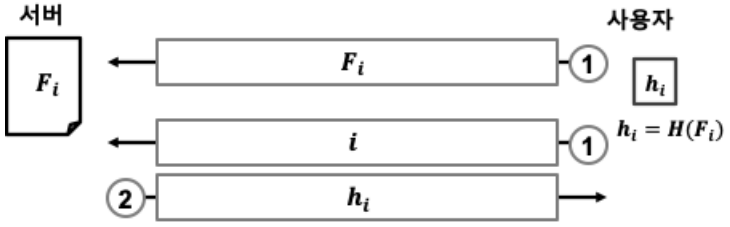
첫째, 저장 서버가 모든 권한을 가지고 있도록 하는 방법을 생각해볼 수 있다. 사용자가 데이터를 업로드하면 서버가 이를 암호화하여 저장하고, 암호키를 서버가 관리한다. 해당 파일을 사용자가 요청하면 암호화된 데이터를 복호화하여 안전한 채널로 전달해주거나 암호화된 데이터와 이를 복호화하기 위한 키를 전달할 수 있다. 이 방법은 서버는 데이터의 내용을 볼 수 있으므로 궁극에 원하는 형태의 서비스가 아니다.

둘째, 사용자가 데이터를 업로드할 때 암호화하여 업로드하고, 이와 관련된 키 관리를 사용자가 직접 하는 방법을 생각해볼 수 있다. 이때 가장 효과적인 방법은 데이터마다 랜덤 대칭키를 생성하여 암호화하고 그것을 사용자의 공개키로 암호화하여 파일과 함께 저장하는 것이다. 이 경우 사용자는 자신의 공개키 쌍만 유지하면 되며, 어느 장치에서나 해당 공개키를 사용할 수 있도록 하면 여러 장치에서 서버에 접근하여 사용할 수 있다. 이 방법은 서버에 대해 전혀 신뢰할 필요가 없지만, 다중 사용자 환경에서 공유와 같은 기능을 서버에 위탁하기 어렵다는 문제점이 있다.

두 번째 방법의 경우 데이터마다 공개키로 암호화된 대칭키가 함께 저장되어야 한다. 보통 사용하는 공간에 비례하여 비용을 지불하기 때문에 이것이 부담될 수 있다. 이 경우 모든 데이터를 동일 대칭키로 암호화하고, 해당 대칭키를 패스워드 기반이나 공개키로 암호화하여 사용자가 유지하는 방법도 생각해 볼 수 있다. 이렇게 하면 공간 사용을 최소화할 수 있지만, 안전성이나 공유와 같은 기타 기능을 제공하는 것이 더 어렵게 된다. 그 이유는 어떤 키가 노출되면 그 키로 암호화된 모든 데이터도 노출되기 때문이다.

4.2 아웃소싱된 데이터의 무결성

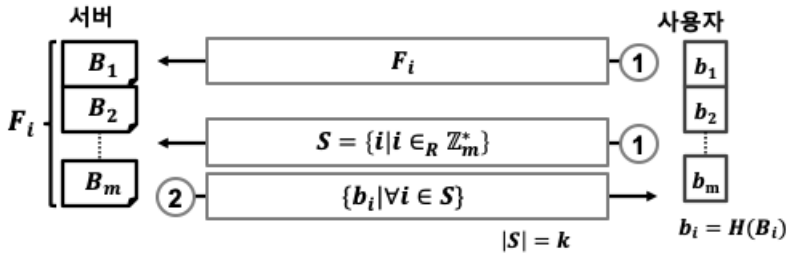
사용자는 아웃소싱된 데이터를 자신이 필요할 때 조작되지 않은 상태로 접근할 수 있기를 원한다. 이를 위해 해당 데이터의 무결성을 검증할 수 있어야 한다[3, 5]. 이와 같은 서비스를 회수 증명(POR, Proof-Of-Retrievability)이라 한다. 하지만 이 서비스는 해당 데이터를 다운받지 않은 상태에서 확인할 수 있어야 하며, 사용자는 해당 데이터를 로컬에 가지고 있지 않다고 가정해야 한다. 더욱이 이 서비스가 어려운 점은 무결성 서비스에 대해서는 저장 서버를 신뢰할 수 없다는 것이다. 데이터가 변조되었다 하더라도 이에 대한 책임을 지지 않기 위해 저장 서버는 그것을 인정하지 않을 수 있다.



<그림 13.1> 해시함수 기반 무결성 검증 기법

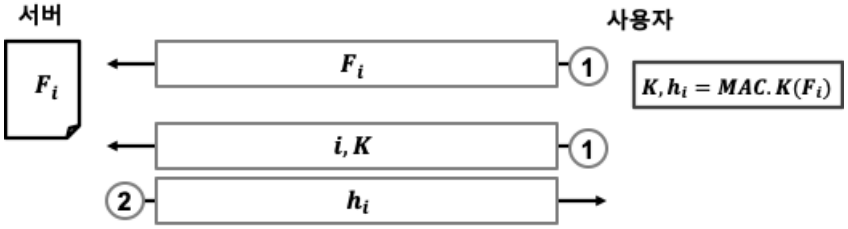
무결성을 검증하기 위해 가장 많이 사용하는 것이 해시함수이다. 해시함수를 이용한 그림 13.1과 같은 간단한 해결책을 생각하여 보자. 사용자는 데이터를 업로드하기 전에 그것의 해시값을 계산하여 로컬에 유지한다. 서버는 사용자가 특정 데이터에 대한 무결성 검증 서비스를 요청하면 해당 데이터의 해시값을 계산하여 사용자에게 전달하여 준다. 이 방식은 크게 두 가지 문제점이 있다. 첫째, 서버는 요청마다 데이터에 대한 해시값을 계산하여 전달하여야 하는데, 데이터의 크기가 크면 이 비용이 부담될 수 있다. 둘째, 서버는 항상 해시값을 보관한 다음에 매번 새롭게 계산하지 않고 보관된 해시값을 전달하여 데이터가 조작되었지만 속일 수 있다. 물론 사용자도 자신이 업로드한 데이터마다 해시값을 유지해야 하는 번거로움이 있지만, 무결성을 확인하기 위한 어떤 값도 유지하지 않고 무결성 서비스를 받을 수는 없다.

데이터의 크기에 따른 해시함수 계산비용 문제는 데이터가 보통 특정 블록 크기 단위로 나누어 관리되기 때문에 블록 단위의 해시값을 사용하여 개선할 수 있다. 예를 들어 그림 13.2와 같이 사용자가 특정 데이터에 대한 무결성을 검증하기 위해 임의의 블록들에 대한 해시값을 요구하고 이를 받아 확인하는 방법을 생각해 볼 수 있다. 하지만 이



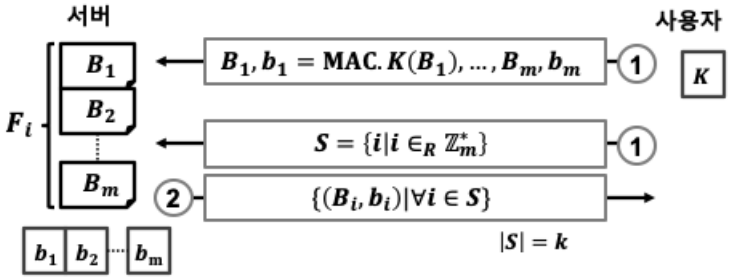
<그림 13.2> 해시함수 기반 무결성 검증 기법: 블럭 단위 기법

방법은 사용자가 특정 데이터마다 블록 개수만큼의 해시값을 유지해야 하므로 사용자 측면에서 부담될 수 있다. 서버는 모든 블록의 해시값을 유지하여 속일 수 있지만, 저장 공간이 낭비되기 때문에 이를 통해 얻을 수 있는 이득이 없다.



<그림 13.3> MAC 기반 무결성 검증 기법

제시된 해시함수를 이용한 해결책을 분석해 보면 효과적인 안전한 해결책이 되기 위해서는 사용자에게 전달하는 값을 서버가 매번 새롭게 계산해야 하며, 사용자나 서버나 데이터를 검증하기 위해 유지해야 하는 것이 많지 않아야 한다. 서버가 매번 새롭게 계산하도록 하기 위해 해시함수 대신에 MAC을 이용하는 것을 생각해 볼 수 있다. MAC은 해시함수와 달리 키가 필요하다. 이 특징을 활용하기 위해 MAC값을 계산할 때 그림 13.3과 같이 사용할 키를 매번 사용자가 전달하는 방법을 생각해 볼 수 있다. 하지만 이 방법을 사용하기 위해서는 사용자가 데이터를 서버에 저장하기 전에 여러 개의 대칭키를 생성하고 각 대칭키를 이용하여 MAC값을 계산하여 보관해야 한다. 또 보관한 값을 다 사용하면 원 데이터를 다운받아 다시 MAC값을 계산해야 한다.



<그림 13.4> MAC 기반 무결성 검증 기법: 블럭 단위 기법

또 다른 방법으로 해시함수를 사용할 때 제시한 방법처럼 전체 파일 대신에 특정 블록 크기 단위로 파일을 나누어 각 블록에 대한 MAC 값을 계산하는 방법을 생각해 볼 수 있다. 이때 키를 주어 서버가 MAC을 계산하도록 하면 사용자가 각 블록마다 여러 개의 MAC 값을 유지해야 하기 때문에 유지해야 하는 값이 너무 많아 사용할 수 있는 해결책이 될 수 없다. 따라서 확인을 요청할 때 키를 전달하는 것이 아니라 그림 13.4와 같이 모든 블록의 MAC 값을 계산하여 서버에 주어 유지하도록 하고, 검증할 때 유지된 MAC 값과 블록 자체를 서버가 전달하도록 하는 방법을 생각해 볼 수 있다. 이 방법의 핵심 생각은 서버는 MAC 키가 없어 블록이 조작되었을 경우 현재 상태에 맞는 MAC 값을 계산할 수 없다는 것이다. 이 방법은 서버가 속일 수 있는 방법은 없지만 무결성 검증을 위해 교환해야 하는 값의 크기가 너무 크다는 문제점이 있다. 또 파일마다 여러 개의 MAC 값을 유지해야 하기 때문에 사용자가

사용하는 저장공간이 늘어나는 문제점도 있다.

이처럼 해시함수나 MAC을 단순하게 사용하여 효과적인 해결책을 만들 수 없다는 것을 알 수 있다. Filho와 Barreto는 RSA를 활용하는 기법을 제안하였다[4]. 처음 소개한 MAC 기반 해결책은 사용자가 n개의 키와 MAC 값을 유지하고, 무결성 검증을 요청할 때 서버에 키를 전달해야 하는 문제점이 있었는데, 이 기법은 이 문제를 해결해 주는 기법이다. 서버가 매번 새롭게 MAC 값을 계산하여 전달하지만 사용자는 여러 개의 MAC 값을 유지할 필요가 없다. 사용자는 $n = pq$를 계산한 후, F가 저장할 데이터이면 $k = F \mod \phi(n)$을 계산하여 유지한다. 사용자는 무결성 검증을 확인하고 싶으면 임의의 $g \in \mathbb{Z}_n^*$을 선택하여 g와 n을 전달한다. 이때 서버에 n을 등록해둔 상태이면 g만 전달할 수 있다. 서버는 $s = g^F \mod n$을 계산하여 사용자에게 전달한다. 사용자는 s와 g^k를 비교한다. $s = g^F = g^{a\phi(n)+k} \equiv g^k \pmod n$이므로 두 값이 일치하면 데이터가 조작되지 않았다는 것을 확신할 수 있다. 서버가 사용자가 확인할 때 사용하는 k를 계산할 수 있으면 이를 보관한 다음 매번 새롭게 계산하지 않고 사용자를 속일 수 있지만 서버는 n을 인수분해 할 수 없어 $\phi(n)$을 알 수 없으므로 이것이 가능하지 않다. 따라서 서버는 사용자가 매번 새로운 g를 전달하기 때문에 요청마다 새롭게 계산할 수밖에 없다. 사용자가 유지해야 하는 값도 효과적이고 서버도 매번 새롭게 계산해야 하기 때문에 원하는 모든 조건이 충족되는 기법이다. 참고로 F가 F_1부터 F_m까지 정해진 크기의 블록으로 나누어진다고 하면 실제 k를 계산할 때 사용하는 $F = \sum_{i=1}^{m} F_i$이고, $s_i = g^{F_i} \mod n$일 때 $s = \prod_{i=1}^{m} s_i$가 된다.

RSA는 부분 동형 암호(partially homomorphic encryption)이다. 동형 암호란 암호화된 데이터에 대해 복호화하지 않고 연산을 적용할 수 있는 암호이다. RSA는 $E(M_1)E(M_2) = E(M_1 + M_2)$가 성립한다. 이 때문에 동일 키로 암호화된 두 개의 암호문을 곱하여 복호화하면 두 평문의 합을 얻을 수 있다. RSA는 이 연산만 복호화하지 않고 할 수 있으므로 부분 동형 암호이다. Ateniese 등은 이 특성을 이용하는 기법을 제시하였다[5]. 이 기법은 블록 단위 MAC 값을 서버에 유지하고, 서버가 사용자가 요청한 랜덤 c개 블록에 대해 블록 데이터와 MAC 값을 보내주는 방법을 동형 암호를 이용하여 매번 하나의 값만 보내도록 개선한 기법이다. 이 방식에서 서버는 사용자로부터 n개 블록에 대한 MAC 값을 받아 유지해야 한다. 여기서 사용하는 MAC 값은 RSA 개인키를 이용하여 메시지 블록을 암호화한 값이다. 사용자는 c개의 랜덤 블록에 대한 무결성 검증을 요청한다. 이때 랜덤한 값 $g_s = g^s$를 함께 전달한다. 서버는 동형 암호를 이용하여 저장되어 있는 해당 블록의 MAC값을 결합하고, 도전값 g_s와 해당 블록 데이터를 이용하여 또 다른 값을 계산하여 사용자에게 전달한다. 사용자는 자신의 공개키와 s를 이용하여 전달받은 값을 검증한다. 서버는 사용자가 매번 새로운 g_s 값을 전달하며, 선택하는 블록 위치가 매번 달라지기 때문에 매번 새롭게 계산할 수밖에 없다. 서버는 각 블록마다 MAC값을 유지해야 하지만 증명 과정에서 교환되는 값의 크기는 동형암호로 인하여 사용하는 블록 개수와 무관하다. Filho와 Barreto와 마찬가지로 사용자가 유지해야 하는 공간복잡도는 파일과 무관하게 $O(1)$이다.

4.3 프록시 재암호화

프록시 재암호화(proxy re-encryption)은 한 사용자의 개인키로 복호화할 수 있는 암호문을 다른 사용자의 개인키로 복호화할 수 있도록 변환하여 주는 것을 말한다[6]. 이 변환을 수행하는 프록시는 암호화된 데이터의 내용을 볼 수 없어야 한다. 각 사용자와 대칭키를 공유하고 있는 서버가 한 사용자와 공유한 키로 암호화된 암호문을 복호화한 다음 다른 사용자와 공유한 키로 암호화하는 것은 서버가 데이터의 내용을 볼 수 있기 때문에 프록시 재암호화가 아니다. 더욱이 프록시 재암호화의 경우 사용자가 권한을 줄 경우에만 재암호화를 할 수 있는데, 이 시나리오에서는 특정 사용자가 권한을 주지 않아도 할 수 있다. 즉, B의 공개키로 암호화된 데이터가 있을 때, B가 재암호화 키 RK_{BA}를 프록시에게 주면, 프록시는 이를 이용하여 이 암호문을 A의 공개키로 암호화된 암호문으로 바꾸어 줄 수 있다.

프록시 재암호화는 동일한 데이터를 n명에게 비밀로 전달하고 싶을 때 사용할 수 있다. 동일한 데이터를 다수에게 비밀로 전달하고자 할 때 가장 먼저 떠오르는 방법은 12장에서 살펴본 그룹키 프로토콜이다. 보통 그룹키 프로토콜은 초기 설정 비용이 비싸지만 한번 확립된 이후에는 확장성 있게 사용자들의 가입과 탈퇴를 처리할 수 있다. 따라서 그룹 멤버가 빈번하게 바뀔 수 있는 환경에서 전방향, 후방향 안전성을 만족시키고 싶으면 이 방법이

가장 효과적인 방법일 수 있다.

프록시 재암호화 기법을 이 용도로 사용한다면 전송자는 우선 자신의 공개키로 메시지를 암호화하여 프록시에게 전달한다. 프록시는 이 암호문을 각 n번 재암호화하여 각 사용자에게 전달할 수 있다. 따라서 전송자의 부담은 거의 없는 방식이 된다. 물론 전송자는 n개의 재암호화키를 프록시에게 주어야 하는 문제는 있지만 매번 재암호화키를 주어야 하는 것은 아니기 때문에 지속해서 사용할 경우 이 비용은 초기 설정 비용으로 생각할 수 있다. 반면에 프록시 측면에서는 n에 비례하는 계산비용과 통신비용이 소요되는 문제가 있다.

프록시 재암호화에서 사용자들은 프록시를 무조건 신뢰하지 않는다. 프록시는 정직하게 행동하지만, 항상 궁금해하는 존재로 암호화된 데이터의 내용을 보고 싶어한다고 가정한다. 이와 같은 모델을 "curious-but-honest model" 이라 한다. 프록시를 사용하는 환경에서 프록시가 협조하지 않으면 이 기법은 정상적으로 동작할 수 없기 때문에 프로토콜 수행에 대해서는 신뢰를 할 수 있어야 한다. 따라서 프록시 재암호화의 기본적인 요구사항은 다음과 같다.

- R1. 프록시는 재암호화 과정에서 평문을 얻을 수 없어야 한다.
- R2. 프록시 권한제한: 프록시는 권한이 부여된 재암호화만 할 수 있어야 한다.
- R3. 공모 안전성(collusion-safety): 사용자 간 공모 또는 사용자와 프록시 간의 공모를 통해 불법적인 복호화가 가능하지 않아야 한다.

프록시 재암호화에서 특정 공개키로 암호화되어 있는 암호문을 다른 사용자가 복호화할 수 있도록 재암호화를 하기 위해서는 재암호화키가 필요하다. 이 키는 대응되는 개인키를 가지고 있는 사용자만이 만들 수 있다. 따라서 프록시는 사용자로부터 필요한 재암호화키를 받지 못하면 재암호화를 할 수 없다.

프록시 재암호화는 위 3개의 요구사항 외에 다음과 같은 추가 요구사항을 가지고 있다[7].

- R4. 단방향(unidirectional): A에서 B로 재암호화 권한이 있어도 자동으로 B에서 A로의 재암호화 권한이 위임되지 않는다.
- R5. 비추이성(non-transitive): 프록시는 스스로 재암호화 권한을 위임받을 수 없어야 한다.
- R6. 비전이성(non-transferable): 프록시는 사용자들과 공모하여도 위임받지 않은 권한을 위임받을 수 없어야 한다.
- R7. 비상호작용(non-interactive): A가 재암호화키를 생성하고자 할 때 제3의 신뢰서버를 포함하여 다른 참여자와 상호작용할 필요가 없이 홀로 생성할 수 있어야 한다.

R5의 경우 프록시가 A에서 B로의 재암호화키와 B에서 C로의 재암호화키를 가지고 있더라도 이를 이용하여 A에서 C로의 재암호화키를 만들거나 A의 공개키로 암호화된 암호문을 C가 복호화할 수 있도록 재암호화를 할 수 없어야 한다. R6는 R3 요구사항을 구체화한 것으로 프록시가 A에서 B로의 재암호화키를 가지고 있고 B와 공모하여 B의 개인키를 확보할 수 있다고 하더라도 A에서 C로의 재암호화키를 만들거나 A의 공개키로 암호화된 암호문을 C가 복호화할 수 있도록 재암호화를 할 수 없어야 한다.

R3, R5, R6를 충족하기 위해 보통 두 단계 암호화 수준을 제공한다. 재암호화가 가능하지 않은 암호알고리즘과 재암호화가 가능한 암호알고리즘을 각각 L1과 L2라 할 때, L2 알고리즘을 이용하여 암호화된 암호문을 재암호화하면 L1 수준의 암호문이 되도록 설계한다. 따라서 이와 같은 방식에서 한번 재암호화된 암호문은 다시 재암호화할 수 없게 된다. 어떤 프록시 재암호화는 일정 기간이 지나면 재암호화가 가능하지 않도록 제한할 수 있는 것도 있다.

프록시 재암호화를 데이터 아웃소싱에서 어떻게 활용할 수 있을까? 앞서 데이터를 업로드할 때 데이터를 랜덤 대칭키로 암호화하고, 대칭키를 공개키로 암호화하여 함께 보관할 수 있다고 하였다. 이 상태에서 어떤 데이터를

다른 사용자와 공유하고 싶으면 재암호화키를 발급하면 된다. 저장 서버는 발급된 재암호화키로 암호화된 랜덤 대칭키를 재암호화하여 해당 사용자에게 주면 해당 사용자는 암호화된 데이터를 복호화할 수 있게 된다. 하지만 프록시 재암호화는 세밀한 수준의 접근 제어를 제공하지 못하기 때문에 특정 사용자에게 공유하고 싶지 않은 데이터도 저장 서버가 재암호화하여 줄 수 있는 문제점이 있다.

프록시가 RK_{AB}를 가지고 있으면 A의 공개키로 암호화된 모든 암호문을 재암호화할 수 있다. 이 문제는 두 단계 암호화를 통해 어느 정도 극복할 수는 있다. 공유할 목적이 아닌 데이터들은 L1 알고리즘으로 암호화하고, 공유 목적이 있는 데이터들은 L2 알고리즘으로 암호화하는 것이다. 하지만 이 경우에도 L2로 암호화된 데이터가 여러 개 있으면 이 모든 데이터는 재암호화가 가능하다.

4.4 속성기반 암호화

프록시 재암호화의 단점은 세밀한 접근제어를 제공할 수 없다는 것이다. 이 단점은 속성기반 암호화(attribute-based encryption)를 통해 해결할 수 있다. 속성기반 암호화는 복호화키 또는 암호화된 데이터에 할당된 속성 또는 접근 정책을 바탕으로 암호화된 데이터에 대한 접근을 제어하여 주는 기술이다. 이 기술은 신원기반 암호시스템을 기반하고 있다. 신원기반에서는 사용자의 신원을 이용하여 공개키를 생성하는데, 속성기반에서는 신원정보 대신에 속성을 이용하여 공개키를 생성하는 방식이다.

신원기반 암호시스템에서 확장된 개념이므로 사용자에게 키를 발급하여 주는 서버를 사용한다. 따라서 이 서버는 신원기반 암호시스템과 마찬가지로 막강한 권한을 가지게 된다. 물론 신원기반 암호시스템과 마찬가지로 임계기반 비밀공유 기법을 사용하여 이 문제는 어느 정도 효과적으로 해결할 수 있다. 속성기반은 추가로 공모 문제도 해결해야 한다. 두 사용자가 각각은 접근할 수 없는 것이면 공모를 하더라도 접근을 할 수 없어야 한다. 이 기법의 또 다른 문제는 철회 문제이다. 사용자로부터 또는 암호화된 문서로부터 속성을 철회할 수 있어야 하는데, 이것이 쉽지 않다는 것이다. 이것은 신원기반 암호시스템에서 키 갱신을 효과적으로 할 수 없는 것과 연관되어 있다.

속성기반 암호기법이 기존 기법들과 가장 차별화되는 점은 개념적으로는 하나의 공개키에 여러 개의 개인키를 연결하여 사용할 수 있다는 것이다. 당연한 것이지만 어떤 데이터가 속성기반을 이용하여 암호화되어 있으면 그 속성이나 접근 정책을 만족하는 다수가 복호화할 수 있다. 따라서 속성기반 암호화는 그룹키 대신에 다자간 통신에 활용할 수 있다. 컴퓨터공학부 학생만 볼 수 있도록 암호화한 하나의 암호문을 멀티캐스트하면 그 조건을 충족하는 모든 사용자는 그 문서를 수신하여 복호화할 수 있다. 더욱이 속성기반 암호시스템은 복호화하는 조건을 기존 프록시 기법과 달리 매우 세밀하고 풍부하게 표현할 수 있는 장점이 있다.

클라우드 환경에서는 클라우드 서비스 제공자와 독립적으로 데이터를 암호하하는 주제가 접근제어 정책을 스스로 암호화할 때 결정할 수 있다. 공개키 개념이므로 누구나 필요한 접근 정책에 따라 문서를 암호화할 수 있지만 그것을 복호화할 수 있는 사용자는 제한이 된다. 따라서 속성 기반으로 암호화된 데이터는 그 조건을 충족하는 사용자만 복호화할 수 있기 때문에 암호화된 데이터의 공유 문제를 효과적으로 해결할 수 있다.

속성기반 암호화의 요구사항은 다음과 같다.

- R1. 데이터 기밀성(data confidentiality): 권한이 없는 사용자는 데이터를 얻을 수 없어야 한다.
- R2. 공모에 대한 저항성(collusion resistance): 여러 사용자가 공모하더라도 각 개인이 얻을 수 있는 데이터만 얻을 수 있어야 한다.
- R3. 전방향 안전성(forward secrecy): 특정 속성이 철회된 사용자는 속성이 철회된 이후에는 해당 속성이 필요로 하는 데이터를 얻을 수 없어야 한다.

R3와 관련하여 후방향 안전성도 고려되는 경우가 있다. 즉, 새 속성을 얻을 사용자가 현재 권한으로 접근할 수

있었던 이전 데이터를 얻을 수 없어야 한다는 것을 말한다. 하지만 이 요구사항은 그룹키 프로토콜과 달리 속성 기반에서는 중요하게 고려되는 요구사항은 아니다.

초기 속성기반 암호화 기법은 임계기반 접근 정책만 제공하였다. 즉, 가지고 있는 속성 수를 이용하여 접근 가능 여부를 판단하였다. 하지만 후에 트리 기반 접근 구조 및 다양한 논리연산을 사용할 수 있도록 개선되었다. 특히, 일부 연구에서는 논리부정까지 사용할 수 있도록 개선됨에 따라 접근 정책을 매우 풍부하게 표현할 수 있게 되었다. 실제 특정 속성 A가 있으면 NOT A라는 속성을 만들어 논리부정을 제공할 수 있다. 하지만 이와 같은 방식으로 논리부정을 사용하기에는 속성의 개수가 너무 많아지는 문제점이 있다. 따라서 이와 같은 직관적인 방법을 사용하지 않고 논리부정을 제공할 수 있어야 효과적인 방식이 된다.

(1) 암호문 정책 속성기반 암호화 (2) 키 정책 속성기반 암호화

<그림 13.5> 속성기반 암호화 종류

속성기반 암호화는 크게 그림 13.5와 같이 **암호문 정책 속성기반 암호화**(CP-ABE, Ciphertext-Policy ABE)[8]와 **키 정책 속성기반 암호화**(KP-ABE, Key-Policy ABE)[9]로 분류할 수 있다. 두 방식은 속성과 접근 정책을 서로 상대반 된 것에 연관한다. CP-ABE는 암호문과 접근구조(정책)를 연관하며, 이를 복호화할 수 있는 사용자 개인키에 속성을 연관한다. 따라서 CP-ABE는 접근 정책을 문서를 암호화할 때 결정할 수 있다. 예를 들어 홍길동은 “컴퓨터공학부”, “교수”라는 속성과 연관된 키를 발급받게 되며, 문서는 컴퓨터공학부 교수 또는 컴퓨터공학부 조교만 볼 수 있도록 암호화할 수 있다. 논리적으로 문서에 대한 접근 정책을 표현하면 {(“컴퓨터공학부” and “교수”) or (“컴퓨터공학부” and “조교”)}가 된다.

KP-ABE는 CP-ABE와 정반대로 접근 정책은 사용자 개인키에 연관하고, 암호문에 속성을 연관한다. 예를 들어 성춘향 사용자는 {“성춘향” or (“컴퓨터공학부” and “입학”)}이라는 접근 정책에 해당하는 개인키를 발급받는다. 이때 문서가 “컴퓨터공학부”, “입학” 속성들로 암호화되어 있다면 성춘향은 이 문서를 복호화할 수 있다. KP-ABE는 문서를 암호화할 때 할당된 속성 간의 관계를 나타내지 못하는 단점이 있다. 위 예에서 홍길동 사용자가 {“홍길동” or “컴퓨터공학부”}라는 접근 정책에 해당하는 키를 발급 받은 사용자이면 “컴퓨터공학부” 속성이 할당되어 모든 문서를 복호화할 수 있다. 해당 문서에 할당된 다른 속성은 복호화하는데 아무런 영향을 주지 못한다.

당연한 것이지만 접근 정책 구조가 복잡할수록 암호문의 크기는 그것에 비례하여 커진다. 이 때문에 CP-ABE에서 암호문은 접근 정책을 나타내는 트리가 복잡할수록 그 암호문의 크기가 커진다. 반면에 KP-ABE에서 암호문의 크기는 할당된 속성에 비례한다.

속성기반 암호화에는 두 종류 철회가 존재한다. 하나는 속성 철회이고, 다른 하나는 사용자 철회이다. 속성 철회는 암호화된 문서에 연관된 특정 속성을 철회하는 것을 말하며, 사용자 철회는 다시 사용자에게 연관된 특정 속성을 철회하는 것과 사용자가 시스템을 사용할 수 없도록 하는 것으로 나누어질 수 있다. 암호문에 연관된 정책이나 속성을 바꿀 필요가 있으면 기존 암호문에 어떤 연산을 적용하여 바꾸는 것보다 새롭게 암호화하는 것이 더 효과적이다. 따라서 이 부분을 어떤 연산을 통해 제공할 필요는 없다. 반면에 사용자의 경우에는 그 사용자의 권한을 변경할 수 있어야 한다. KP-ABE는 사용자에게 속성을 할당하기 때문에 할당된 속성 철회하거나 추가할 수 있어야 한다. 속성의 추가는 관련 개인키를 추가로 발급하면 되지만 기존에 발급된 키를 사용할 수 없도록 만드는 것은 간단한 문제가 아니다. 실제 그 키로 복호화할 수 있는 모든 문서의 변경이 필요하기 때문이다. CP-ABE는 사용자에게 접근 정책이 할당되기 때문에 이 정책을 바꾸는 것은 새로 발급하면 쉽게 해결될 수 있다고 생각할 수 있지만 기존에 가지고 있는 개인키들을 사용할 수 없도록 하여야 하기 때문에 이 또한 간단한 문제가 아니다.

4.5 검색가능 암호화

외부 서버에 기밀성 또는 프라이버시 때문에 데이터를 암호화하여 유지할 수 있다. 이때 접근 제어를 세밀하고 풍부하게 제공하는 문제의 해결도 필요하지만 데이터의 양이 많으면 필요한 데이터를 효과적으로 찾는 문제도 해결되어야 한다.

필요한 데이터를 찾는 방법은 크게 브라우징과 검색으로 구분할 수 있다. 브라우징은 데이터를 폴더 기반으로 정리하여 유지하고 사용자는 폴더를 변경하면서 필요한 폴더로 이동하여 데이터를 찾는 방법을 말한다. 따라서 브라우징은 사용자가 대략 데이터가 어느 폴더에 위치하고 있는지 알고 있어야 한다. 반면에 검색은 데이터가 어디에 있는지 모르는 상태에서 찾는 것을 말한다. 따라서 다중 사용자 환경에서 브라우징을 이용하여 필요한 데이터를 찾는 것은 한계가 있다. 또 데이터의 내용을 숨기고 싶으면 파일 이름도 원래 이름을 사용할 수 없다.

데이터가 암호화되어 있으면 다양한 검색 기술을 직접 적용할 수 없다. 물론 색인 작업을 하는 검색 서버가 복호화키를 알고 있으면 기존 검색 기술을 그대로 적용할 수 있지만, 이 경우 검색 서버를 신뢰하여야 한다. **검색가능 암호화**(searchable encryption)[10]는 암호화된 데이터의 비밀성과 프라이버시를 보호한 상태(검색 서버를 포함하여)로 효과적으로 검색할 수 있도록 해주는 기술이다. 이와 같은 환경에서 검색 서버는 프록시 재암호화와 마찬가지로 데이터 내용에 대해서는 궁금해하지만 다른 모든 서비스에 대해서는 신뢰 있게 해동한다고 가정한다.

<그림 13.6> 키워드 기반 검색가능 암호화

지금까지 제안된 대부분의 검색가능 암호시스템은 그림 13.6과 같이 동작하는 키워드 기반 검색만 지원하였다. 보통 데이터 생성자가 데이터를 검색할 때 사용할 키워드를 암호화하여 암호화된 데이터와 함께 저장한다. 질의자는 질의 키워드를 암호화하여 검색서버에게 전달하면 검색서버는 질의 키워드의 내용을 모르는 상태에서 암호화된 키워드와 비교하여 검색하게 된다. 질의자가 검색 서버에 전달하는 암호화된 질의 키워드를 질의 트랩도어라 하기도 한다.

트랩도어와 키워드 암호문과의 비교는 보통 순차 검색을 통해 이루어진다. 키워드 암호문을 색인하는 것을 생각할 수 있지만 트랩도어를 이용하여 이 색인을 검색할 수 없기 때문에 가능한 방법이 아니다. 대신에 문서와 키워드 암호문을 연결하여 유지하는 것이 아니라 키워드 암호문마다 그것과 연관된 문서를 유지하면 검색이 문서 수에 비례하는 것이 아니라 키워드 수에 비례하게 된다. 또 검색할 키워드를 암호화하여 트랩도어를 만들기 때문에 질의에 사용된 키워드와 저장된 키워드가 조금만 달라도 검색이 안 된다. 여러 개의 트랩도어를 전달하여 다중 키워드 검색을 할 수 있지만 각 트랩도어를 이용하여 검색한 후에 각 결과를 종합하는 수준밖에는 제공할 수 없다. 이와 같은 키워드 기반 검색의 한계를 극복하기 위한 여러 기법[10]이 제안되고 있지만 여기서는 초기 키워드 기반 검색만 살펴본다,

초기 키워드 기반 검색 기법 중 대칭키 기반도 있고, 공개키 기반도 있다. 대칭키 기반의 경우에는 키워드 암호문과 트랩도어 암호문을 비교해서 키워드 일치 여부를 판단해야 하기 때문에 키워드 암호문을 생성할 때 사용하는 키와 트랩도어를 생성할 때 사용하는 키가 같은 대칭키이다. 공개키 기반에 경우에는 공개키로 키워드 암호문을 생성하고, 개인키를 이용하여 트랩도어를 생성한다. 보통 이들 기법에서 키워드 암호문을 확률적 암호화를 하지만 트랩도어는 결정적 암호화를 한다. 이 경우 동일 키워드에 대한 트랩도어가 변하지 않는다. 검색서버는 전달받은 트랩도어를 별도 검증하지 않는다. 검색서버는 받은 트랩도어의 유효 여부와 상관없이 검색을 진행한다. 따라서 유효하지 않은 트랩도어는 문서와 연관되어 있지 않은 키워드를 이용하여 검색하는 것과 차이가 없다. 검색서버는

보통 트랩도어가 아니라 다른 방법으로 사용자를 인증한다. 또 문서를 암호화하는 키와 검색가능 암호시스템에 사용하는 키는 연관될 필요는 없다.

검색가능 암호시스템은 전자우편과 데이터 아웃소싱에 활용할 수 있다. 전자우편의 경우 전자우편을 암호화하고 이를 검색하기 위한 키워드를 암호화하여 서버에 전달할 수 있으며, 사용자는 서버에 질의 트랩도어를 전달하여 원하는 메일을 검색할 수 있다. 데이터 아웃소싱에서는 저장 서버에 대한 신뢰를 줄이기 위해 데이터를 암호화여 유지할 수 있으며, 이 데이터에 대한 검색을 위해 사용할 수 있다.

검색가능 암호시스템의 요구사항은 다음과 같다.

- R1. 검색 서버는 암호화된 문서의 내용을 알 수 없어야 한다.
- R2. 검색 서버는 암호화된 키워드의 내용을 알 수 없어야 한다.
- R3. 검색 서버는 사용자가 전달한 질의 트랩도어의 내용을 알 수 없어야 한다,
- R4. 유효한 질의 트랩도어는 오직 권한을 가지고 있는 사용자만 생성할 수 있어야 한다.

R4는 트랩도어를 생성할 때 사용하는 키를 통해 제공한다.

검색가능 암호시스템은 저장하는 주체와 검색하는 주체에 따라 다음과 같이 분류할 수 있다.

- SWSR(Single-Writer-Single-Reader) 모델. 특정 사용자만 홀로 데이터를 저장할 수 있고, 검색할 수 있는 모델
- SWMR(Single-Writer-Multi-Reader) 모델. 특정 사용자만 홀로 데이터를 저장할 수 있지만 검색은 다수가 할 수 있는 모델
- MWSR(Multi-Writer-Single-Reader) 모델. 여러 사용자가 데이터를 저장할 수 있지만, 오직 한 사용자만 검색할 수 있는 모델
- MWMR(Multi-Writer-Multi-Reader) 모델. 여러 사용자가 데이터를 저장할 수 있고, 여러 사용자가 검색할 수 있는 모델

기본적으로 대칭키 방식은 SWSR 모델만 제공할 수 있다. 이 대칭키를 그룹키로 공유하면 다중 사용자 모델로 확장할 수 있지만 SWMR이나 MWSR을 제공하기는 어렵다. 공개키 방식은 기본적으로 MWSR 모델을 제공한다. 당연한 것이지만 MWMR이 가능하도록 시스템을 설계하는 것이 가장 어렵다. 특히, 다중 사용자가 참여하는 경우 사용자의 가입과 탈퇴에 대한 처리까지 필요하다. 하지만 그룹관리는 저장 서버와 분리하여 운영할 수 있으며, 이렇게 운영하는 것이 보안 측면에서 바람직하다. 보통 기업의 경우에는 데이터를 아웃소싱하더라도 외부 서버에 대한 의존도를 줄이기 위해 그룹 관리는 자체적으로 하는 것이 필요하다.

사용하는 암호기술이 안전하다고 가정할 경우 검색가능 암호시스템을 통해 정보를 어느 정도까지 숨길 수 있는지 생각하여 보자. 첫째, 검색 서버는 암호화된 문서의 내용을 알 수 없다. 둘째, 저장된 키워드 암호문에 대한 키워드 추측 공격이 가능하지 않을 경우, 특정 문서와 연관된 키워드를 알 수 없다. 셋째, 키워드 암호문은 보통 확률적 암호화 기법을 사용하기 때문에 키워드 암호문의 비교를 통해서는 문서 간의 연관관계를 알 수 없지만, 이 부분은 질의를 통해 어차피 노출된다. 예를 들어 사용자의 특정 질의에 대해 특정 문서들이 해당 질의에 충족되었다면 이 문서들은 특정 키워드를 공유한 문서라는 것은 서버에게 바로 노출된다.

넷째, 트랩도어에 대한 키워드 추측 공격이 가능하지 않으면 트랩도어에 포함된 키워드를 알 수 없다. 다섯째, 특정 트랩도어와 연관된 문서의 수는 노출될 수밖에 없다. 여섯째, 특정 키워드에 대한 트랩도어가 변하지 않으면 사용자의 질의 패턴이 노출된다.

실제 지금까지 제안된 여러 기법에서 동일 키워드에 대한 트랩도어 값은 변하지 않는다. 따라서 질의의 불연결성을 제공하지 못하는 문제점이 있다. 트랩도어를 서버에 전달할 때 안전한 채널로 전달할 수 있지만, 검색 서버에게는 여전히 노출될 수밖에 없다. 또한 암호화된 키워드와 암호화된 문서를 서로 바인딩할 수 없다. 질의자가 트랩도어를 구성할 때 문서를 사용할 수 없으며, 특정 키워드가 여러 문서와 연관될 수 있기 때문에 바인딩하는 것 자체가 가능하지 않다. 이 때문에 색인 데이터베이스가 보호되지 않으면 여러 가지 방해 공격이 가능하다.

5 동형 암호

동형 암호(homomorphic encryption)는 암호화된 데이터를 복호화하지 않고 데이터에 필요한 계산을 수행할 수 있는 암호 방식을 말한다. 따라서 동형 암호화된 데이터에 어떤 연산을 적용한 후에 복호화하면 평문에 연산이 적용된 상태로 복호화된다. 동형 암호는 이 장에서 살펴본 데이터 아웃소싱에서 중요한 역할을 할 수 있다. 기밀 또는 프라이버시를 위해 외부에 암호화하여 유지한 데이터를 복호화하지 않고 필요한 연산을 수행할 수 있기 때문에 데이터가 불필요하게 노출되는 것을 방지할 수 있다.

동형 암호는 크게 다음과 같이 분류한다.

- 부분(partially) 동형 암호: 한 가지 연산만 제공하는 경우
- 완전(full) 동형 암호: 적용할 수 있는 연산에 제한이 없는 경우

부분과 완전 사이에 위치하는 동형 암호도 존재한다. 두 가지 연산만 제공하는 동형 암호도 있고, 임의 연산을 적용할 수 있지만 적용할 수 있는 횟수가 사전에 제한되어 있는 동형 암호도 존재한다. 동형 암호는 암호문을 조작하면 평문에 영향을 주는 형태이므로 본질적으로 NM 특성을 만족하지 못한다.

완전 동형 암호의 가능성은 Gentry가 양자내성암호 기술로 고려되고 있는 격자 기반 암호기술을 사용하여 보였다[11]. 지금은 2017년에 천정희 등[12]이 제안한 방식이 가장 효과적인 완전 동형 암호로 인정되고 있으며, 라이브러리로 개발되어 프라이버시 보장 기계 학습에 활용되고 있다. 동형 암호는 이처럼 공개키 기술과 관련되어 있다. 하지만 평문에 직접 연산을 적용할 때 소요되는 비용과 비교하면 아직 성능이 우수하다고 말하기 어렵다.

채우기를 사용하지 않는 RSA, ElGamal 암호는 부분 동형 암호를 제공한다. ElGamal 암호를 통해 동형 암호를 간단히 살펴보자. 다음과 같이 m_1, m_2를 암호화한 두 ElGamal 암호문이 있을 때,

$$E_1 = (g_1^r, y_1^r m_1), E_2 = (g_2^r, y_2^r m_2)$$

이 두 암호문을 곱하면 다음을 얻게 된다.

$$E_1 E_2 = (g^{r_1+r_2}, y^{r_1+r_2} m_1 m_2)$$

따라서 기본 ElGamal에서 암호문 두 개를 곱한 후 복호화하면 두 평문을 곱한 결과를 얻게 된다. 평문이 g^a, g^b일 때 두 암호문을 곱하면 다음을 얻게 된다.

$$E_1 E_2 = (g^{r_1+r_2}, y^{r_1+r_2} g^{a+b})$$

따라서 암호문 두개를 곱하면 평문의 지수가 덧셈이 되는 형태를 얻을 수 있다.

참고문헌

[1] Y. Zheng, "Digital Signcryption or How to Achieve Cost (Signature & Encryption) << Cost (Signature) + Cost (Encryption)," Advances in Cryptology, Crypto 97, LNCS 1294, pp. 165–179, Springer, 1997.

[2] A. Shamir, "How to Share a Secret," Communications of ACM, Vol. 22, No. 11, pp. 612–613, 1979.

[3] A. Juels, B. Kaliski, "PORs: Proofs of Retrievability for Large Files," ACM Conf. on Computer and Communications Security, pp. 584–597, 2007.

[4] D.L.G. Filho, P.S.L.M. Barreto, "Demonstrating Data Possession and Uncheatable Data Transfer," IACR eArchive, 150, 2006.

[5] G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, D. Song. "Provable Data Possession at Untrusted Stores," ACM Conf. on Computer and Communications Security, pp. 598–609, 2007.

[6] Matt Blaze, G. Bleumer, M. Strauss, "Divertible protocols and atomic proxy cryptography," Advances in Cryptology, Eurocrypt 98, LNCS 1403, pp. 127–144, Springer, 1998.

[7] G. Ateniese, K. Fu, M. Green, S. Hohenberger, "Improved Proxy Re-encryption Schemes with Applications to Secure Distributed Storage," ACM Trans. on Information and System Security, Vol. 9, No. 1, pp. 1–30, Feb. 2006.

[8] J. Bethencourt, A. Sahai, B. Waters, "Ciphertext-Policy Attribute-Based Encryption," IEEE Symp. on Security and Privacy, pp. 321–334, May 2007.

[9] Vipul Goyal, Amit Sahai, Omkant Pandey, Brent Waters, "Attribute-Based Encryption for Fine-Grained Access Control of Encrypted Data," ACM Conf. on Computer and Communications Security, pp. 89–98, Nov. 2006.

[10] Christoph Bosch, Pieter Hartel, Willem Jonker, Andreas Peter, "A Survey of Provably Secure Searchable Encryption," ACM Computing Surveys, Vol. 47, No. 2, Aug. 2014.

[11] Craig Gentry. "Fully Homomorphic Encryption Using Ideal Lattices," Proc. of the 41st ACM Symp. on Theory of Computing, pp. 169–178, May 2009.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, Yongsoo Song, "Homomorphic Encryption for Arithmetic of Approximate Numbers," Advances in Cryptology, Asiacrypt 2017, LNCS 10624, pp. 409–437, Springer, 2017.

퀴즈

1. 그룹 통신을 위해 여러 가지 암호기술을 사용할 수 있다. 그룹 크기가 n일 때 한 사용자가 동일 메시지를 그룹 멤버들에게 보내고 싶다고 가정하였을 때, 각 방법의 사용과 관련된 다음 설명 중 틀린 것은?

   ① 프록시 재암호화 기법을 사용할 경우 전송자는 자신의 공개키로 1번만 암호화하면 된다. 대신에 프록시는 해당 사용자로부터 $n-1$개의 재암호화키를 받아 재암호화하여 각 재암호화된 암호문을 각 사용자에게 유니캐스트로 전달해야 한다.

   ② LKH, OFT와 같은 기법을 이용하여 그룹 멤버 간의 그룹키를 확립한 다음, 그룹키로 메시지를 암호화한 암호문을 멀티캐스트하여 그룹 통신을 할 수 있다.

   ③ 모든 그룹 멤버들이 같은 속성을 갖도록 속성 기반 키를 발급받아 가지고 있다면 사용자는 이 속성을 가진 사용자들만 복호화할 수 있도록 속성기반 암호화한 암호문을 멀티캐스트하여 그룹 통신을 할 수 있다.

   ④ $(2, n)$ 비밀 공유 기법을 사용하여 개인키를 그룹 멤버들과 공유한 다음, 대응되는 공개키로 암호화한 암호문을 멀티캐스트하여 그룹 통신을 할 수 있다.

2. 3명이 중앙서버를 사용하지 않고 (2,3) 임계기반 비밀 공유를 하기 위해 각 사용자는 다음과 같은 다항식을 만들었다.

$$\begin{aligned} f_1(x) &= 3x + 2 \\ f_2(x) &= 2x + 4 \\ f_3(x) &= 4x + 1 \end{aligned}$$

사용자 2가 사용자 1과, 사용자 3으로부터 받는 값은 무엇이며, 사용자 2의 최종 몫은 다음 중 어느 것인가?

① 8, 9, 25

② 5, 5, 16

③ 11, 13, 34

④ 8, 8, 25

3. 키워드 기반 검색 가능 암호시스템에 대한 다음 설명 중 틀린 것은?

① 각 문서와 연관되어 있는 키워드 암호문과 검색 키워드 암호문(트랩도어)을 비교하여 검색 키워드와 연관된 문서를 찾는다.

② 키워드 암호문을 확률적 암호화하는 이유는 두 문서와 연관된 키워드 암호문 목록을 비교하더라도 서로 연관된 문서인지 알 수 없도록 하는 것인데, 이 정보는 어차피 검색 결과를 통해 노출된다. 이 때문에 확률적 암호화 대신에 결정적 암호화를 하고 해당 키워드와 연관된 문서 목록을 유지하는 것이 더 효과적이다.

③ SWSR, SWMR, MWSR, MWMR 모델과 무관하게 검색 가능 암호화에 사용되는 키는 검색 서버와 별도로 관리되어야 한다.

④ 질의 트랩도어를 확률적 암호알고리즘을 이용하여 생성하면 질의 패턴을 검색 서버로부터 숨길 수 있다.

4. 아웃소싱된 데이터에 대한 무결성 검증 서비스와 관련된 다음 설명 중 틀린 것은?

① 사용자는 각 데이터마다 무결성 검증을 위한 무언가의 데이터는 유지하고 있어야 서비스가 가능하며, 데이터를 로컬에 유지하고 있지 않다고 가정한다.

② 이 서비스에서 서버를 신뢰할 수 없기 때문에 서버는 매번 사용자가 확인할 검증 코드를 새롭게 계산하여 전달하도록 해야 한다.

③ 서버가 검증 코드를 효과적으로 계산할 수 있도록 전체 데이터가 아니라 데이터의 일부만 이용하여 계산하도록 하는 것이 필요하다.

④ 데이터가 암호화되어 있으면 무결성 검증 서비스를 제공하기 어렵다.

5. 임계기반 비밀공유 기법과 관련된 다음 설명 중 틀린 것은?

① 대칭키는 한번 복원하면 키가 노출되기 때문에 여러 번 사용할 수 없다.

② (t, n) 임계기반 비밀 공유를 하고 싶으면 t차 다항식이 필요하다.

③ 중앙서버가 몫을 계산하여 나누어 주지 않고 분산 방식으로 비밀 공유를 하고 싶으면 각 참여자는 스스로 정해진 차수의 다항식을 랜덤하게 만들어 다른 참여자에게 해당 다항식의 값을 하나 주면 된다. 참여자는 이들 값들을 모아 최종 공유할 비밀값의 몫을 계산할 수 있다.

④ (n, n)은 가용성 때문에 $(1, n)$은 권한 분산이 아니므로 보통 사용하지 않는다.

연습문제

1. 전자서명 측면에서 일반 서명과 signcryption의 차이를 설명하시오.

2. 데이터를 직접 통제할 수 있는 서버에 유지하지 않고 비용을 절감하기 위해 외부 서버에 유지하는 경우가 많아지고 있다. 이를 데이터 아웃소싱이라 한다. 데이터를 아웃소싱할 경우 외부 서버에 대한 신뢰가 매우 중요하다. 하지만 외부 서버를 완전히 신뢰하기 힘들며, 외부업체가 충분히 신뢰할 수 있는 업체라 하여도 그 서버에 접근 권한을 가지고 있는 모든 직원을 신뢰하기는 어렵다. 따라서 데이터를 아웃소싱하였을 때 데이터를 암호화하여 유지하고 싶은 욕구가 많다. 이를 위해 3.1에서 제시한 다음 두 가지 방법을 사용한다고 가정하자.

   - 방법 1: 데이터마다 새로운 대칭키를 생성하고 이 키를 공개키로 암호화하여 암호화된 데이터와 함께 유지함
   - 방법 2: 하나의 대칭키로 모든 데이터를 암호화하고 해당 대칭키를 로컬에 사용자가 안전하게 유지함

   이와 관련하여 다음 각각에 대해 답변하시오.

① 방법1과 방법 2 둘 다 AES-CTR 모드로 데이터를 암호화한다고 가정하자. 그러면 암호화하지 않았을 때와 암호화하였을 때 공간 사용의 차이가 있는지 설명하시오. 단, 키 유지를 위한 비용은 여기서 고려하지 않는다.

② 안전성 측면에서 방법 2는 어떤 문제가 있는지 설명하시오.

③ 방법 1을 사용하였을 때 프록시 암호기법을 이용하여 공유 문제를 해결하고 싶다. 데이터를 저장한 주체가 A이고, 특정 데이터를 B와 공유하고 싶으면 어떻게 해야 하는지 간단히 설명하시오. 또한 이 경우 문제점은 무엇인지 설명하시오.

3. 데이터 아웃소싱과 관련하여 원격에 있는 데이터가 변경되지 않았다는 것을 확인하고 싶다. 하지만 이때 파일 자체를 다운받아 확인하는 것이 번거로울 수 있다. 이 경우에도 서버는 변경된 사실을 숨기고 싶을 수 있다. 이를 위해 사용자가 해당 파일의 해시값을 유지하는 방법을 생각해 볼 수 있다. 다음 각각에 대해 간단히 답하시오.

① 서버는 원래 이 기법에서 매번 요청이 있을 때마다 새롭게 해시값을 계산하여 주어야 한다. 하지만 변경된 사실을 숨기고 싶으면 서버는 어떻게 하면 숨길 수 있는지 설명하시오.

② 위 문제를 극복하기 위해 해시값 대신에 MAC값을 사용하고자 한다. 즉, 요청할 때마다 사용자는 새 대칭키를 주어 계산하도록 한다. 이것의 문제점을 설명하시오.

4. 프록시 재암호화 기법에서 사용하는 두 단계 암호화 수준을 설명하시오. 특히, 비추이성을 위해 왜 두 단계 암호화가 필요한지 설명하시오.

5. 비밀공유기법에서 (t, n) 임계기반 기법을 만들 때 다항식의 역할을 설명하시오.