KISA 안내 · 해설 제2011-9호

암호기술 구현 안내서

Ver 1.0

2011. 11

방송통신위원회
KOREA COMMUNICATIONS COMMISSION

KISA 한국인터넷진흥원
Korea Internet & Security Agency

CONTENTS

Korea Internet & Security Agency

1. 개 요

1 개요

1.1 배경 및 목적

최근 인터넷을 통한 개인정보 유출 사고가 빈번해지면서, 개인정보 암호화를 위해 웹사이트 등에 구현된 암호기술의 안전성에 대한 논란이 일고 있다. 이는 취약성이 있는 암호기술을 구현 · 적용 했거나 또는 안전한 암호기술을 선정했음에도 불구하고 구현상의 오류로 인해 개인정보가 노출될 가능성이 있기 때문이다.

본 안내서는 이러한 문제점을 해소하기 위해 필수적으로 구현해야 할 암호기술을 다양한 기준과 적용 상황에 따라 분류하고 이를 안전하게 구현하는 방법을 소개하고자 한다. 본 안내서에서 소개하는 암호기술 구현 방법은 업체의 시스템 및 서비스 현황에 따라 달라질 수 있다.

1.2 적용대상 및 구성

본 안내서는 보호의무가 있는 개인정보 등을 관리하는 기업 및 공공기관, 단체 등의 개발자에게 암호기술을 구현하기 위한 기술적 활용 방안을 제시하고 있다.

안내서 구성은 다음과 같다. 먼저 2장에서는 암호기술 적용이 필요한 정보의 대상과 세부 기술을 소개하고 있으며, 3장에서는 비밀번호 및 암호키 관리 방안을 제시한다. 또한, 4장에서는 암호기술 구현 방법을 예를 들어 소개하고 있으며, 예시로 소개되고 있는 소스코드의 라이브러리는 안내서와 함께 제공된다. 마지막으로 암호화 관련 국내 법 · 규정 현황을 부록에 포함했다.

1.3. 용어정의

다음은 본 안내서에서 사용되는 용어들에 대한 정의이다.

- **개인정보처리자** : 업무를 목적으로 개인정보 파일을 운용하기 위하여 자체적으로 또는 위탁을 통하여 개인정보를 처리하는 공공기관, 법인, 단체 및 개인 등을 말한다.

- **정보주체** : 처리되는 정보에 의하여 인지 및 식별이 가능한 객체로 대상 정보의 주체가 되는 사람을 말한다.

- **암호화** : 의미를 알 수 없는 형식(암호문)으로 정보를 변환하는 것을 말한다. 이는 개인정보취급자의 실수 또는 해커의 공격 등으로 인해 개인정보가 비인가자에게 유 · 노출되더라도 그 주요 내용을 확인할 수 없게 하는 보안기술이다.

- **암호키** : 메시지를 암호화 또는 복호화 하는데 사용되는 키로, 암호키를 소유한 자만이 암호문을 생성하거나 복호할 수 있다.

- **해쉬함수** : 임의 길이의 메시지를 항상 고정된 길이의 해쉬 값으로 변환하는 일방향 함수를 말한다.

- **일방향 함수** : 결과값을 가지고 입력값을 구하는 것이 어려운 함수로, 해쉬함수는 일방향 함수에 해당한다.

- **블록암호** : 평문을 일정한 블록 크기로 나누어, 각 블록을 송 · 수신자 간에 공유한 비밀키를 사용하여 암호화하는 방식

- **비밀번호** : 정보주체 또는 개인정보취급자 등이 개인정보처리시스템, 업무용 컴퓨터 또는 정보통신망에 접속할 때 식별자와 함께 입력하여 정당한 접속 권한을 가진 자라는 것을 식

별할 수 있도록 시스템에 전달해야 하는 고유의 문자열로서 타인에게 공개되지 않는 정보를 말한다.

- **바이오정보** : 지문, 얼굴, 홍채, 정맥, 음성, 필적 등 개인을 식별할 수 있는 신체적 또는 행동적 특징에 관한 정보로서 그로부터 가공되거나 생성된 정보를 포함한다.

- **SSL(Secure Sockets Layer)** : 데이터를 송 · 수신하는 두 컴퓨터 사이, 종단 간 즉 TCP/IP 계층과 어플리케이션 계층(HTTP, TELNET, FTP 등) 사이에 위치하여 인증, 암호화, 무결성을 보장하는 사실 표준 프로토콜을 말한다.

- **TLS(Transport Layer Security)** : SSL을 대신하는 차세대 안전 통신 규약으로, SSL을 대신하는 차세대 안전 통신 규약으로, SSL보다 적용 범위가 넓고 안전성과 활용성이 강화되었다.

- **보안서버** : 인터넷상에서 전송되는 자료를 암호화하여 송 · 수신하는 기능을 제공하는 웹 서버를 지칭한다.

1.4 용어의 효력

본 안내서에서 사용된 용어들은 암호기능의 구현 정보를 의미하는 것으로 다음과 같은 의미를 지닌다.

- **해야한다, 필수이다, 강제한다** : 반드시 준수해야 한다.
- **권고한다** : 보안성 등을 고려하여 준수할 것을 권장한다.
- **할 수 있다, 쓸 수 있다** : 주어진 상황을 고려하여 필요한 경우에 한해 선택적으로 사용할 수 있다.
- **권고하지 않는다** : 보안성 등을 고려하여 사용하지 말 것을 권장한다.
- **금지한다, 허용하지 않는다** : 반드시 사용하지 않아야 한다.
- **언급하지 않는다, 정의하지 않는다** : 준수 여부에 대해 기술하지 않는다.

Korea Internet & Security Agency

2. 암호 적용 대상 및 기술

2 암호 적용 대상 및 기술

정보통신망 이용촉진 및 정보보호 등에 관한 법률(이하 '정보통신망법'), 개인정보보호법 등에서는 인터넷을 통해 유통되는 정보의 보호를 위해 암호기술을 구현하도록 규정하고 있다. 이에, 본 장에서는 관련법을 기반으로 암호기술 적용이 필요한 정보의 종류를 식별하고, 정보의 종류에 따라 적용 가능한 암호기술에 대해 설명한다. 암호기술 적용을 규정하고 있는 관련법에 대한 상세 내용은 「부록. 암호관련 법 · 규정 현황」을 참조하기 바란다.

2.1 암호화가 필요한 정보의 종류

개인정보 등이 분실 · 도난 · 유출 또는 훼손되지 아니하도록 암호화가 필요한 정보는 다음과 같이 크게 두 가지로 분류할 수 있다.

① 암호화된 정보를 다시 복호화 할 수 없어야 하는 정보

- ▣ 정보주체를 제외하고 정보를 다루는 관리자조차 암호화된 정보의 원래 정보가 무엇인지 알 수 없어야 하는 정보들
- ▣ 예 : 비밀번호

※ 정보통신망법 시행령 제15조 및 개인정보의 기술적·관리적 보호조치 기준(방통위 고시 제2011-1호) 제6조에서는 바이오정보에 대해 복호화할 수 없도록 규정하고 있으나, 향후 시행령 및 고시 개정을 통해 복호화할 수 있어야 하는 정보로 재분류 예정

② 암호화된 정보를 다시 복호화 할 수 있어야 하는 정보

- ▣ 정보 보관 시에는 암호화되어 있어야 하나, 사용 시에는 복호화가 가능하여 원래 정보를 알 수 있어야 하는 정보들
- ▣ 예 : 바이오정보, 주민등록번호, 신용카드번호, 계좌번호, 여권번호, 운전면허번호, 외국인등록번호

※ 정보통신망법 시행령 및 개인정보의 기술적 · 관리적 보호조치 기준(방통위 고시 제2011-1호)에서는 주민등록번호, 신용카드번호 및 계좌번호만 규정

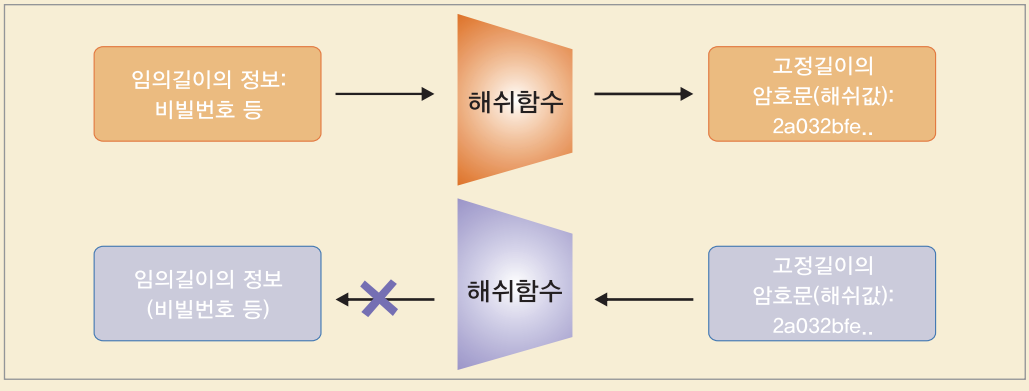
이러한 정보들은 데이터베이스에 저장하는 경우는 물론, 정보통신망을 통해 송 · 수신되는 경우에도 반드시 암호화되어야 한다.

2.2 정보 저장 시 적용 가능한 암호기술

2.1에서 도출한 정보들의 종류에 따라, 암호화된 정보를 다시 복호화 할 수 없어야 하는 정보들은 해쉬함수, 복호화 할 수 있어야 하는 정보들은 블록암호 알고리즘을 사용하여 암호화해야 한다.

2.2.1 해쉬함수

해쉬함수는 임의길이의 정보를 입력으로 받아, 고정된 길이의 암호문(해쉬값)을 출력하는 암호기술로 암호화된 정보는 복호화가 불가능한 특징을 가지고 있다. 따라서 해쉬함수를 이용하면 (그림 2-1)과 같이 비밀번호를 입력하여 암호문(해쉬값)을 생성해 낼 수는 있지만, 암호문(해쉬값)을 가지고 원래의 비밀번호를 알아낼 수는 없다. 즉, 개인정보처리자도 시스템에 저장된 암호문(해쉬값)을 가지고 원래의 사용자 비밀번호를 알 수 없기 때문에 안전한 비밀번호 관리가 가능해 진다.



(그림 2-1) 해쉬함수의 개념

대표적인 해쉬함수로는 〈표 2-1〉과 같이 국내 표준 HAS-160, 국제 표준 SHA-1, SHA-2등이 있다.

〈표 2-1〉 대표적인 해쉬함수

	HAS-160	SHA-1	SHA-2			
			SHA-224	SHA-256	SHA-384	SHA-512
암호문(해쉬값) 길이(비트)	160	160	224	256	384	512
참조규격	TTA[1] TAS.KO.12.0011/R2	NIST[2] FIPS 180-3				

위에서 나열된 해쉬함수들은 각기 다른 보안강도를 제공한다. 보안강도란 암호 알고리즘이나 시스템의 비밀키 또는 해쉬함수의 취약성을 찾아내는데 소요되는 작업 양을 수치화한 것이다. 예를 들어, 80비트의 보안강도란 2^{80}번의 계산을 해야 비밀키 또는 암호 알고리즘의 취약성을 알아낼 수 있다는 것을 의미한다.

국내 · 외 암호 관련 전문기관에서 분석한 결과를 참조하여 도출한 해쉬함수의 보안강도는 〈표 2-2〉와 같으며, 이는 IT 환경의 기술수준(컴퓨팅 파워, 해킹 능력 등)이 변화되면 달라질 수 있다. 2011년 현재 권고하는 해쉬함수는 SHA-224, SHA-256, SHA-384, SHA-512이다. 다만, 현재 많은 서비스에서 사용하고 있는 SHA-1 해쉬함수는 알고리즘 변경에 따른 비용, 구축 시간 등을 고려하여 2013년까지 안전한 112비트 이상의 보안 강도를 가지는 해쉬함수로 변경할 것을 권고한다.

〈표 2-2〉 보안강도별 해쉬함수 분류

보안강도	해쉬함수	권고여부
80 비트 미만	MD5, SHA-1	권고하지 않음
80 비트	HAS-160	

1) TTA : 한국정보통신기술협회, www.tta.or.kr
2) NIST : National Institute of Standards and Technology, http://csrc.nist.gov/publications/fips

보안강도	해쉬함수	권고여부
112 비트	SHA-224	권고함
128 비트	SHA-256	
192 비트	SHA-384	
256 비트	SHA-512	

2.2.2 블록암호 알고리즘

블록암호 알고리즘은 (그림 2-2)와 같이 주민등록번호, 계좌번호 등을 일정한 블록 크기로 나누어, 각 블록을 송 · 수신자간에 공유한 비밀키를 사용하여 암호화하는 방식이다.

(그림 2-2) 블록암호 알고리즘의 개념

대표적인 블록암호 알고리즘으로는 〈표 2-3〉과 같이 국내의 SEED, HIGHT, ARIA, 국외의 TDEA, AES 등이 있다.

〈표 2-3〉 대표적인 블록암호 알고리즘

	SEED	HIGHT	ARIA	AES	TDEA
입출력 크기 (비트)	128	64	128	128	64
비밀키 크기 (비트)	128	128	128/192/256	128/192/256	112(2TDEA)/168(3TDEA)
참조규격	TTA TTAS.KO-12.0004/R1	TTA TTAS.KO-12.0040/R1	KATS[3] KS X.1213-1	NIST[3] FIPS 197	ISO/IE[4] CISO/IEC 18033-3

위에서 나열된 블록암호 알고리즘 역시 비밀키의 길이에 따라 각기 다른 보안강도를 제공한다. 2011년 현재 보안강도별 블록암호 알고리즘은 〈표 2-4〉와 같으며, 권고하는 블록암호 알고리즘은 SEED, HIGHT, ARIA-128/192/256, AES-128/192/256이다. 해쉬함수와 마찬가지로 블록암호 알고리즘 또한 IT환경 변화에 따라 권고 알고리즘이 변경될 수 있다.

〈표 2-4〉 보안강도별 블록암호 알고리즘 분류

보안강도	블록암호 알고리즘	권고여부
80 비트 미만	DES	권고하지 않음
80 비트	2TDEA	
112 비트	3TDEA	
128 비트	SEED, HIGHT, ARIA-128, AES-128	권고함
192 비트	ARIA-192, AES-192	
256 비트	ARIA-256, AES-256	

2.3 정보 송 · 수신 시 적용 가능 암호기술

2.1에서 도출한 정보들을 정보통신망을 통해 송 · 수신 하는 경우 SSL/TLS 등 통신 암호기술을 사용해야 한다.

3) KATS : 기술표준원, http://www.standard.go.kr
4) ISO/IEC : International Organization for Standardization/International Electrotechnical Commission, http://www.iso.org

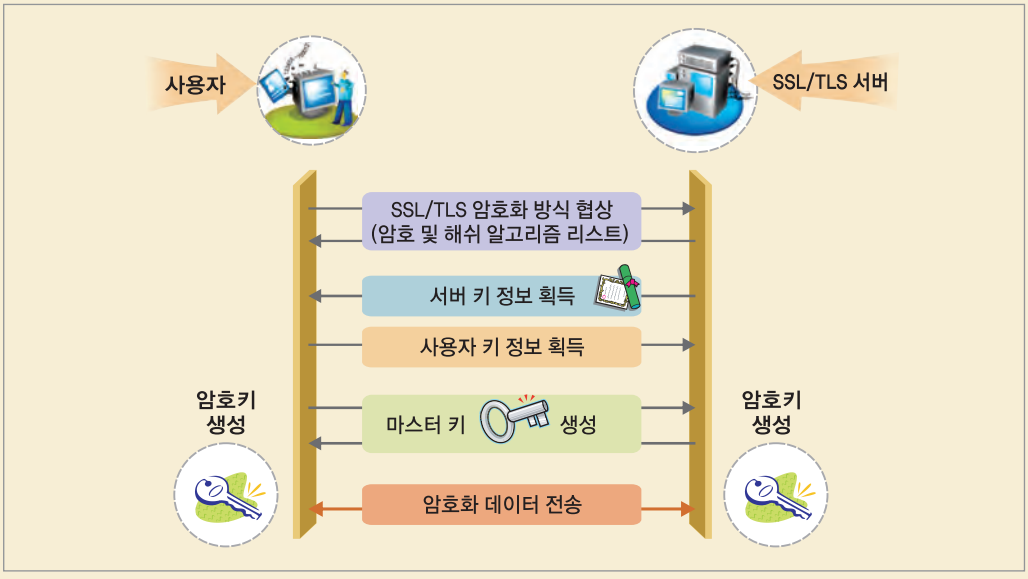
2.3.1 SSL/TLS

개인정보 및 중요한 정보를 웹사이트에서 입력할 때, 거래당사자의 신원 및 거래내용의 위, 변조여부를 확인하고 중요 정보가 제 3자에게 유출되는 것을 막기 위해 (그림 2-3)과 같이 SSL 및 TLS와 같은 통신 암호기술 또는 응용프로그램에서 제공하는 암호화 방법을 이용할 수 있다.

(그림 2-3) SSL/TLS 통신암호 개념

SSL/TLS 프로토콜의 동작 절차는 (그림 2-4)와 같으며, 사용자가 웹 서버에 처음 접속하면 인증서 및 통신 암호화에 이용할 암호키를 생성하기 위한 정보를 공유하고, 이후 공유된 정보를 통해 생성된 암호키를 이용하여 데이터를 암호화하여 전송한다.

SSL/TLS 통신을 하는 경우에는 로그인 페이지 등 보안이 필요한 웹페이지에 접속하면 브라우저 하단 상태 표시줄에 자물쇠 모양의 마크를 확인할 수 있다. 다만, 웹 사이트의 구성 방법에 따라 자물쇠 모양의 마크가 보이지 않을 수도 있다.



(그림 2-4) SSL/TLS 동작 절차

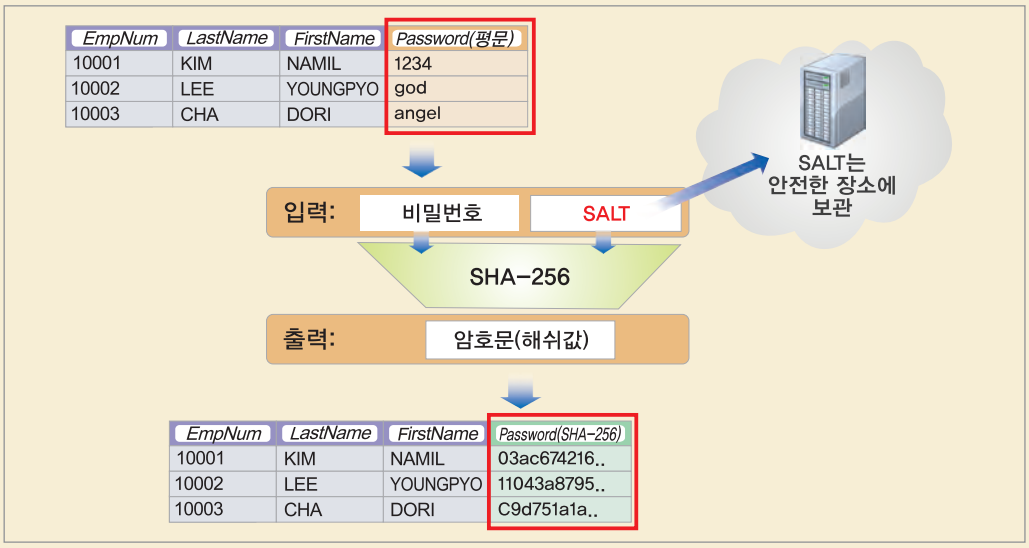
웹 서버에서 SSL/TLS 등의 보안 통신 기능을 적용하기 위한 보다 상세한 정보는 한국인터넷진흥원에서 발간한 「보안서버 구축 안내서」[5]을 참조하기 바란다.

2.4 암호기술 적용 시나리오

2.4.1 해쉬함수

비밀번호와 같이 일방향 암호화가 필요한 정보에 대하여 신규로 해쉬함수를 적용하는 경우는 (그림 2-5)와 같이 처리할 수 있다.

5) http://secsv.kisa.or.kr/kor/security/build.jsp



(그림 2-5) 해쉬함수를 이용한 비밀번호 암호화 개념

(그림 2-5)에서는 비밀번호에 Salt라는 비밀값을 추가하여 해쉬함수에 적용하고 있는데, 이 값은 업체 서버에서 사용자마다 랜덤하게 생성해야 한다. Salt는 단순히 비밀번호만을 해쉬함수에 적용할 경우 비밀번호 사전공격에 취약할 수 있는 문제를 해결하고, 동일한 비밀번호인 경우에도 해쉬값을 다르게 만들어 비밀번호 노출 위협을 막을 수 있다. Salt값를 저장하는 경우에는 비밀번호와 분리하여 따로 저장하여야 하며 서버 프로그램 노출에 따른 Salt 값 노출 위험을 막기 위해, 서버 프로그램은 Salt를 외부에서 호출하여 사용하는 형태로 구현되어져야 한다.

Salt를 추가하는 방법은 위와 같이 간단히 비밀번호와 Salt를 연접하는 방법 외에도 다음과 같은 다양한 방식이 있을 수 있다. 특히, 비밀번호와 Salt를 연접하는 방법은 누구나 쉽게 생각해 낼 수 있는 방법이기 때문에 아래의 다양한 방법을 응용하여 사용하기를 권고하며, 선택한 방법은 외부에 노출되지 않도록 해야 한다.

- hash (SALT || 비밀번호 || SALT)
- hash (SALT || hash (비밀번호))
- hash (비밀번호 || hash (비밀번호 || SALT))

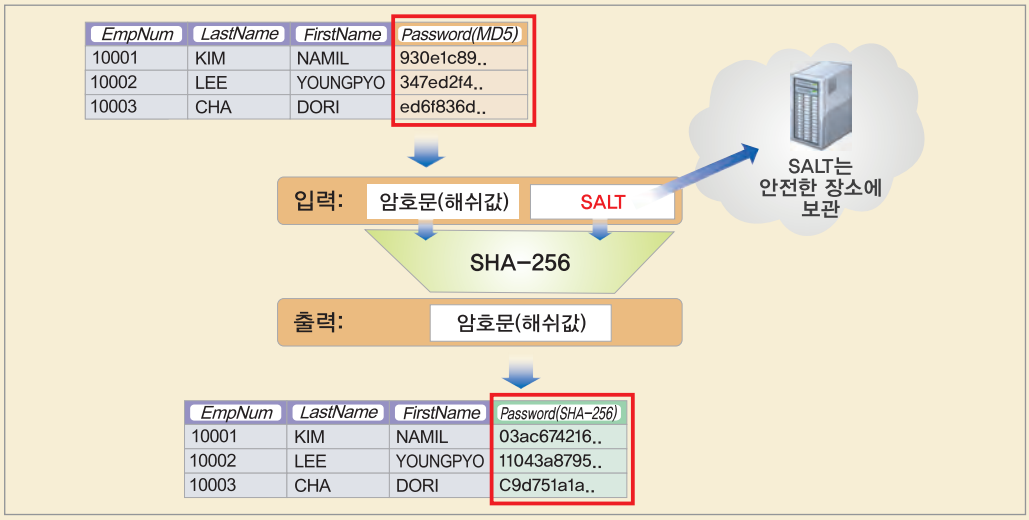
- hash (hash (SALT) || hash (비밀번호))
- hash1 (hash2 (비밀번호 || SALT))

또한, Salt는 사용자마다 랜덤하게 생성하여 할당된 비트열 등과 같이 해커가 예측하기 어렵고 언제라도 변경 가능한 값을 사용하는 것을 권고한다.

2.4.2 해쉬함수 변경

기존에 웹사이트 비밀번호 암호화 시 MD5와 같이 보안강도가 낮은 해쉬함수가 적용되어 있다면 이를 SHA-256와 같은 안전한 해쉬함수로 변경해야 한다. 이를 위해서는 암호화되기 전 원래의 비밀번호를 알아야 하는데, 해쉬된 비밀번호는 복호화가 불가능하므로 모든 사용자가 비밀번호를 다시 입력한 후 이를 SHA-256과 같은 새로운 함수로 해쉬하여 저장하여야 한다. 그러나 모든 사용자의 비밀번호를 일시에 다시 입력받는 것은 불가능하므로 사용자가 다시 로그인하여 비밀번호를 입력하기 전까지는 자체적으로 과거에 해쉬되어 저장된 비밀번호에 변경된 안전한 해쉬함수를 적용해놓아야 한다.

즉, (그림 2-6)과 같이 이미 MD5 등으로 해쉬되어 저장된 비밀번호 해쉬값에 Salt 값을 추가한 후, SHA-256 등의 안전한 해쉬함수로 다시 한번 해쉬해서 저장한다.



(그림 2-6) 두 번 해쉬함수를 이용한 비밀번호 암호화 개념

이렇게 비밀번호가 새로운 해쉬함수로 두 번 해쉬되어 저장된 이후에 사용자가 로그인하여 비밀번호를 입력하면, 이 비밀번호를 새로운 해쉬함수로 한 번만 해쉬하여 저장하는 작업을 수행해야 한다. 이를 위한 절차는 다음과 같다.

① 사용자가 비밀번호를 입력하면 웹 서버는 사용자가 입력한 비밀번호에 MD5(기존에 쓰던 취약한 해쉬함수) → SHA-256(변경된 안전한 해쉬함수) 순서로 두 번의 해쉬함수를 적용한 해쉬값(해쉬값 a)과 SHA-256으로 한번만 해쉬함수를 적용한 해쉬값(해쉬값 b)을 생성한다.

② 해쉬값 a가 기존에 저장되어 있던 해쉬값과 동일한지 확인한다.

③ ②의 값이 동일한 경우, 서버는 해당 사용자가 정당한 사용자라고 판단하고, 기존 두 번 해쉬된 값 대신 ①에서 생성한 해쉬값 b를 사용자의 비밀번호 해쉬값으로 다시 저장한다.

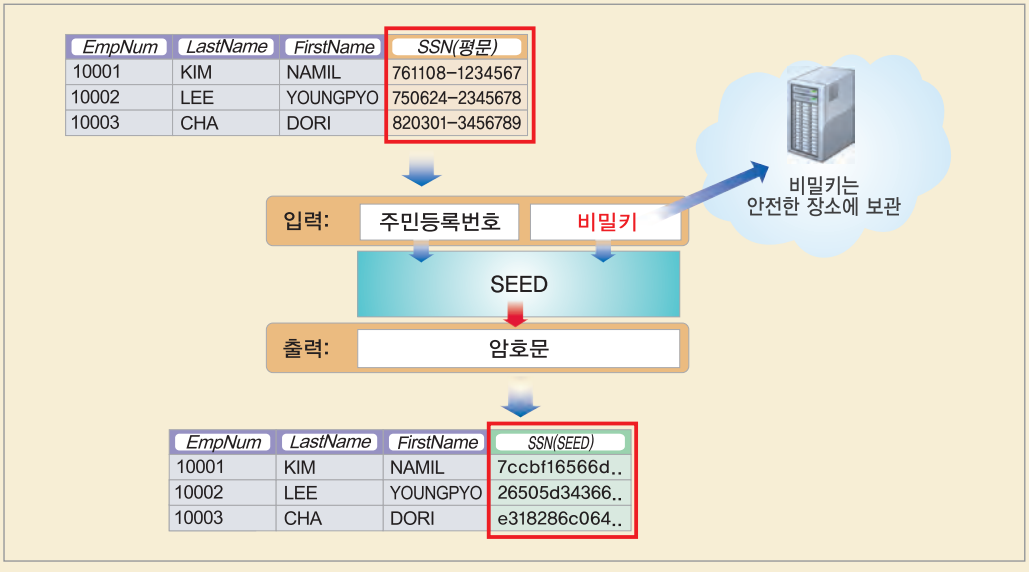
이 방법을 이용하는 경우, 웹 서버는 사용자 로그인 시 해당 사용자의 비밀번호 해쉬값이 두 번 해쉬된 값(해쉬값 a)인지, 한 번 해쉬된 값(해쉬값 b)인지 확인할 수 있어야 한다. 이를 위해 서버는 사용자 로그인 시간을 활용할 수 있는데, 서버가 해쉬함수를 변경한 시점 이후로 처음 로그인한 사용자의 경우에는 해쉬값 a가 저장되어 있는 사용자이므로, 해쉬값 b로 바꾸어 저장해야 한다. 반대로, 사용자의 최근 로그인 시간이 해쉬함수 변경 이후인 사용자의 경우에는 새로운 해쉬함수로 한 번 해쉬한 해쉬값 b가 저장되어 있는 사용자라 판단할 수 있다.

사용자 로그인 시간 외에도 새로운 DB 컬럼을 생성하여 해당 컬럼에 해쉬함수 변경 여부를 기록할 수 있다. 즉, 해쉬함수 변경이 이뤄지지 않은 사용자는 플래그를 0으로, 변경된 사용자는 1로 기록해서 구분할 수 있다. 서버가 해쉬함수를 변경하는 시점에서 모든 사용자의 플래그를 0으로 설정하고, 이후 사용자 로그인 시 해당 사용자의 플래그를 확인해서 해당 플래그가 0인 사용자는 (그림 2-5)의 과정을 거친 후 플래그를 1로 재설정하고, 플

래그가 1인 사용자는 비밀번호에 SHA-256을 적용해 해쉬값을 확인한다.

2.4.3 블록암호 알고리즘

주민등록번호 등에 대해서 신규로 블록암호 알고리즘을 적용하는 시나리오는 (그림 2-7)와 같다. 이 때, 암호화에 사용되는 비밀키는 3.2.2에서 제시된 방법을 통해 안전하게 보관되어 쉽게 노출되지 않도록 하여야 한다.

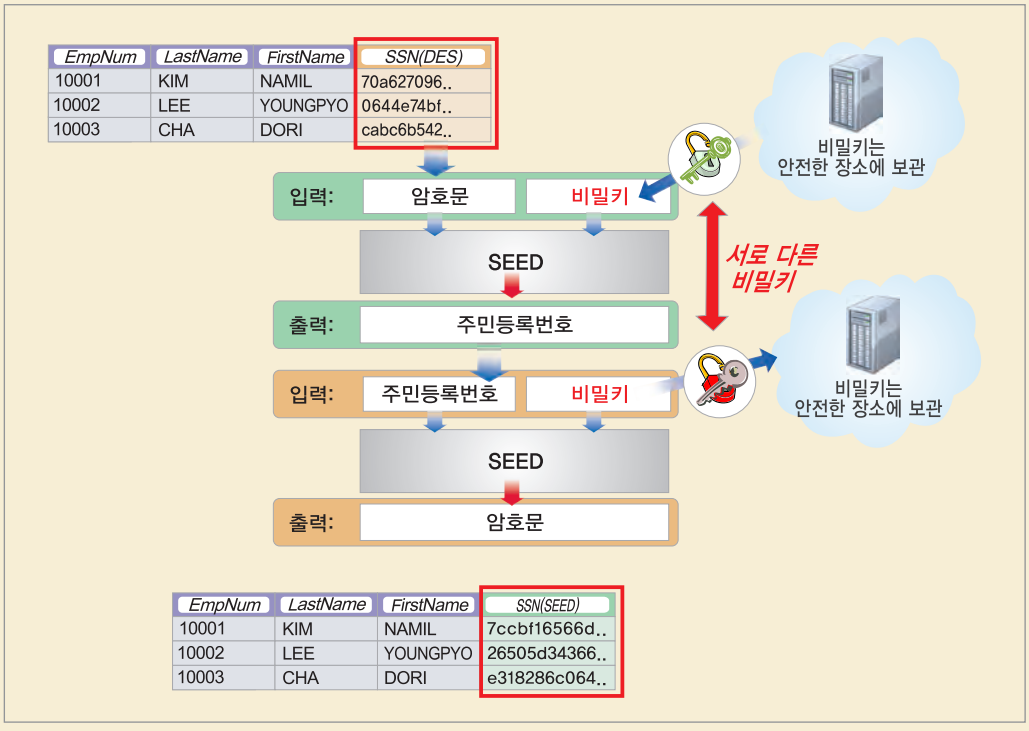


(그림 2-7) 블록암호 알고리즘을 이용한 암호화 개념

2.4.4 블록암호 알고리즘 변경

기존에 블록암호 알고리즘으로 DES 등이 적용되어 있고, 이를 SEED 등과 같은 안전한 블록암호 알고리즘으로 변경하고자 하는 경우에는 (그림 2-8)와 같은 시나리오를 적용할 수 있다. 즉, 기존 DB에 DES로 암호화 되어 저장되어 있던 암호문들을 복호화 한 후, DES에 사용되었던 비밀키와 다른 새로운 비밀키를 생성하고 이를 이용하여 SEED로 암호화

한 후 다시 DB에 저장한다. 이 시나리오는 암호 알고리즘의 변경 뿐 아니라, 비밀키 노출 가능성, 안전성 저하 등의 이유로 비밀키를 변경하는 경우에도 동일하게 적용된다.



(그림 2-8) 블록암호 알고리즘 변경 개념

Korea Internet & Security Agency

3. 안전한 비밀번호 및 암호키 관리 방안

3.1 안전한 비밀번호의 생성 · 변경 · 이용 · 검증 방법

3.2 암호키 관리 방안

3 안전한 비밀번호 및 암호키 관리 방안

3.1 안전한 비밀번호의 생성 · 변경 · 이용 · 검증 방법

본 절에서는 웹사이트 사용자의 로그인 수단으로 비밀번호를 사용하는 경우, 비밀번호 생성 · 변경 · 이용 등 비밀번호 생명 주기에 따라 비밀번호를 안전하게 관리하기 위한 기준 및 방안을 명시한다. 다만, 본 장에서 제시하는 비밀번호 기준은 2011년도 IT서비스 환경을 고려하여 제시하고 있으므로, 컴퓨터의 성능 향상, 해킹 기술 발달 등에 따라 주기적으로 검토, 갱신될 수 있다.

3.1.1 비밀번호 생성 단계

안전한 비밀번호는 제3자가 쉽게 추측할 수 없으며, 시스템에 저장되어 있는 사용자 정보 또는 인터넷을 통해 전송되는 정보를 해킹하여 알아낼 수 없거나, 알아낸다 하더라도 많은 시간이 요구되는 비밀번호를 말한다. 업체는 자사의 서비스에 접근하기 위해 사용자가 비밀번호를 생성하는 경우, 안전한 비밀번호를 사용하도록 유도하여야 한다.

안전한 비밀번호의 문자구성 및 길이 조건은 다음과 같이 사용할 것을 권고한다.

〈안전한 비밀번호〉

- 3가지 종류 이상의 문자구성으로 8자리 이상의 길이로 구성된 비밀번호
- 2가지 종류 이상의 문자구성으로 10자리 이상의 길이로 구성된 비밀번호

※ 문자 종류는 알파벳 대문자와 소문자, 특수기호, 숫자의 4가지임

3.1.2 비밀번호 변경 단계

업체는 모든 사용자에게 주기적으로 비밀번호를 변경하도록 유도하여 비밀번호의 노출 위협을 최소화하여야 한다. 이를 위해 업체는 반기 혹은 분기 동안 변경되지 않은 비밀번

호 사용자가 로그인 한 경우, 비밀번호를 변경할 수 있는 화면을 보여주어야 한다. 일반적으로 사용자 비밀번호 변경 주기는 3개월에서 6개월 이하로 설정해야 한다.

사용자가 비밀번호 변경을 필요로 하거나 요청하였을 때 언제든지 비밀번호를 변경할 수 있는 기능을 제공해야 한다. 또한, 비밀번호 변경 시, 이전에 사용하지 않은 새로운 비밀번호를 사용하도록 유도해야 하며 변경된 비밀번호는 이전 비밀번호와 연관성이 없어야 한다.

3.1.3 비밀번호 이용 단계

업체는 회원으로 가입된 사용자의 비밀번호를 2.2에서 설명한 해쉬함수를 적용하여 저장하여야 한다. 또한, 오직 인가된 관리자만이 사용자의 비밀번호가 저장된 시스템에 접근할 수 있어야 하며, 해당 시스템은 외부의 침입으로부터 안전한 장소에 보관해야 한다.

3.1.4 비밀번호 검증 기능

업체는 서비스 종류, 취급하는 개인정보의 종류, 개인정보 노출시 파급 효과 등을 고려하여 적합한 비밀번호 정책을 수립하고 공시하여야 한다. 비밀번호 정책에는 최소 비밀번호 길이 및 문자조합, 변경 주기 등을 포함해야 한다.

비밀번호 정책이 수립되면, 업체는 사용자 비밀번호가 자사의 비밀번호 정책을 만족하는지 확인할 수 있는 비밀번호 검증 기능을 구현하여 적용해야 한다. 비밀번호 검증 기능은 사용자가 설정하는 비밀번호가 업체의 비밀번호 정책을 만족하는지 여부를 직접적으로 검토하여, 보안성이 떨어지는 비밀번호는 변경하도록 유도할 수 있다.

다음 조건에 해당하는 비밀번호는 취약한 비밀번호이므로, 비밀번호 검증 기능에서 아래 조건에 해당하는 비밀번호인지를 검증하여 이를 사용하지 않도록 구현할 것을 권고한다.

	취약한 비밀번호 조건
문자구성 및 길이 조건	– 2가지 종류 이하의 문자구성으로 7자리 이하의 길이로 구성된 비밀번호 – 문자구성과 관계없이 7자리 이하 길이로 구성된 비밀번호 ※ 문자 종류는 알파벳 대문자와 소문자, 특수기호, 숫자의 4가지임
특정 정보 이용 및 패턴 조건	– 한글, 영어 등을 포함한 사전적인 단어로 구성된 비밀번호 ※ 스펠링을 거꾸로 구성한 비밀번호도 포함 – 널리 알려진 단어로 구성된 비밀번호 ※ 컴퓨터 용어, 사이트, 기업 등의 특정 명칭으로 구성된 비밀번호도 포함 – 사용자 ID를 이용한 비밀번호 ※ 사용자 ID 혹은 사용자 ID를 거꾸로 구성한 비밀번호도 포함 – 제3자가 쉽게 알 수 있는 개인정보를 바탕으로 구성된 비밀번호 ※ 가족, 생일, 주소, 휴대전화번호 등을 포함하는 비밀번호 – 패턴이 존재하는 비밀번호 ※ 동일한 문자의 반복 : ex) aaabbb, 123123 ※ 키보드 상에서 연속한 위치에 존재하는 문자들의 집합 : ex) qwerty, asdfgh ※ 숫자가 제일 앞이나 제일 뒤에 오는 구성의 비밀번호 : ex) security1, may12 – 숫자와 영문자를 비슷한 문자로 치환한 형태를 포함한 구성의 비밀번호 ※ 영문자 “O” 를 숫자 “0” 으로, 영문자 “l” 을 숫자 “1” 로 치환 등의 비밀번호

사전적 단어를 포함하는 비밀번호, 특정한 패턴을 가지는 비밀번호 등 비밀번호 특정정보 이용 및 패턴 조건에 대한 검증 기능은 KISA에서 배포하는 「비밀번호 검증 S/W」를 활용할 수 있다. 「비밀번호 검증 S/W」는 사용자가 입력한 비밀번호에 대한 안전성을 검사하여 그 결과를 알려주는 기능을 제공한다. 이 S/W의 기본 알고리즘은 C언어로 개발되었고, PHP Extention과 JNI 기능을 활용하여 PHP 및 JSP에서 사용할 수 있도록 구현되어 있다. 본 S/W는 기업 및 단체를 대상으로 보급하고 있으며 KISA 홈페이지에서 소프트웨어 사용 동의 후 내려 받을 수 있다.

3.2 암호키 관리 방안

암호키 관리는 암호를 효과적으로 사용하기 위해 필수적인 요소이다. 만약 민감한 정보를 암호화하기 위해 사용한 암호키가 공격자에게 노출되었다면 공격자는 해당 암호문을 쉽게 복호하여 민감한 정보에 접근할 수 있게 된다. 즉, 암호키 관리를 잘못하는 경우 아무리 보안성 높은 암호 알고리즘을 적용하는 경우에도 암호화된 중요 정보가 쉽게 유출될 수 있다.

3.2.1 암호키 사용기간 및 유효기간

암호키는 보안을 위해 사용기간과 유효기간을 구분할 필요가 있다. 암호키의 사용기간은 사용자 또는 관리자가 암호키를 사용할 수 있도록 허용된 기간이고, 유효기간은 사용기간이 완료된 이후라도 추후 복호화를 위해 해당 암호키를 사용하도록 허용된 기간이다. 키의 사용기간 및 유효기간을 설정할 때는 키 노출을 야기하는 위험 요소와 키 노출에 따른 비용 등을 고려하여야 한다.

암호키의 사용기간은 최대 2년, 유효기간은 최대 5년으로 설정할 수 있다.

3.2.2 암호키 생성 방법의 예

암호키를 생성하는 방법은 비밀번호, 난수발생기 등을 이용하는 다양한 방법들이 있다. 다음은 RSA, NIST 등에서 명시하고 있는 암호키 생성 방법들로, 현재 국제적으로 널리 쓰이고 있는 방법들이다.

- RSA PKCS#5 v2.1
  - Password-Based Cryptography Standard
    - Password-Based Key Derivation Function 1
    - Password-Based Key Derivation Function 2

※ URL : http://www.rsa.com/rsalabs/node.asp?id=2127

- NIST SP 800-108
  - Recommendation for Key Derivation Using Pseudorandom Functions
    - KDF in Counter Mode
    - KDF in Feedback Mode
    - KDF in Double-Pipeline Iteration Mode

    ※ http://csrc.nist.gov/publications/nistpubs/800-108/sp800-108.pdf

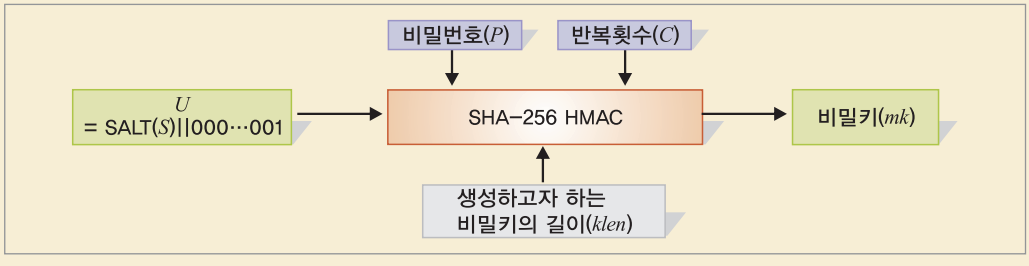
- NIST SP 800-132
  - Recommendation for Password-Based Key Derivation Part 1: Storage Applications
    - Password-Based Key Derivation Function

    ※ http://csrc.nist.gov/publications/nistpubs/800-132/nist-sp800-132.pdf

위 방법들 중 RSA PKCS#5 v2.1의 Password-based key derivation function(PBKDF) 1은 취약한 해쉬함수 MD2, MD5 그리고 SHA-1을 사용하는 표준이기 때문에 사용하지 않을 것을 권고한다.

본 안내서에서는 NIST SP 800-132의 PBKDF에 대해서 SHA-256에 기반한 HMAC을 이용하여 128비트 비밀키를 생성하는 예제를 소개한다.

PBKDF은 비밀번호, SALT, 반복횟수, 생성하고자 하는 비밀키의 길이를 입력으로 비밀키를 출력한다. (그림 3-1)는 SHA-256에 기반한 HMAC을 이용하는 PBKDF의 일반적인 구조를 나타낸다. SALT는 난수발생기 등을 이용하여 생성하고, 최소 128비트 이상의 길이를 사용하는 것을 권고한다. 반복횟수는 최소 1,000번 이상을 권고한다.



(그림 3-1) Password-based key derivation function의 일반적인 구조

PBKDF에 대해서 SHA-256에 기반한 HMAC을 이용하여 128비트 비밀키를 생성하는 과정은 다음과 같다.

① U를 SALT의 오른쪽에 32비트 0x00000001을 연접한 값으로, 출력변수 T는 0으로 설정한다.
② SHA-256 HMAC에 비밀번호와 U를 입력하여 256비트 MAC 값을 출력한다. MAC 값은 다음과 같이 계산된다.

$$MAC = SHA\text{-}256(K \oplus opad) \| SHA\text{-}256((K \oplus ipad) \| U))$$

함께 사용된 파라미터 값들은 다음과 같다.

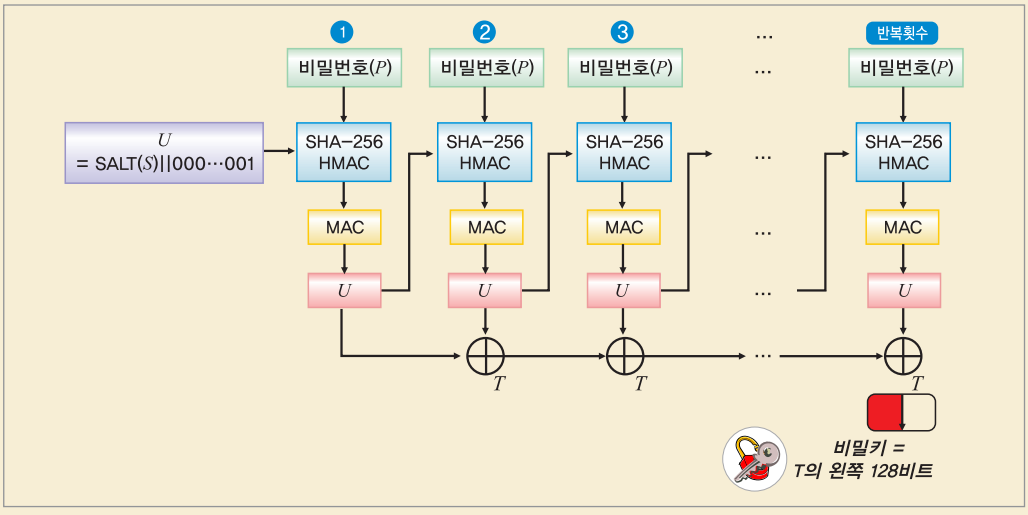
- opad = 0x5c5c5c 5c5c (512비트)
- ipad = 0x363636 ... 3636 (512비트)
- K = 비밀번호 || 00 ...00 (비밀번호 길이 〈 512비트인 경우)
  = 비밀번호 (비밀번호 길이 = 512비트인 경우)
  = SHA-256(비밀번호) || 00 ...00 (비밀번호 길이 〉 512비트인 경우)

K값의 경우, 위에서 설명한 것과 같이 비밀번호가 512비트인 경우 비밀번호 값을 그대로 사용하고, 512비트보다 작은 경우 0을 연접하여 전체 길이가 512비트가 되도록 한다. 또한 비밀번호가 512비트보다 클 경우 SHA-256으로 해쉬한 값 256비트에 0을 256비트 연접하여 총 512비트가 되도록 한다. 다만, 일반적으로 사용자가 입력하는 비밀번호는 512비트를 넘는 경우가 드물기 때문에 대부분 비밀번호 오른쪽에 0을 연접해서 K값을 만드는 방식을 사용한다.

※ 자세한 내용은 NIST FIPS 198-1 The Keyed-Hash Message Authentication Code(HMAC) 참조.
URL : http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf)

③ U를 MAC 값으로 대체하고, T는 U와 이전 단계의 T를 XOR한 값으로 대체한다.
④ ②, ③의 과정을 (반복횟수-1) 만큼 반복한다.
⑤ 최종 출력된 T의 왼쪽 128비트(즉, 생성하고자 하는 비밀키의 길이)를 비밀키로 사용한다.

(그림 3-2)는 위의 ①~⑤과정을 도식화한 것이다.



(그림 3-2) SHA-256 HMAC을 이용하여 128비트 비밀키를 생성하는 과정

3.2.2 암호키 저장 방법의 예

암호키는 서버 또는 하드웨어 토큰에 저장되어질 수 있다. 암호키를 저장하는 서버는 웹 서버 또는 DB 서버와 같은 서버일 수 있으나, 물리적으로 분리되어 있는 서버를 사용하는 것을 권고한다. 하드웨어 토큰은 저장된 정보가 위 · 변조 또는 외부로 노출되기 어려운 장치로, 스마트카드, USB 토큰 등의 보안토큰(HSM, Hardware Security Module)을 의미한다.

암호키 저장은 환경에 따라 다양한 방법이 적용될 수 있다. 본 안내서에서는 서비스 및 DB에서 사용되는 암호키와 직원 PC에서 사용되는 암호키에 대한 저장 방법 예시를 제시한다.

예제 1) 서비스 및 DB에서 사용되는 암호키

● 하나의 암호키를 사용하는 경우

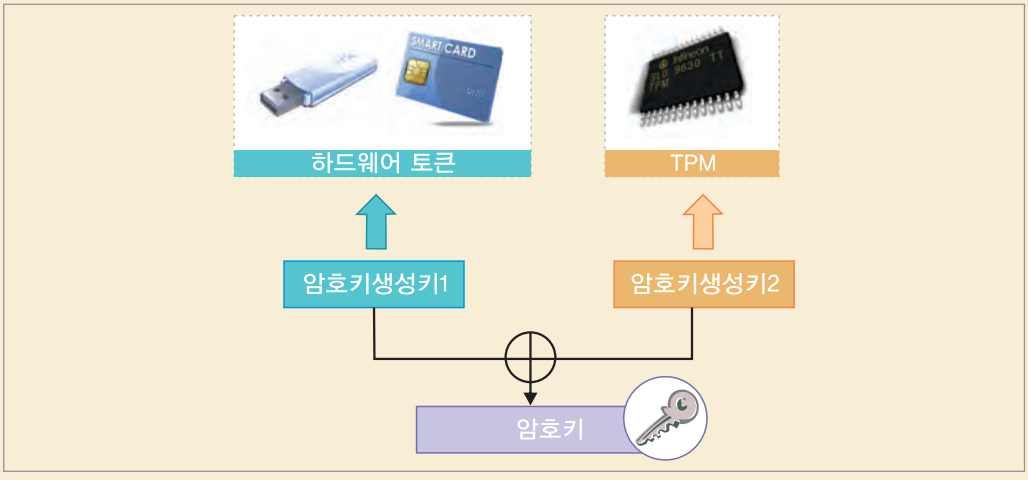
암호키는 앞에서 언급한 바와 같이 서버 또는 하드웨어 토큰에 저장한다. 만일, 사용 중인 서버가 Trusted Platform Module(TPM)을 지원한다면 TPM에 암호키를 저장할 수 있다.

(그림 3-3) 하나의 암호키 저장 예

● 두 개 이상의 암호키생성키 사용하는 경우

암호키생성키란 실제 암호화에 사용할 암호키를 생성하기 위한 키들을 의미한다. 암호키는 두 개 이상의 암호키생성키들에 특정 연산을 적용하여 생성된다. 여기에서 암호키 생성을 위한 특정 연산은 암호키생성키들간의 배타적 논리합(exclusive-or) 등의 연산이 될 수 있으며, 이 방법은 외부로 노출되지 않도록 해야 한다.

두 개 이상의 암호키생성키를 사용하는 경우, 각각의 암호키생성키들을 물리적으로 다른 공간에 저장하는 방법을 사용할 수 있다. 즉, 하나의 암호키생성키는 서버에, 다른 암호키생성키는 하드웨어 토큰에 저장하여 사용할 수 있다. 이 경우 공격자가 두 개의 암호키생성키들을 모두 획득해야하기 때문에 하나의 암호키를 사용 · 저장하는 방법보다 좀 더 안전하다.



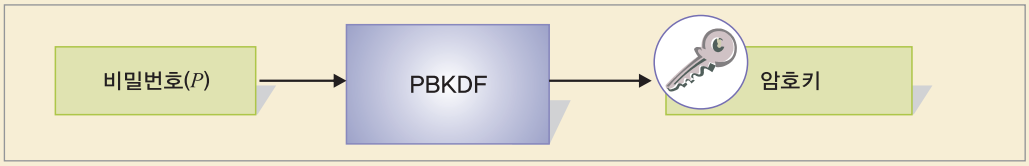
(그림 3-4) 두 개의 암호키생성키 저장 예

예제 2) 직원 PC에서 사용되는 암호키

● 한명의 사용자가 한 개의 암호키를 사용하는 경우(개인PC)

한명의 사용자가 자신의 PC에 고객의 개인정보, 중요정보 등을 암호화하여 저장하기 위해 암호키를 생성해야하는 경우가 있다. 이 경우에는 사용자 본인만이 암호키를 생성해 낼 수 있도록 3.2.2에 명시되어 있는 PBKDF를 사용할 수 있다. 이 경우에는 사용자 비밀번호를 통해 암호키가 생성되므로 암호키를 PC에 저장할 필요가 없다.

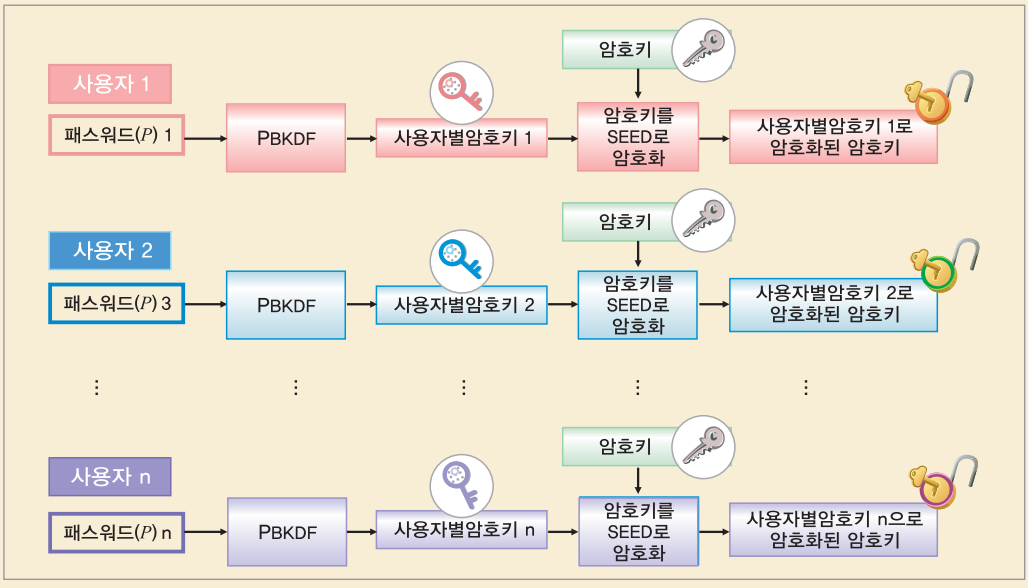
하지만 강도가 낮은 비밀번호를 사용할 경우에는 사전공격 등에 취약한 방법이며, 다수의 사용자가 이 방법을 사용하기 위해 비밀번호를 공유할 경우, 관리가 어려워 외부로 노출될 가능성이 높아진다.



〈그림 3-5〉 PBKDF를 이용한 암호키 생성 예

● 다수의 사용자가 한 개의 암호키를 사용하는 경우(공용PC)

한 개의 암호키로 암호화된 정보들을 다수의 사용자가 공유하기 위한 방법으로는 해당 암호키를 사용자마다 다른 "사용자별암호키"로 다시 암호화하는 방법을 사용할 수 있다. 사용자별암호키는 PBKDF를 이용하여 생성할 수 있으며, 이때 사용되는 비밀번호는 사용자마다 각각 다른 비밀번호를 사용하여야 한다. 암호키는 사용자별암호키로 암호화되어 저장되고, 사용시에는 사용자가 비밀번호로 자신의 사용자별암호키를 생성하여 암호화된 암호키를 복호화하여 사용한다. 이 방법 역시 강도가 낮은 비밀번호를 사용할 경우에는 사전공격 등에 취약하다.



(그림 3-6) 사용자별암호키 생성 및 암호키 암호화의 예

Korea Internet & Security Agency

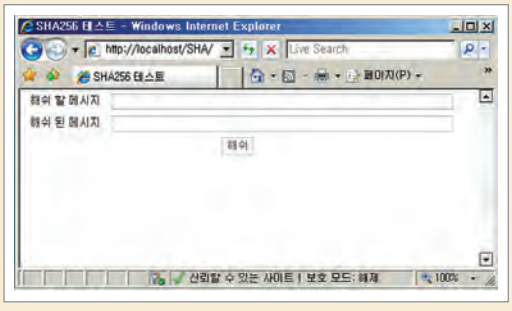
4. 암호기술 구현방법

4 암호기술 구현방법

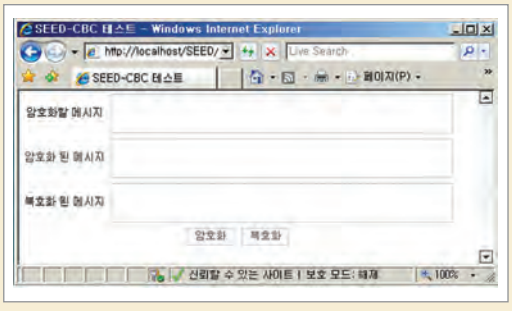
최근 국내 정보통신망법 하위 고시인 '개인정보의 기술적 · 관리적 보호조치 기준' 개정으로 정보통신 서비스 제공자들의 고객 개인정보 암호화 저장이 의무화됨에 따라, 주민등록번호, 신용카드번호 및 계좌번호 등을 저장 · 관리하는 개인정보 데이터베이스 암호화에 대한 관심이 높아지고 있다.

이에, KISA는 블록암호알고리즘 SEED 및 해쉬함수 SHA-256의 ASP, JSP용 암호라이브러리를 배포하고 있다(SEED의 경우, CBC 운영모드를 사용). 본 장에서는 SEED와 SHA-256에 대한 암호라이브러리를 이용하여 개인정보를 암호화하는 방법을 예제로 소개한다.

다음의 (그림4-1), (그림4-2)는 웹페이지 상에서 평문과 암호문을 입력받아 각각 SEED, SHA-256 알고리즘을 이용하여 입력 데이터를 암 · 복호화 및 해쉬한 결과를 보여주는 예제이다. 본 예제는 암 · 복호화 및 해쉬한 결과를 직접적으로 보여주기 위해 결과를 웹페이지에 출력하는 형태로 구성하였으나, 실제 개인정보를 암호화 또는 해쉬하여 DB에 저장하기 위해서는 웹페이지 출력 명령어 부분을 DB에 저장하는 명령어로 변경해 주면 된다. 또한 암호화한 개인정보를 복호화하기 위해서는 암호문을 입력받는 명령어를 DB에서 해당 암호문을 호출하는 명령어로 변경해주면 된다.



〈그림 4-1〉 SHA-256 이용 예제 웹페이지



〈그림 4-2〉 SEED-CBC 이용 예제 웹페이지

4.1 SHA-256을 이용한 개인정보 데이터베이스 암호화

4.1.1. 해쉬함수 SHA-256

SHA-256은 NIST에서 개발한 미국 표준 해쉬함수(FIPS 180-3) 중 하나이다. SHA-256은 임의의 길이를 가지는 입력 메시지를 512비트 블록 단위로 처리하여 256비트의 해쉬한 값을 출력한다. SHA-256은 임의의 길이를 갖는 입력 메시지를 512비트의 배수로 만들기 위한 덧붙이기를 방법을 사용하는데, 해당 덧붙이기 방법은 FIPS 180-3에 명시되어 있다.

4.1.2 SHA-256 해쉬 예(웹페이지)

다음은 SHA-256을 이용해 입력된 개인정보를 해쉬하는 웹페이지의 소스이다.

- index.html

다음은 SHA-256 해쉬 예시에 대한 index.html 소스이다. 본 소스에서는 ASP를 예시로 들고 있기 때문에, 해쉬 버튼 클릭 시 실행되는 웹페이지가 sha256hash.asp이다. JSP의 경우 이 부분을 sha256hash.jsp로 변경하여 사용한다.

```
<html>
<head>
    <title>SHA256 테스트</title>
    <meta http-equiv="content-type" content="text/html; charset=euc-kr" />
    <link rel="stylesheet" href="style.css" type="text/css">
    <script src="ajax.js" language="JavaScript"></script>
    <script language="JavaScript">
        function sha256( URL )
        {
            var form = document.getElementById( "sha256_form" );
            form.oCipherText.value = (GetHttpRequest( URL, "iPlainText=" +
 encodeURIComponent(form.iPlainText.value))).trim();
```

```
        }
    </script>
</head>
<body>
<form name="sha256_form">
    <table border="0" width="500" celpadding="5">
        <tr>
            <td width="100">해쉬 할 메시지</td>
            <td width="*"><input type="text" name="iPlainText" style="width:100%;" /></td>
        </tr>
        <tr>
            <td>해쉬 된 메시지</td>
            <td><input type="text" name="oCipherText" style="width:100%;"
            readonly="readonly" /></td>
        </tr>
        <tr>
            <td colspan="2"><input type="button" value="해쉬" name="enc" onclick=
"JavaScript:sha256('sha256hash.asp');" /></td>
    ☞ JSP의 경우 sha256hash.asp를 sha256hash.jsp로 변경하여 사용
        </tr>
    </table>
</form>
</body>
</html>
```

4.1.3 ASP 구현 예

- **SHA256.dll의 레지스트리에 등록**

- SHA256.dll 파일을 "%windir%\system32" 디렉토리에 저장
- SHA-256 모듈을 레지스트리에 등록하는 명령어(RegSvr32)

```
regsvr32.exe %windir%₩system32₩SHA256.dll ☞ SHA256.dll을 레지스트리에 등록
```

● sha256hash.asp

SHA-256 해쉬를 수행하는 sha256hash 페이지의 ASP 소스이다. 해쉬 수행 명령어(오브젝트.hash)로 입력 메시지를 입력받아 해쉬를 수행한다. DB 적용 시에는 Response.Write 명령어 대신 해쉬된 값을 DB에 저장하도록 명령어를 수정하여 사용한다.

```
<%@ codepage="65001" language="VBScript" %>
<%
    ' 유니코드로 파라미터가 전송되므로 값을 코드페이지를 유니코드로 설정
    ' 65001 : 유니코드 (UTF-8)
    sPlainText = Trim( Request("iPlainText") ) ☞ 웹페이지에서 메시지를 가져오는 명령어
    set oSha = Server.CreateObject( "SHA.sha256" ) ☞ SHA256.dll에 정의된 오브젝트 생성
    Response.Write( oSha.hash(sPlainText) )
    ☞ 해쉬 수행 및 웹페이지에 해쉬한 값을 출력하는 명령어
    ☞ DB 저장 : oSha.hash(sPlainText)의 결과를 DB에 저장하는 명령어로 수정
    ☞ ex) DBconn.Execute( "INSERT INTO 저장할 테이블(필드명) values
       ( '" & oSha.hash(sPlainText) & "' )" )
    set oSha = nothing
%>
```

4.1.4 JSP 구현 예

● SHA256.class의 등록

- JSP에서 불러올 class 파일들이 위치하는 폴더에 "KISA" 폴더를 생성하고, SHA256.class를 저장

```
c:₩webapps₩ROOT₩WEB-INF₩classes ☞ class 파일들이 위치하는 폴더
c:₩webapps₩ROOT₩WEB-INF₩classes₩KISA ☞ KISA 폴더 생성 후 SHA256.class 파일을 저장
```

● sha256hash.jsp

SHA-256 해쉬를 수행하는 sha256hash 페이지의 JSP 소스이다. 해쉬 수행 명령어(오브젝트.hash)로 입력 메시지를 입력받아 해쉬를 수행한다. DB 적용 시에는 out.print 명령어 대신 해쉬된 값을 DB에 저장하도록 명령어를 수정하여 사용한다.

```
<%@page language="java" contentType="text/html;charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@page import="sun.misc.BASE64Encoder"%>
<%@page import="java.io.*"%>
<%@page import="KISA.SHA256"%>
<%
    String sPlainText = request.getParameter( "iPlainText" );
    ☞ 웹페이지에서 메시지를 가져오는 명령어
    SHA256 s = new SHA256( sPlainText.getBytes() );
    ☞ SHA256.class에 정의된 오브젝트 생성
    BASE64Encoder Base64Encoder = new BASE64Encoder();
    ☞ Base64인코더 오브젝트 생성
    out.print( Base64Encoder.encode(s.GetHashCode()) );
    ☞ 해쉬 수행 및 웹페이지에 해쉬한 값을 출력하는 명령어
    ☞ DB 저장 : Base64Encoder.encode(s.GetHashCode())의 결과를 DB에 저장하는
                명령어로 수정
    ☞ ex) DBconn.Execute( "INSERT INTO 저장할 테이블(필드명) values
       ( '" & Base64Encoder.encode(s.GetHashCode()) & "' )" )
%>
```

4.2. SEED를 이용한 개인정보 데이터베이스 암호화(CBC 운영모드)

4.2.1 SEED 및 CBC 운영모드

SEED(TTAS.KO-12.0004/R1)는 1999년 2월 한국인터넷진흥원을 중심으로 국내 암호 전문가들이 참여하여 순수 국내기술로 개발한 블록암호 알고리즘이다. SEED는 128비트의 비밀키를 사용하여 128비트 평문을 128비트 암호문으로 암호화하는 128비트 블록암호 알고리즘이다.

임의의 길이를 갖는 평문을 SEED로 암호화하기 위해서는 평문을 128비트 블록단위로 처리해주는 운영모드와 평문을 128비트의 배수로 만들기 위한 덧붙이기 방법을 사용하여야 한다. 본 예시에서는 CBC 운영모드와 PKCS #5 패딩을 사용한다.

4.2.2 SEED-CBC 암 · 복호화 예(웹페이지)

다음은 SEED-CBC 암 · 복호화를 위한 웹페이지의 소스이다.

- index.html

다음은 SEED-CBC 암·복호화 예시에 대한 index.html 소스이다. 본 소스에서는 ASP를 예시로 들고 있기 때문에, 암·복호화 버튼 클릭 시 실행되는 웹페이지가 SeedEncryption.asp, SeedDecryption.asp이다. JSP의 경우 이 부분을 SeedEncryption.jsp, SeedDecryption.jsp로 변경하여 사용한다.

```
<html>
<head>
    <title>SEED-CBC 테스트</title>
    <link rel="stylesheet" href="style.css" type="text/css">
    <meta http-equiv="content-type" content="text/html; charset=euc-kr" />
    <script src="./ajax.js" language="JavaScript"></script>
    <script language="JavaScript">
        function SeedCBC( URL, FORM, ACT )
        {
            if( ACT == "enc" )
                FORM.oCipherText.value = ( GetHttpRequest( URL, "iPlainText=" +
encodeURIComponent(FORM.iPlainText.value)) ).trim();
            else if( ACT == "dec" )
                FORM.oPlainText.value = ( GetHttpRequest( URL, "oCipherText=" +
encodeURIComponent(FORM.oCipherText.value)) ).trim();
        }
    </script>
</head>
<body>
<form name="seed_form">
    <table border="0" width="500" celpadding="5">
        <tr>
```

```
            <td width="100">암호화할 메시지</td>
            <td width="*">
                <textarea name="iPlainText" style="width:100%;" rows="3"></textarea>
            </td>
        </tr>
        <tr>
            <td>암호화 된 메시지</td>
            <td>
                <textarea name="oCipherText" style="width:100%;" rows="3"
readonly="readonly"></textarea>
            </td>
        </tr>
        <tr>
            <td>복호화 된 메시지</td>
            <td>
                <textarea name="oPlainText" style="width:100%;" rows="3"
readonly="readonly"></textarea>
            </td>
        </tr>
        <tr>
            <td colspan="2" align="center">
                <input type="button" value="암호화" name="enc" onclick=
"JavaScript:SeedCBC( 'SeedEncryption.asp', document.seed_form, 'enc');" /> 
    ☞ JSP의 경우 SeedEncryption.asp를 SeedEncryption.jsp로 변경하여 사용
                <input type="button" value="복호화" name="dec" onclick=
"JavaScript:SeedCBC( 'SeedDecryption.asp', document.seed_form, 'dec');" />
    ☞ JSP의 경우 SeedDecryption.asp를 SeedDecryption.jsp로 변경하여 사용
            </td>
        </tr>
    </table>
</form>
</body>
</html>
```

4.2.3 ASP 구현 예

다음은 ASP에서 SEED-CBC를 이용해 입력받은 개인정보를 암호화하기 위한 방법이다.

● **SeedCBC.dll의 레지스트리 등록**

- SeedCBC.dll 파일을 "%windir%\system32" 디렉토리에 저장
- SEED-CBC 모듈을 레지스트리에 등록하는 명령어(RegSvr32)

```
regsvr32.exe %windir%₩system32₩SeedCBC.dll ☞ SeedCBC.dll을 레지스트리에 등록
```

● **키파일(key.dat)**

- SEED-CBC 암 · 복호화에 사용할 128비트 초기값(IV)과 128비트 비밀키값(MK)이 저장된 파일
- 키파일은 외부에 노출되면 안 되므로 로컬시스템의 안전한 위치에 저장해야 함
- 개발자는 반드시 IV, MK의 바이트값을 수정해서 사용해야 함

```
IV[16] = {0x26,0x8D,0x66,0xA7,0x35,0xA8,0x1A,0x81,0x6F,0xBA,0xD9,0xFA,0x36,0x16,0x25,0x01}
☞ CBC모드를 위함 128비트 초기화값 (개발자는 위 바이트값을 수정해서 사용해야 함)
MK[16] = {0x88,0xE3,0x4F,0x8F,0x08,0x17,0x79,0xF1,0xE9,0xF3,0x94,0x37,0x0A,0xD4,0x05,0x89}
☞ 128비트 비밀키  (개발자는 위 바이트값을 수정해서 사용해야 함)
```

● **config.asp**

키파일 경로를 config 페이지에 명시한다.

```
<%
    KEY_PATH = "c:₩lib₩key.dat"
    ☞ (개발자 수정 항목) key.dat 파일을 저장한 로컬위치를 KEY_PATH로 수정해야함
%>
```

● SeedEncryption.asp

SEED-CBC 암호화를 수행하는 SeedEncryption 페이지의 ASP 소스이다. 암호화 수행 명령어(오브젝트.Encrypt)로 평문을 입력받아 암호화를 수행한다. DB 적용 시에는 Response.Write 명령어 대신 암호화된 결과값을 DB에 저장하도록 명령어를 수정하여 사용한다.

```
<%@ codepage="65001" language="VBScript" %>
<!--#include file='./config.asp'-->
<%
    ' 키파일(key.dat) 위치정보 include
    ' 유니코드로 파라미터가 전송되므로 값을 코드페이지를 유니코드로 설정
    ' 65001 : 유니코드 (UTF-8)
    sPlainText = Trim( Request("iPlainText") ) ☞ 웹페이지에서 평문을 가져오는 명령어
    set oSeed = Server.CreateObject( "seed.CBC" ) ☞ SeedCBC.dll에 정의된 오브젝트 생성
    returnMsg = oSeed.LoadConfig(KEY_PATH)
    ☞ 키와 초기값이 들어있는 key.dat의 경로를 확인하는 명령어(리턴값이 OK일 경우
정상이며, 나머진 오류 메시지 리턴됨)
    if StrComp("OK", returnMsg)= 0 then
            Response.Write( oSeed.Encrypt(sPlainText) )
            ☞ 암호화 수행 및 웹페이지에 암호문을 출력하는 명령어
            ☞ DB 적용 시 : oSeed.Encrypt(sPlainText)의 결과를 DB에 저장하는 명령어로 수정
            ☞ ex) DBconn.Execute( "INSERT INTO 저장할 테이블(필드명) values
( '" & oSeed.Encrypt(sPlainText) & "' )" )
    else
Response.Write( returnMsg )
    end if
    set oSeed = nothing ☞ 메모리 반환
%>
```

● SeedDecryption.asp

SEED-CBC 복호화를 수행하는 SeedDecryption 페이지의 ASP 소스이다. 복호화 수행 명령어(오브젝트.Decrypt)로 암호문을 입력받아 복호화를 수행한다. DB 적용 시에는 Request 명령어 대신 암호문을 DB에서 불러오는 명령어로 수정하여 사용한다.

```
<%@ codepage="65001" language="VBScript" %>
<!--#include file='./config.asp'-->
<%
    ' 키파일(key.dat) 위치정보 include
    ' 유니코드로 파라미터가 전송되므로 값을 코드페이지를 유니코드로 설정
    ' 65001 : 유니코드 (UTF-8)
    sCipherText = Trim( Request("oCipherText") )
    ☞ 웹페이지에서 암호문을 가져오는 명령어
    ☞ DB 적용 시 : Trim( Request("oCipherText") ) 대신 DB에서 암호문을 호출하는 명령어로
수정
    ☞ ex) Set rs = DBconn.Execute( "SELECT 암호문 필드 FROM 해당테이블 [WHERE
조건문]" )sCipherText = rs( "암호문 필드" )
    set oSeed = Server.CreateObject( "seed.CBC" )
    ☞ SEEDCBC.dll에 정의된 오브젝트 생성
    returnMsg = oSeed.LoadConfig(KEY_PATH)
    ☞ 키와 초기값이 들어있는 key.dat의 경로를 확인하는 명령어 (리턴값이 OK일 경우
정상이며, 나머진 오류 메시지 리턴됨)
    if StrComp("OK", returnMsg)= 0 then
            Response.Write( oSeed.Decrypt(sCipherText))
            ☞ 복호화 수행 및 웹페이지에 평문을 출력하는 명령어
    else
Response.Write( returnMsg )
    end if
    set oSeed = nothing ☞ 메모리 반환
%>
```

4.2.4 JSP 구현 예

다음은 JSP에서 SEED-CBC를 이용해 입력받은 개인정보를 암호화하기 위한 방법이다.

● SeedCBC.class의 등록

- JSP에서 불러올 class 파일들이 위치하는 폴더에 "KISA" 폴더를 생성하고, SeedCBC.class를 저장

```
c:₩webapps₩ROOT₩WEB-INF₩classes ☞ class 파일들이 위치하는 폴더
c:₩webapps₩ROOT₩WEB-INF₩classes₩KISA
☞ KISA 폴더 생성 후 SeedCBC.class 파일을 저장
```

● **키파일(key.dat)**

- SEED-CBC 암·복호화에 사용할 128비트 초기값(IV)과 128비트 비밀키값(MK)이 저장된 파일
- 키파일은 외부에 노출되면 안 되므로 로컬시스템의 안전한 위치에 저장해야 함
- 개발자는 반드시 IV, MK의 바이트값을 수정해서 사용해야 함

```
V[16] = {0x26,0x8D,0x66,0xA7,0x35,0xA8,0x1A,0x81,0x6F,0xBA,0xD9,0xFA,0x36,0x16,0x25,0x01}
☞ CBC모드를 위함 128비트 초기화값 (개발자는 위 바이트값을 수정해서 사용해야 함)
MK[16] = {0x88,0xE3,0x4F,0x8F,0x08,0x17,0x79,0xF1,0xE9,0xF3,0x94,0x37,0x0A,0xD4,0x05,0x89}
☞ 128비트 비밀키 (개발자는 위 바이트값을 수정해서 사용해야 함)
```

● **config.jsp**

키파일 경로를 config 페이지에 명시한다.

```
<%
    String KEY_PATH = "c:/lib/key.dat";
    ☞ (개발자 수정 항목) key.dat 파일을 저장한 로컬위치를 KEY_PATH로 수정해야함
%>
```

● **SeedEncryption.jsp**

SEED-CBC 암호화를 수행하는 SeedEncryption 페이지의 JSP 소스이다. 오브젝트 생성 시 비밀키와 초기값을 입력받으며, 암호화 수행 명령어(오브젝트.Encryption)로 평문을 입력받아 암호화를 수행한다. DB 적용 시에는 out.print 명령어 대신 암호화된 결과값을 DB에 저장하도록 명령어를 수정하여 사용한다.

```
<%@page language="java" contentType="text/html;charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@page import="sun.misc.BASE64Encoder"%>
<%@page import="java.io.*"%>
<%@page import="KISA.SeedCBC"%>
<%@include file="config.jsp"%>
<%
    SeedCBC s = new SeedCBC(); ☞ SeedCBC.class에 정의된 오브젝트 생성
    String retMsg = s.LoadConfig(KEY_PATH);
    ☞ 키와 초기값이 들어있는 key.dat의 경로를 확인하는 명령어 (리턴값이 OK일 경우
정상이며, 나머진 오류 메시지 리턴됨)
    if (retMsg.equals("OK") == false) {
        out.println(retMsg);
    } else {
    String sPlainText = request.getParameter( "iPlainText" );
    ☞ 웹페이지에서 평문을 가져오는 명령어
    byte[] bCipherText = s.Encryption( sPlainText.getBytes() ); ☞ 암호화 수행
    BASE64Encoder Base64Encoder = new BASE64Encoder();
    ☞ Base64인코더 오브젝트 생성
    out.print( Base64Encoder.encode(bCipherText) );
    ☞ 웹페이지에 암호문을 출력하는 명령어
    ☞ DB 적용 시 : Base64Encoder.encode(bCipherText)의 결과를 DB에 저장하는
명령어로 수정
    ☞ ex) DBconn.Execute( "INSERT INTO 저장할 테이블(필드명) values
( '" & Base64Encoder.encode(bCipherText) & "' )" )
    }
%>
```

- SeedDecryption.jsp

  SEED-CBC 복호화를 수행하는 SeedDecryption 페이지의 JSP 소스이다. 오브젝트 생성 시 비밀키와 초기값을 입력받으며, 복호화 수행 명령어(오브젝트.Decryption)로 암호문을 입력받아 복호화를 수행한다. DB 적용 시에는 request.getParameter 명령어 대신 암호문을 DB에서 불러오는 명령어로 수정하여 사용한다.

```
<%@page language="java" contentType="text/html;charset=EUC-KR"
pageEncoding="EUC-KR"%>
<%@page import="sun.misc.BASE64Decoder"%>
<%@page import="java.io.*"%>
<%@page import="KISA.SeedCBC"%>
<%@include file="config.jsp"%>
<%
    String sCipherText = request.getParameter( "oCipherText" );
    ☞ 웹페이지에서 암호문을 가져오는 명령어
    ☞ DB 적용 시 : request.getParameter( "oCipherText" ) 대신 DB에서 암호문 호출
명령어로 수정
    ☞ ex) Set rs = DBconn.Execute( "SELECT 암호문 필드 FROM 해당테이블 [WHERE
조건문]" )sCipherText = rs( "암호문 필드" )
    SeedCBC s = new SeedCBC(); ☞ SeedCBC.class에 정의된 오브젝트 생성
    String retMsg = s.LoadConfig(KEY_PATH);
    ☞ 키와 초기값이 들어있는 key.dat의 경로를 확인하는 명령어 (리턴값이 OK일 경우
정상이며, 나머진 오류 메시지 리턴됨)
    if (retMsg.equals("OK") == false) {
        out.println(retMsg);
    } else {
        BASE64Decoder Base64Decoder = new BASE64Decoder();
        ☞ Base64디코더 오브젝트 생성
        byte[] bCipherText = Base64Decoder.decodeBuffer( sCipherText );
        ☞ Base64디코딩 수행
        byte[] bPlainText = s.Decryption( bCipherText ); ☞ 복호화 수행
        out.print( new String(bPlainText) );
        ☞ 웹페이지에 평문을 출력하는 명령어
    }
%>
```

4.3. 기타 사항

4.3.1 dll 및 class 파일의 구성

- **SEEDCBC.dll의 키파일(key.dat) 불러오는 인터페이스 함수**

STDMETHODIMP CCBC::LoadConfig(BSTR keyPath, BSTR *retMsg)
– 인터페이스 함수명 : LoadConfig – 입력 : [문자열] BSTR keyPath – 로컬의 키파일 (key.dat) 위치 – 출력 : [문자열] BSTR *retMsg – 키파일을 처리한 결과 (성공일 경우 "OK" 이며, 나머진 실패)

- **SEEDCBC.dll의 암호화 인터페이스 함수**

STDMETHODIMP CCBC::Encrypt(BSTR PlainText, BSTR *retCipherText)
– 인터페이스 함수명 : Encrypt – 입력 : [문자열] BSTR PlainText – 평문 – 출력 : [문자열] BSTR *retCipherText – 암호문 (문자열로 반환하기 위해 Base64로 인코딩됨)

- **SEEDCBC.dll의 복호화 인터페이스 함수**

STDMETHODIMP CCBC::Decrypt(BSTR CipherText, BSTR *retPlainText)
– 인터페이스 함수명 : Decrypt – 입력 : [문자열] BSTR CipherText – 암호문 (Base64로 인코딩된 문자열) – 출력 : [문자열] BSTR *retPlainText – 평문

- **SEEDCBC.class의 키파일(key.dat) 불러오는 인터페이스 함수**

public String LoadConfig(String keyPath)
– 인터페이스 함수명 : LoadConfig – 입력 : [문자열] String keyPath – 로컬의 키파일 (key.dat) 위치 – 출력 : [문자열] String retMsg – 키파일을 처리한 결과 (성공일 경우 "OK" 이며, 나머진 실패)

● SEEDCBC.class의 암호화 인터페이스 함수

public byte[] Encryption(byte[] bPlainText)
– 인터페이스 함수명 : Encryption – 입력 : [1BYTE 배열] byte[] bPlainText – 평문 – 출력 : [문자열] String – 암호문 (문자열로 반환하기 위해 Base64로 인코딩됨)

● SEEDCBC.class의 복호화 인터페이스 함수

public byte[] Decryption(byte[] bCipherText)
– 인터페이스 함수명 : Decryption – 입력 : [1BYTE 배열] byte[] bPlainText – 평문 – 출력 : [문자열] String – 평문

● sha256.dll의 해쉬 인터페이스 함수

STDMETHODIMP Csha256::hash(BSTR sMessage, BSTR *retVal)
– 인터페이스 함수명 : hash – 입력 :[문자열] BSTR sMessage – 메시지 – 출력 :[문자열] BSTR *retVal – 해쉬한 값 (문자열로 반환하기 위해 Base64로 인코딩됨)

● sha256.class의 해쉬 인터페이스 함수

public String hash(byte[] bpMessage)
– 인터페이스 함수명 : hash – 입력 :[1Byte 배열] byte[] bpMessage – 메시지 – 출력 :[문자열] String – 해쉬한 값 (문자열로 반환하기 위해 Base64로 인코딩됨)

4.3.2 예시를 위한 공통 파일

● style.css

```
BODY
{
    FONT-FAMILY:  "굴림", "Tahoma", "Verdana", "MS Sans Serif", "Courier New";
    margin-top: 0px; margin-left: 0px; margin-right: 0px;
    /* border : Blue dotted f7f7f7;*/
    page-break-after : auto;
    page-break-before : auto;
    scrollbar-3dlight-color:#424142;
    scrollbar-arrow-color:#848284;
    scrollbar-base-color:#DEDFDE;
    scrollbar-darkshadow-color:white;
    scrollbar-face-color:#DEDFDE;
    scrollbar-highlight-color:white;
    scrollbar-shadow-color:#424142;
}
INPUT
{
    BORDER-RIGHT: #C2C2C2 1pt solid;
    BORDER-TOP: #C2C2C2 1pt solid;
    FONT-SIZE: 9pt;
    BORDER-LEFT: #C2C2C2 1pt solid;
    COLOR: #4F4E4E;
    BORDER-BOTTOM: #C2C2C2 1pt solid;
    FONT-FAMILY: 굴림, Arial;
    background-color: #FFFFFF;
    PADDING-RIGHT: 0px;
    MARGIN-TOP: 1pt;
    PADDING-LEFT: 2px;
    PADDING-TOP: 2px;
}
TEXTAREA
```

```
{
    BORDER-RIGHT: #C2C2C2 1px solid;
     BORDER-TOP: #C2C2C2 1px solid;
     FONT-SIZE: 9pt;
     OVERFLOW: auto;
     BORDER-LEFT: #C2C2C2 1px solid;
     COLOR: #4F4E4E;
     LINE-HEIGHT: 17px;
     BORDER-BOTTOM: #C2C2C2 1px solid;
     FONT-FAMILY: 굴림, Arial;
     BACKGROUND-COLOR: #FFFFFF;
}
TABLE
{
    BORDER : 0px solid #000000;
    TABLE-LAYOUT : fixed;
}
TD
{
    font-size: 12px;
    color: #3E3E3E;
    LINE-HEIGHT: 17px;
    TEXT-ALIGN: center;
}
```

● ajax.js

```
function GetHttpRequest( URL, Param )
{
    var xmlhttp = null;
    // FF일 경우 window.XMLHttpRequest 객체가 존재한다.
    if( window.XMLHttpRequest )
        // FF 로 객체선언
        xmlhttp = new XMLHttpRequest();
    else
        // IE 경우 객체선언
        xmlhttp = new ActiveXObject( "Microsoft.XMLHTTP" );
    // 값을 가져 왔을경우 호출할 메소드를 바로 선언
    xmlhttp.onreadystatechange = function()
    {
        // readyState 가 4 고 status 가 200 일 경우 올바로 가져옴
        if( xmlhttp.readyState == 4 && xmlhttp.status == 200 )
            // responseText 에 값을 저장
            var responseText = xmlhttp.responseText;
    }
    // POST 모드로 HTTP 요청을 전송함
    xmlhttp.open( "POST", URL, false );
    xmlhttp.setRequestHeader( "Content-Type", "application/x-www-form-urlencoded" );
    xmlhttp.send( Param );
    responseText = xmlhttp.responseText;
    // 가져온 xmlhttp 객체의 responseText 값을 반환
    return responseText;
}
String.prototype.trim = function()
{
     return this.replace( /(^₩s*)|(₩s*$)/gi, "" );
}
```

Korea Internet & Security Agency

부록. 암호관련 법 · 규정 현황

부록 1. 암호관련 법 · 규정 현황

개인정보 등 보호가 필요한 정보에 대하여 암호화 조치를 명시하고 있는 관련 법 · 규정은 다음과 같다. 업체는 자신이 제공하는 서비스에 따라 적용이 필요한 법을 구분하고, 해당 법에 명시된 요구사항을 적용하여야 한다.

1. 정보통신망 이용촉진 및 정보보호 등에 관한 법률

정보통신망 이용촉진 및 정보보호등에 관한 법률(이하 정보통신망법)은 정보통신망의 이용을 촉진하고 정보통신서비스를 이용하는 자의 개인 정보를 보호함과 아울러 정보통신망을 건전하고 안전하게 이용할 수 있는 환경을 조성하여 국민생활의 향상과 공공복리의 증진에 이바지함을 목적으로 한다.

정보통신망법 제28조(개인정보의 보호조치)에서는 개인정보의 기술적·관리적 보호조치의 하나로서 암호화기술 등을 이용한 보호조치를 규정하고 있으며, 정보통신망법 시행령 제15조(개인정보의 보호조치)에서는 비밀번호 및 바이오정보의 일방향 암호화 저장, 주민등록번호 및 금융정보를 암호화하여 저장할 것을 규정하고 있다.

정보통신망법 시행령

제15조(개인정보의 보호조치) ① 법 제28조제1항제1호에 따라 정보통신서비스 제공자등은 개인정보의 안전한 취급을 위하여 다음 각 호의 내용을 포함하는 내부관리계획을 수립 · 시행하여야 한다.

1. 개인정보 관리책임자의 지정 등 개인정보보호 조직의 구성 · 운영에 관한 사항
2. 개인정보취급자의 교육에 관한 사항
3. 제2항부터 제5항까지의 규정에 따른 보호조치를 이행하기 위하여 필요한 세부 사항

② 법 제28조제1항제2호에 따라 정보통신서비스 제공자등은 개인정보에 대한 불법적인

접근을 차단하기 위하여 다음 각 호의 조치를 하여야 한다.

1. 개인정보를 처리할 수 있도록 체계적으로 구성한 데이터베이스시스템(이하 "개인정보처리시스템"이라 한다)에 대한 접근권한의 부여 · 변경 · 말소 등에 관한 기준의 수립 · 시행
2. 개인정보처리시스템에 대한 불법적인 접근을 차단하기 위한 침입차단시스템 및 침입탐지시스템의 설치 · 운영
3. 비밀번호의 생성 방법 및 변경 주기 등의 기준 설정과 운영
4. 그 밖에 개인정보에 대한 접근통제를 위하여 필요한 조치

③ 법 제28조제1항제3호에 따라 정보통신서비스 제공자등은 접속기록의 위조 · 변조 방지를 위하여 다음 각 호의 조치를 하여야 한다.

1. 개인정보취급자가 개인정보처리시스템에 접속하여 개인정보를 처리한 경우 접속일시, 처리내역 등의 저장 및 이의 확인 · 감독
2. 개인정보처리시스템에 대한 접속기록을 별도 저장장치에 백업 보관

④ 법 제28조제1항제4호에 따라 정보통신서비스 제공자등은 개인정보가 안전하게 저장 · 전송될 수 있도록 다음 각 호의 보안조치를 하여야 한다.

1. 비밀번호 및 바이오정보(지문, 홍채, 음성, 필적 등 개인을 식별할 수 있는 신체적 또는 행동적 특징에 관한 정보를 말한다)의 일방향 암호화 저장
2. 주민등록번호 및 계좌정보 등 금융정보의 암호화 저장
3. 정보통신망을 통하여 이용자의 개인정보 및 인증정보를 송신 · 수신하는 경우 보안서버 구축 등의 조치
4. 그 밖에 암호화 기술을 이용한 보안조치

⑤ 법 제28조제1항제5호에 따라 정보통신서비스 제공자등은 개인정보처리시스템 및 개인정보취급자가 개인정보 처리에 이용하는 정보기기에 컴퓨터바이러스, 스파이웨어 등 악성프로그램의 침투 여부를 항시 점검 · 치료할 수 있도록 백신소프트웨어를 설치하여야 하며, 이를 주기적으로 갱신 · 점검하여야 한다.

⑥ 방송통신위원회는 제1항부터 제5항까지의 규정에 따른 사항과 법 제28조제1항제6호에 따른 그 밖에 개인정보의 안전성 확보를 위하여 필요한 보호조치의 구체적인 기준을 정하여 고시하여야 한다.

또한 정보통신망법 시행령 제15조제6항에 따라 방송통신위원회는 2009년 4월 개인정보의 기술적 · 관리적 보호조치 기준을 고시하였는데, 이 기준의 제6조(개인정보의 암호화)에 따르면 비밀번호와 바이오정보는 일방향 암호화하고, 주민등록번호, 신용카드번호 및 계좌번호는 암호화하여 저장하도록 규정하고 있다.

개인정보의 기술적 · 관리적 보호조치 기준

제6조(개인정보의 암호화) ①정보통신서비스 제공자등은 비밀번호 및 바이오정보는 복호화 되지 아니하도록 일방향 암호화하여 저장한다.

② 정보통신서비스 제공자등은 주민등록번호, 신용카드번호 및 계좌번호에 대해서는 안전한 암호알고리듬으로 암호화하여 저장한다.

③ 정보통신서비스 제공자등은 정보통신망을 통해 이용자의 개인정보 및 인증정보를 송 · 수신할 때에는 안전한 보안서버 구축 등의 조치를 통해 이를 암호화해야 한다. 보안서버는 다음 각 호 중 하나의 기능을 갖추어야 한다.

1. 웹 서버에 SSL(Secure Socket Layer) 인증서를 설치하여 전송하는 정보를 암호화하여 송 · 수신하는 기능
2. 웹 서버에 암호화 응용프로그램을 설치하여 전송하는 정보를 암호화하여 송 · 수신하는 기능

④ 정보통신서비스 제공자등은 이용자의 개인정보를 개인용 컴퓨터에 저장할 때에는 이를 암호화해야 한다.

2. 개인정보보호법

'11년 3월 29일 제정 · 공포되어 '11년 9월 30일 시행된 이 법은 개인정보의 수집 · 유출 · 오용 · 남용으로부터 사생활의 비밀 등을 보호함으로써 국민의 권리와 이익을 증진하고, 나아가 개인의 존엄과 가치를 구현하기 위하여 개인정보 처리에 관한 사항을 규정함을 목적으로 한다.

개인정보보호법 제24조제3항에서는 개인을 고유하게 구별하기 위하여 부여된 식별정보(고유식별정보)에 대하여 대통령령으로 정하는 바에 따라 암호화 등의 조치를 취하도록 하고 있으며, 시행령 제19조에서는 고유식별정보의 범위를 주민등록번호, 여권번호, 운전면허번호, 외국인등록번호로 정의하고, 제20조에서는 고유식별정보의 안전성 확보조치에 관하여 제30조(개인정보 안전성 확보조치)를 준용하도록 하고 있다.

개인정보보호법

제24조(고유식별정보의 처리 제한) ①개인정보처리자는 다음 각 호의 경우를 제외하고는 법령에 따라 개인을 고유하게 구별하기 위하여 부여된 식별정보로서 대통령령으로 정하는 정보(이하 “고유식별정보” 라 한다)를 처리할 수 없다.

1. 정보주체에게 제15조제2항 각 호 또는 제17조제2항 각 호의 사항을 알리고 다른 개인정보의 처리에 대한 동의와 별도로 동의를 받은 경우
2. 법령에서 구체적으로 고유식별정보의 처리를 요구하거나 허용하는 경우

② 대통령령으로 정하는 기준에 해당하는 개인정보처리자는 정보주체가 인터넷 홈페이지를 통하여 회원으로 가입할 경우 주민등록번호를 사용하지 아니하고도 회원으로 가입할 수 있는 방법을 제공하여야 한다.

③ 개인정보처리가자 제1항 각 호에 따라 고유식별정보를 처리하는 경우에는 그 고유식별정보가 분실 · 안전성 확보에 필요한 조치를 하여야 한다.

④ 행정안전부장관은 제2항에 따른 방법의 제공을 지우언하기 위하여 관계 법령의 정비, 계획의 수립, 필요한 시설 및 시스템의 구축 등 제반 조치를 마련할 수 있다.

또한, 개인정보 보호법 제29조(개인정보의 안전성 확보조치)에서는 개인정보의 안전성 확보에 필요한 기술적·관리적·물리적 조치를 하도록 하고 있고, 개인정보 보호법 시행령 제30조(개인정보의 안전성 확보조치)에서는 개인정보를 안전하게 저장·전송할 수 있는 암호화 기술 및 이에 상응하는 조치를 할 것을 규정하고 있다.

개인정보 보호법 시행령

제30조(개인정보의 안전성 확보 조치) ① 개인정보처리자는 법 제29조에 따라 다음 각 호의 안전성 확보 조치를 하여야 한다.

1. 개인정보의 안전한 처리를 위한 내부 관리계획의 수립 · 시행
2. 개인정보에 대한 접근 통제 및 접근 권한의 제한 조치
3. **개인정보를 안전하게 저장 · 전송할 수 있는 암호화 기술의 적용 또는 이에 상응하는 조치**
4. 개인정보 침해사고 발생에 대응하기 위한 접속기록의 보관 및 위조 · 변조 방지를 위한 조치
5. 개인정보에 대한 보안프로그램의 설치 및 갱신

6. 개인정보의 안전한 보관을 위한 보관시설의 마련 또는 잠금장치의 설치 등 물리적 조치

② 행정안전부장관은 개인정보처리자가 제1항에 따른 안전성 확보 조치를 하도록 시스템을 구축하는 등 필요한 지원을 할 수 있다.

③ 제1항에 따른 안전성 확보 조치에 관한 세부 기준은 행정안전부장관이 정하여 고시한다.

개인정보 보호법 제24조제3항 및 제29조와 시행령 제21조 및 제30조에 따른 개인정보처리자의 개인정보 처리의 안전성을 확보하기 위하여 행정안전부는 「개인정보의 안전성 확보조치 기준(행안부 고시 제2011-제43호)」을 고시하였는데, 이 고시 제7조(개인정보의 암호화)에서는 암호화가 필요한 개인정보를 명시하고 암호화 방법을 제시하고 있다.

개인정보의 안전성 확보조치 기준

제7조(개인정보의 암호화) ① 영 제21조 및 영 제30조제1항제3호에 따라 **암호화하여야 하는 개인정보는 고유식별정보, 비밀번호 및 바이오정보를 말한다.**

② 개인정보처리자는 제1항에 따른 **개인정보를 정보통신망을 통하여 송·수신하거나 보조저장매체 등을 통하여 전달하는 경우에는 이를 암호화하여야 한다.**

③ **개인정보처리자는 비밀번호 및 바이오정보는 암호화하여 저장하여야 한다. 단 비밀번호를 저장하는 경우에는 복호화되지 아니하도록 일방향 암호화하여 저장하여야 한다.**

④ **개인정보처리자는 인터넷 구간 및 인터넷 구간과 내부망의 중간 지점(DMZ : Demilitarized Zone)에 고유식별정보를 저장하는 경우에는 이를 암호화하여야 한다.**

⑤ 개인정보처리자가 내부망에 고유식별정보를 저장하는 경우에는 다음 각 호의 기준에 따라 암호화의 적용여부 및 적용범위를 정하여 시행할 수 있다.

1. 법 제33조에 따른 개인정보 영향평가의 대상이 되는 공공기관의 경우에는 해당 개인정보 영향평가의 결과
2. 위험도 분석에 따른 결과

⑥ 개인정보처리자는 제1항에 따른 개인정보를 암호화하는 경우 안전한 암호알고리즘으로 암호화하여 저장하여야 한다.

⑦ 개인정보처리자는 제3항, 제4항 및 제5항에 따른 개인정보 저장시 암호화를 적용하는 경우, 이 기준 시행일로부터 3개월 이내에 다음 각 호의 사항을 포함하는 암호화 계

획을 수립하고, 2012년 12월 31일까지 암호화를 적용하여야 한다. 단 인터넷 구간 및 인터넷 구간과 내부망의 중간 지점(DMZ : Demilitarized Zone)에 고유식별정보를 저장하는 경우 위험도 분석과 관계없이 암호화를 적용하여야 한다.

1. 개인정보의 저장 현황분석
2. 개인정보의 저장에 따른 위험도 분석절차(또는 영향평가 절차) 및 방법
3. 암호화 추진 일정 등

⑧ 개인정보처리자는 업무용 컴퓨터에 고유식별정보를 저장하여 관리하는 경우 상용 암호화 소프트웨어 또는 안전한 암호화 알고리즘을 사용하여 암호화한 후 저장하여야 한다.

3. 위치정보의 보호 및 이용 등에 관한 법률

이 법은 위치정보의 유출 · 오용 및 남용으로부터 사생활의 비밀 등을 보호하고 위치정보의 안전한 이용환경을 조성하여 위치정보의 이용을 활성화함으로써 국민생활의 향상과 공공복리의 증진에 이바지함을 목적으로 한다. 이 법에서는 위치정보의 보호를 위해 암호기술의 사용을 명시하고 있다.

이 법에서는 제16조 위치정보의 보호조치에서 위치정보의 유출, 변조, 훼손등을 방지하기 위한 암호화 소프트웨어의 활용 등 기술적 조치를 명시하고 있다.

위치정보의 보호 및 이용 등에 관한 법률

제16조(위치정보의 보호조치 등) **①위치정보사업자등은 위치정보의 누출, 변조, 훼손 등을 방지하기 위하여 위치정보의 취급 · 관리 지침을 제정하거나 접근 권한자를 지정하는 등의 관리적 조치와 방화벽의 설치나 암호화 소프트웨어의 활용 등의 기술적 조치를 하여야 한다. 이 경우 관리적 조치와 기술적 조치의 구체적 내용은 대통령령으로 정한다.**

② 위치정보사업자등은 위치정보 수집 · 이용 · 제공사실 확인 자료를 위치정보시스템에 자동으로 기록되고 보존되도록 하여야 한다.

③ 방송통신위원회는 위치정보를 보호하고 오용 · 남용을 방지하기 위하여 소속 공무원으

로 하여금 제1항의 규정에 의한 기술적 · 관리적 조치의 내용과 제2항의 규정에 의한 기록의 보존실태를 대통령령이 정하는 바에 의하여 점검하게 할 수 있다.
④ 제3항의 규정에 의하여 보존실태를 점검하는 공무원은 그 권한을 표시하는 증표를 지니고 이를 관계인에게 내보여야 한다.

위치정보의 보호조치 등에 관한 관리적 · 기술적 조치의 구체적 내용은 위치정보의 보호 및 이용 등에 관한 법률 시행령에서 정의하고 있는데, 시행령 제20조제2항에서 위치정보 시스템에의 권한 없는 접근을 차단하기 위한 암호화 · 방화벽 설치 등의 조치를 명시하고 있다.

부 록 2. 안내서 연혁

버전	제 · 개정일	제 · 개정내역
v1.00	2011년 11월	• “암호기술 구현 안내서” 로 제정

안내서 작성 공헌자

본 안내서의 제정 및 발간을 위해 아래와 같이 여러분들이 공헌을 하였습니다.

구분	성명	소속사
제안	연구개발팀	한국인터넷진흥원
초안 제출	연구개발팀	한국인터넷진흥원
검토	원유재	한국인터넷진흥원
	노명선	한국인터넷진흥원
	정현철	한국인터넷진흥원
	전인경	한국인터넷진흥원
	이향진	한국인터넷진흥원
	이환진정	한국인터넷진흥원
	책표준화분과	한국암호포럼
	암호기술분과	한국암호포럼
	안전성평가분과	한국암호포럼
	조태희	NHN
	조양현	다음커뮤니케이션즈
	김종덕	SK커뮤니케이션즈
	민병찬	KT
	이상훈	SKT
	양도영	롯데홈쇼핑
	박규선	롯데홈쇼핑
	김동현	이베이
	유다혜	이베이
	민원기	CJ오쇼핑
	진승옥	엠게임
	노윤재	한국EMC
	이현수	(주)안철수연구소
	정상곤	소프트포럼
	손병록	위즈베라

암호기술 구현 안내서

2011년 11월 인쇄
2011년 11월 발행

발행처 : 방송통신위원회 · 한국인터넷진흥원

서울특별시 종로구 세종대로 178
방송통신위원회
Tel: (02) 750-1114

서울특별시 송파구 중대로 109번지
대동빌딩 한국인터넷진흥원
Tel: (02) 405-5118

인쇄처 : 한올
Tel: (02) 2279-8494

(비매품)