화이트박스 암호 및 응용 기술 동향 분석

Analysis on Trends for White-Box Cryptography and Its Application Technology

김신효 (S.H. Kim) 지식정보보호연구팀 책임연구원
이윤경 (Y.K. Lee) 지식정보보호연구팀 선임연구원
정병호 (B.H. Chung) 지식정보보호연구팀 팀장

목 차

화이트박스 암호 기술이 소프트웨어 콘텐츠 저작권 보호(불법유출방지)를 위한 핵심 툴킷으로 등장하고 있다. 암호 알고리듬이 동작하는 단말과 단말의 사용자는 믿을 수 있다는 기존의 생각에서 최근에는 암호 알고리듬이 동작하는 단말은 오픈 플랫폼이 되어감에 따라 사용자가 모르는 사이 단말웨어가 설치되어 동작할 수도 있고, 사용자가 악의적인 목적으로 암호 알고리듬을 분석하고자 할 수도 있다는 인식이 퍼져가고 있다. 따라서 암호 알고리듬이 안전하게 동작하고, 암호 키의 불법유출을 막기 위한 하나의 방안으로 화이트박스 암호 기술이 등장하였다. 이 기술은 기존의 보안 하드웨어를 이용한 기술과는 달리 소프트웨어만으로 암호 알고리듬의 중간 연산값 및 암호화 키를 보호할 수 있다는 장점 외에도 다양한 장점을 보유하고 있으며, 암호 알고리듬의 화이트박스 구현 방법에 관한 연구가 활발히 진행되고 있다. 본 고에서는 화이트박스 암호 기술의 탄생 배경 및 화이트박스 암호 기술의 개념, 그리고 이를 이용한 응용 기술에 관하여 기술하고자 한다.

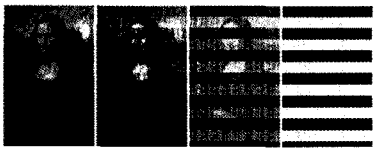
I. 서론

지금까지 개발되어 사용되고 있는 암호 알고리듬들은 통신하는 양쪽이 서로 믿을 수 있다는 가정 하에 통신 양단간 주고받은 암호화된 메시지는 제 3자가 해독할 수 없도록 설계되어 왔다. 그러나 실제 통신 환경에서는 암호화 통신에 참여한 사용자가 공격자가 될 수도 있고, 사용자가 사용하는 단말에 심어진 악성프로그램이 공격자가 될 수도 있다[1]. 이에 대한 해결책으로 TPM, 스마트카드 등의 대안이 제시되었으나, 하드웨어 사용에 대한 비용증가 및 설치의 어려움, 내부 결함 발생시 업데이트 혹은 패치의 어려움 등의 문제가 있다. 그러나 화이트박스 암호 기술은 소프트웨어만으로 암호 키를 안전하게 보관할 수 있고, 신뢰할 수 없는 단말에서 암호화 알고리듬이 실행되더라도 암호 키가 드러나지 않도록 할 수 있는 기술로서, 각종 암호 알고리듬의 화이트박스 구현에 관한 연구가 활발히 진행되고 있다.

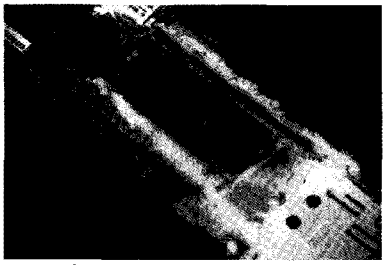
본 고에서는 화이트박스 암호 기술의 정의 및 화이트박스 기법을 적용한 AES 암호 알고리듬의 구조, 그리고 그 응용 기술에 관하여 기술하고자 한다.

II. 화이트박스 암호 기술

2008년 USENIX 보안 심포지엄에서는 "Cold boot attack"을 통해서 보안 하드웨어 TPM 칩이 장착된 노트북에 대해서도 암호 키 정보가 누출될 수 있음을 발표하여 큰 반향을 일으킨 바 있다[2]. 이 공격은 메모리에 로드된 정보는 전원 공급이 되지 않은 상태에서도 일정시간 동안 삭제되지 않으며(그림 1a), 온도를 더 낮추면 수 분 이상 해당 정보가 유지된다는 사실에 기반한 공격이다. 컴퓨터의 전원을 끄고 장착되어 있는 메모리를 저온 상태로 만들어서(그림 1b), 해당 메모리를 타 플랫폼에 장착하여 메모리에서 아직 지워지지 않는 암호 키 정보를 분석해서 알아내는 방법이다. 위 사례에서 고비용의 보안 하드웨어도 일시적인 저장소인 메모리 공격에 의해 쉽게 무력화 될 수 있음을 알 수 있다.



(a) 시간(15초, 30초, 60초, 5분)에 따른 메모리 내의 영상 유실 정도



(b) 메모리 정보를 유지하는 방법(저온 상태)

(그림 1) Cold Boot Attack 방법

최근에는 고비용에 보안 위협 대응이 유연하지 못한 보안 하드웨어의 장착보다는 소프트웨어적으로 보안 기능을 수행하도록 하고 공격 위협 인지시, 보안 패치로 문제를 해결하려는 움직임도 있다. 하지만 소프트웨어는 복제가 용이하다는 특성상 공격자가 해당 소프트웨어를 디버거, 에뮬레이터 등의 소프트웨어 분석 툴로 직접 실행하여 동작 과정이나 중요 정보를 알아내거나, 알고리듬 및 정보(프로그램의 실행 순서, 파라미터 등)를 변조하려는 공격에는 취약할 수 밖에 없다. 그러나 이러한 상황에서도 암호화 키 혹은 중요 비밀 정보를 공격자로부터 안전하게 지킬 수 있는 방법에 대한 연구가 진행중인데, 대표적인 기술로 화이트박스 암호 기술을 꼽을 수 있다.

1. 암호 공격 기술

암호 키를 알아내기 위한 암호 알고리듬에 대한

공격은 공격자가 제어할 수 있는 정보의 종류와 양에 따라서 블랙박스 공격, 그레이박스 공격, 화이트박스 공격으로 분류할 수 있다.

블랙박스 공격은 전통적인 공격 모델로서, 공격자는 암호화 연산이 일어나는 장치의 내부를 전혀 알 수 없기 때문에 블랙박스 공격이라고 한다. 즉, 공격자는 입력과 출력 데이터만을 알 수 있고, 연산이 일어나는 중간 정보를 알 수 없기 때문에 보안 강도는 암호 알고리듬의 강도에 의존한다. 주로 사용되는 블랙박스 공격에는 CPA와 CCA가 있다[1]. CPA는 특정 평문에 대응하는 암호문을 알고 있는 상태의 공격을 의미하고, CCA는 특정 암호문에 대응되는 평문을 알고 있는 상태의 공격을 의미한다.

그레이박스 공격은 공격자가 보안 하드웨어에 대해서 실행 시간, 전력 소비량, 전자파 특성 등을 모니터링하여 암호 키를 알아내거나 보안 하드웨어에 물리적 힘을 가하여(과도한 전류를 흘리거나, 과도한 자장에 노출하는 등) 암호화 모듈이 중간 데이터를 노출하도록 하는 부채널 공격을 의미한다.

화이트박스 공격은 공격자에게 가장 많은 능력을 부여하는 공격 모델로서, 앞서 기술한 블랙박스 및 그레이박스 공격 유형을 포함하고, 소프트웨어의 실행 과정을 모두 볼 수 있으며, 제어할 수 있는 상태의 강력한 공격 방법이다. 즉, 공격자가 메모리에 로드된 값들을 볼 수 있고, 디버거 등의 툴을 이용하여 프로그램의 수정 및 프로그램 실행 과정을 모니터링 할 수 있는 등의 역공학 분석(reverse-engineering)을 포함한다.

이들 공격 모델의 특성을 <표 1>에 정리하여 기술하였다.

특히 소프트웨어로 동작되는 암호 알고리듬에 대해서는 암호 키와 암호문의 입력과 복호화된 평문의 출력 외에도 암호 키의 중간 계산과정과 메모리 정보

〈표 1〉 암호 알고리듬에 대한 공격 유형

유형	공격자의 특징
블랙박스 공격	· 인지 정보: 알고리듬 정보 · 비인지 정보: 실행(즉 오퍼레이션) 과정 · 모니터링 범위: 입력, 출력
그레이박스 공격	· 인지 정보: 알고리듬 정보, 부채널 정보 (실행시간, 전력소비량, 전자파 특성 등) · 비인지 정보: 실행 과정 · 모니터링 범위: 입력, 출력
화이트박스 공격	· 인지 정보: 거의 모든 정보 · 비인지 정보: 거의 없음(실행 과정 및 메모리의 상태까지 알 수 있음) · 모니터링 범위: 입력, 출력, 중간 계산값

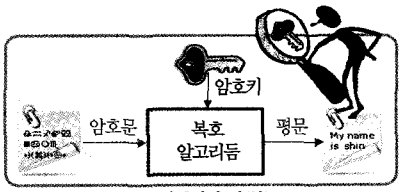
까지 공격자에 의해 모니터링 가능하므로, 암호 키를 유추하는 것이 매우 용이해질 것이다. 결국 입력 정보인 암호 키를 알고리듬 내부에 쉽게 유출할 수 없는 형태로 안전하게 숨길 수 있다면 화이트박스로 구동되는 암호 알고리듬을 해커가 모니터링 하더라도 암호 키 유추는 매우 힘들어질 것이고 이러한 과정을 실현한 것이 화이트박스 암호 기술이다. 다음절에서 이러한 화이트박스 암호 기술에 대해서 자세히 다룬다.

2. 화이트박스 암호 기술

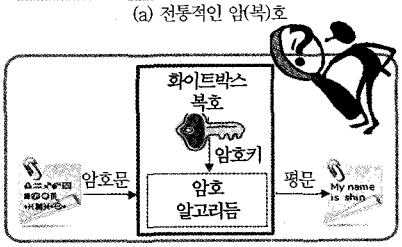
화이트박스 암호의 중심이 되는 암호 동작 과정은 기존 암호 메커니즘과 동일하지만, 이러한 암호 동작을 화이트박스 공격에도 안전하게 만들기 위한 구현 방법은 다양하다. 본 절에서는 기존에 논문으로 발표되어 알려진 AES 화이트박스 암호 구현을 중심으로 기본원리와 동작 메커니즘을 다룬다. 또한 향후 이러한 다양한 구현에 있어 해결되어야 하는 기술적 이슈를 언급하고자 한다.

가. 화이트박스 암호 원리

전통적인 암호 메커니즘은 암호 키가 블랙박스 장치(믿을 수 있는 단말)에서 안전하게 유지 관리된다



(a) 전통적인 암(복)호



(b) 화이트박스 암(복)호

(그림 2) 전통적 암호 vs. 화이트박스 암호

는 가정 하에 동작되는 반면, 화이트박스 암호 메커니즘은 암호 키가 소프트웨어로 구현된 암호 알고리듬 속에 섞여 있어서(obfuscation) 공격자가 암호 키를 쉽게 볼 수 없다는 가정 하에서 동작됨을 (그림 2)에 도시하였다.

즉, 화이트박스 암호는 알고리듬을 큰 룩업테이블로 만들고 그 안에 암호 키를 소프트웨어로 구현된 암호 알고리듬과 뒤섞인 상태로 숨겨둠으로써 내부의 동작을 분석하더라도 암호 키를 쉽게 유추하지 못하도록 하는 기법이다. 암호 알고리듬을 하나의 큰 룩업 테이블로 만들면 암호 키를 숨기는 것이 용이하지만 테이블 크기가 지나치게 커져서 비현실적이므로, 테이블을 암호학적인 기법으로 적절히 분리하되 암호화 연산의 중간값이 노출되지 않도록 디코딩과 인코딩 과정을 수행하도록 하면 된다. 기본적인 화이트박스 암호의 기본 원리는 (그림 3)에 도시한 바와 같다. 인코딩 과정(M_i)과 디코딩 과정(M_i^{-1})이 별도의 테이블에서 계산되므로 중간값이 노출되지 않으면서도 결국은 인코딩과 디코딩이 상쇄되면서 원래의 암호화 동작(X_i)만 수행하는 결과와 동일하게 된다. 암호학적인 안전성 문제로 외부인코딩(G)과 디코딩(F^{-1}) 과정이 추가되기 때문에 동일한 암호 키를 사용하더

$$F^{-1} \circ M_1^{-1} \circ M_1 \circ X_1 \circ M_2 \circ M_2^{-1} \circ M_3^{-1} \circ \cdots M_{2i-1} \circ X_i \circ M_{2i} \circ M_{2i}^{-1} \circ G$$

table table table table table

$$\Leftrightarrow F^{-1} \circ X_1 \circ X_2 \circ \cdots \circ X_i \circ G$$

(그림 3) 화이트박스 암호 기본 원리

라도 AES를 이용한 암복호 결과와 화이트박스로 구현된 AES를 이용한 암복호 결과는 차이가 있다.

결국 화이트박스 암호 구현은 테이블 전체가 암호키라 볼 수 있으며, 화이트박스 암호 테이블 구성이 충분한 임의성을 가지고 있으므로 실행시간 또는 전력량 분석 등을 이용한 그레이박스 공격에도 상당한 강인성을 제공할 수 있다.

나. 화이트박스 AES 동작 메커니즘

화이트박스 AES는 암호 키를 숨기기 위해 사용하는 메커니즘에 따라서 다양한 방법으로 구현될 수 있지만, 본 절에서는 화이트박스 암호의 개념을 처음으로 제안한 S. Chow의 논문[3]에서 제시된 화이트박스 AES(WB-AES) 구현 방법을 기술한다. S. Chow는 화이트박스 AES 뿐만 아니라 화이트박스 DES[4]의 구현 기법도 각각 다른 논문을 통해 발표하였다.

AES 알고리듬[5]의 라운드 연산과 WB-AES의 라운드 연산에는 약간의 차이가 있다. AES의 경우 라운드 연산은 SubBytes, ShiftRows, MixColumns, AddRoundKey의 반복으로 이루어져 있으나, WB-AES의 라운드 연산은 ShiftRows, AddRoundKey, SubBytes, MixColumns의 반복으로 이루어져 있다. 즉, WB-AES에서는, 최초의 key whitening을 위한 AddRoundKey 연산을 첫번째 라운드 내에서 수행하고, 첫번째 라운드의 AddRoundKey 연산은 다음 라운드 연산에서 수행하도록 함으로써, 매 라운드는 AddRoundKey 연산에서 시작해서 MixColumns 연산으로 끝난다. WB-AES에서 라운드 연산을 MixColumns로 끝나도록 한 이유는 WB-AES 구현시 하

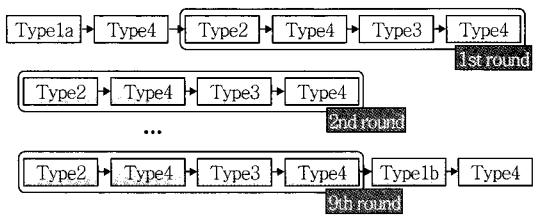
나의 큰 룩업 테이블이 아닌, 여러 개의 작은 룩업 테이블로 만드는 과정과 관계가 있다. ShiftRows는 1바이트 단위의 연산으로써 AddRoundKey 및 SubBytes와 순서가 바뀌어도 연산 결과는 동일하므로 구현의 편의를 위해서 매 라운드 연산의 맨 앞에서 수행한다. <표 2>에서는 128비트 입력 키 길이에 대한 AES 및 WB-AES의 연산 순서를 보여준다.

〈표 2〉 AES 및 WB-AES의 동작순서

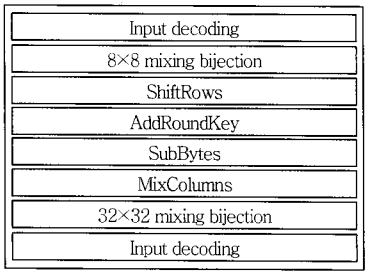
AES	WB-AES
1. AddRoundKey(K_0)	1. for i from 1 to 9:
2. for i from 1 to 9:	(a) ShiftRows;
(a) SubBytes;	(b) AddRoundKey(K_{i-1});
(b) ShiftRows;	(c) SubBytes;
(c) MixColumns;	(d) MixColumns;
(d) AddRoundKey(K_i);	2. ShiftRows;
3. SubBytes;	3. AddRoundKey(K_9)
4. ShiftRows;	4. SubBytes;
5. AddRoundKey(K_{10})	5. AddRoundKey(K_{10})

S. Chow[3]가 제안한 WB-AES는 Type1a, Type1b, Type2, Type3, Type4의 다섯 가지 테이블로 구성되어 있는데, 각 테이블의 입력 데이터와 출력 데이터는 각각 2개의 nibble(4비트)을 치환(permutation)하여 디코딩하고 인코딩하는 비선형 변환을 통해서 테이블 내부 연산이 쉽게 드러나지 않도록 하였다. 이들 5가지 테이블을 이용한 WB-AES의 연산 순서는 (그림 4)를 참고하자.

(그림 4)에서 볼 수 있듯이 Type1a, Type1b, Type2, Type3 테이블 연산 후에는 Type4 테이블 연산이 따라온다. 이는 Type1a, Type1b, Type2, Type3 내에서 수행했던 행렬 곱셈 결과들을 모아서 행렬 곱셈의 마무리를 위한 XOR 연산을 할 필요가 있는데, Type4 테이블에서 이러한 XOR 연산을 수행하기 때문에 Type4 테이블이 다른 테이블의 뒤에 따라오게 된다. Type2 테이블에서는 대부분의 AES 라운드 연산을 수행하는데, 상세 구조는 (그림 5)를 참고하자.



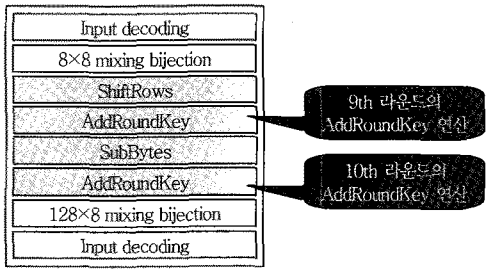
(그림 4) WB-AES 연산 순서



(그림 5) Type2 테이블의 구조

(그림 5)에서 알 수 있듯이, Type2 테이블에서는 입력 데이터의 디코딩, 출력 데이터의 인코딩 외에도 라운드 연산 전/후에 8×8 가역행렬을 곱해주는 8×8 mixing bijection 연산과 32×32 가역행렬을 곱해주는 32×32 mixing bijection 연산이 존재한다. 이들 행렬을 라운드 연산 전/후에 곱해줌으로써 라운드 연산의 중간 데이터 및 키를 공격자로부터 안전하게 숨길 수 있다.

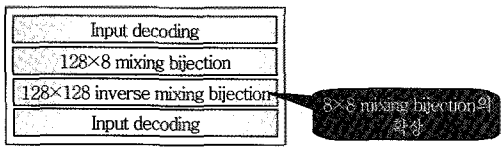
Type3 테이블에서는 Type2 테이블에서 곱해진 8×8 행렬(8×8 mixing bijection)과 32×32 행렬(32×32 mixing bijection)에 대한 역행렬을 곱해줌으로써 Type2, Type4, Type3, Type4 테이블 연산을 모두 수행하였을 때, AES의 라운드 연산만 남을 수 있도록 한다. Type1a 테이블과 Type1b 테이블은 WB-AES의 안전성을 높이기 위하여 128비트 입력 및 출력 데이터에 128×8 가역행렬을 곱해주는 연산을 수행한다. 그리고, 이들 행렬 곱으로 인해서 동일 입력 데이터와 동일한 키가 입력되더라도 WB-



(그림 6) Type1b 테이블의 구조

AES의 연산 결과와 AES의 연산 결과는 달라지게 된다. 또한 Type1b 테이블은 앞서 기술한 출력 데이터가 직접 드러나지 않도록 보호해주는 기능 외에도 AES의 마지막 라운드 연산을 함께 수행하는데, Type1b 테이블의 구조는 (그림 6)을 참고하자.

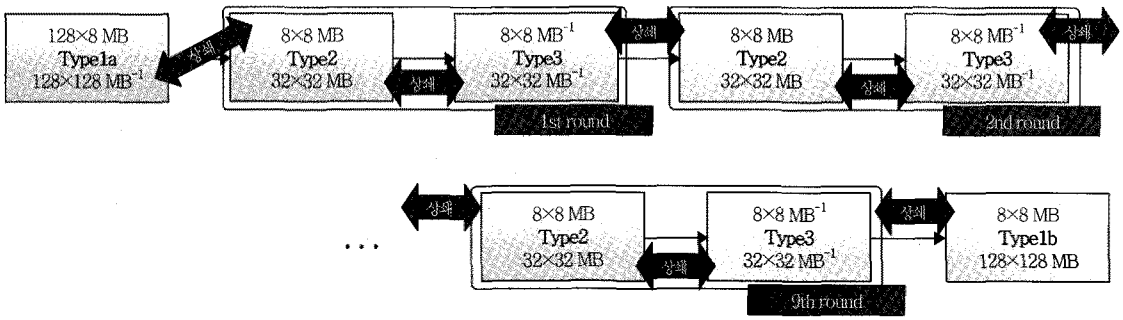
앞서 기술하였듯이, AES의 암호화 연산은 AddRoundKey 연산을 수행한 후 10회의 라운드 연산(128비트 입력 데이터에 대한 암호화 연산의 경우)을 수행하는데, WB-AES에서는 최초의 AddRoundKey 연산을 첫번째 라운드 연산을 수행하는 Type2 테이블 내에서 수행하고, 첫번째 라운드의 AddRoundKey 연산은 두번째 라운드 연산을 수행하는 Type2 테이블 내에서 수행하므로, 마지막 라운드 연산을 수행하는 Type1b 테이블에서는 9번째 라운드용 AddRoundKey 연산과 마지막 라운드용 AddRoundKey 연산을 함께 수행한다. 또한 Type1b 테이블의 8×8 mixing bijection 연산은 9번째 라운드 연산을 수행했던 테이블들 중 Type3 테이블에서 8×8 역행렬을 미리 곱해



(그림 7) Type1a 테이블의 구조

주고, Type1b 테이블에서 이의 역행렬인 8×8 행렬을 곱해주는 연산을 수행함으로써 서로 상쇄될 수 있도록 하였다. 앞서 기술하였듯이, Type3 테이블에서는 32×32 역행렬과 8×8 역행렬을 곱해주는 기능을 수행하는데, 32×32 역행렬은 동일 라운드의 Type2 테이블에서 곱해준 32×32 행렬에 대한 역행렬을 곱해주는 것이고, 8×8 역행렬은 다음 라운드의 Type2(마지막 라운드의 경우 Type1b) 테이블에서 곱해줄 8×8 행렬에 대한 역행렬을 곱해주는 것이다. 또한 첫번째 라운드 연산에서 Type2 테이블에서 곱해줬던 8×8 행렬에 대한 역행렬은 Type1a 테이블에서 미리 곱해주기 때문에 서로 상쇄되어 없어질 수 있다. (그림 7)에서 Type1a 테이블의 구조를 보여준다.

(그림 8)에서는 WB-AES에서 mixing bijection (행렬 곱셈) 과정을 통해서 곱해진 행렬이 어떤 행렬과 서로 역행렬 관계에 있고, 이들이 상쇄되는지를 그림으로 보여준다. (그림 8)에서 볼 수 있듯이 WB-AES 연산 중간에 mixing bijection 과정을 통해서 곱해진 행렬들은 모두 상쇄되지만, 평문 입력 데이터에 곱해진 행렬과, 암호문 출력 데이터에 곱해진 행렬은



(그림 8) WB-AES의 Mixing Bijection 연산

상쇄되지 않고 남아있게 되고, 이들 행렬에 대한 역행렬은 복호화 과정에서 곱해져서 상쇄되어 데이터의 암/복호화가 정상적으로 이루어진다. 다만 (그림 8)에는 설명의 편의를 위하여 Type4 테이블을 생략하였다.

3. 기술적 이슈

아직까지는 화이트박스 암호 알고리듬의 응용 보다는 암호 알고리듬(특히, 비밀키 암호)의 화이트박스 구현 방법에 대한 연구에 집중되고 있다. 따라서 실제 서비스에 화이트박스 암호 기술을 적용하기 위해서는 풀어야 할 숙제가 많다. 가장 대표적인 예로써 화이트박스 구현은 암호 키를 추출해 내고자 하는 공격에 대해서 안전하게 구현되어 있으나, 화이트박스 암호모듈 전체를 가져다 쓰는 공격에 대해서는 취약할 수 밖에 없다. 따라서 소프트웨어로 구현된 화이트박스 암호모듈을 다른 단말에서 불법으로 사용되지 못하도록 하는 노드-락킹(node-locking) 기술이 함께 구현될 필요가 있다. 또한 화이트박스로 구현된 암호모듈은 암/복호화 기능만을 수행하는 기존의 암호모듈에 비해서 암/복호화 속도도 느리고 많은 양의 메모리를 필요로 하는 등의 단점이 있다.

이 외에도 실제 응용에서 고려되어야 할 사항들이 있다. 예를 들면, 기존의 비밀키 암호를 이용한 DRM 시스템에서는 콘텐츠 식별자와 해당 콘텐츠를 암호화한 키 쌍을 저장하면 되지만, 콘텐츠 암호화에 화이트박스 암호 기술을 적용할 경우 화이트박스로 구현된 룩업 테이블 전체를 암호화 키로 볼 수 있기 때문에 서버에 저장되어야 할 정보의 양이 많아지게 되므로 이를 해결할 필요가 있다. 그러나 기존의 방법에 비해서 암호화 키가 그대로 드러나지 않기 때문에 키 자체에 대한 안전성은 높아졌다고 할 수 있을 것이다. 또 다른 예로써 사용자에게 전송할 데이터의 양이 기존의 암/복호 메커니즘에 비해서 많아졌고, 이를 해결할 필요가 있다. 즉, 기존의 방법에서는 콘텐츠를 암호화하는 데 사용했던 키만 안전하게 전송하면 되었지만, 화이트박스 암호 기술을 적용하게 되면 콘텐츠 암호화 키가 숨겨진 룩업 테이블 전체를 전송해야 한다는 문제가 있다. 그러나, 이 경우에도 여전히 화이트박스 암호를 이용하는 데 따르는 장점이 있다. 기존의 암호 알고리듬을 이용하게 되면 콘텐츠 암호화 키를 안전하게 전송하기 위한 별도의 키 전송 보안 메커니즘이 필요하지만, 화이트박스 암호 기술을 적용할 경우 공개키를 전송할 때처럼 키가 숨겨진 화이트박스 룩업 테이블을 별도의 보안 프로토콜 없이 전송이 가능하다는 큰 장점이 있다.

앞서 기술한 화이트박스 암호 기술의 한계 및 이에 따른 단점 외에도, 이 기술은 기존의 키를 숨기기 위한 하드웨어 기반의 솔루션에 비해서 소프트웨어만으로 암호화 키를 숨길 수 있기 때문에 많은 장점을 갖는다. 대표적인 장점으로 키 및 키를 보호하기 위한 모듈의 분배 및 설치가 쉽고, 비용 면에서 효율적이다. 또한 오류 발생시 소프트웨어 업데이트 혹은 패치 전송을 통해서 원격으로 수정이 가능하다는 점을 꼽을 수 있다.

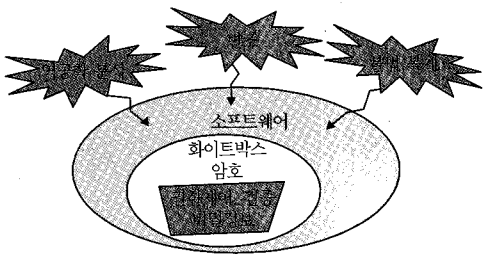
소프트웨어만으로 키를 안전하게 보관할 수 있다는 화이트박스 암호 기술의 특성을 이용하여 공개키 암호에도 화이트박스 암호 기술을 적용하여 구현하기도 하는데 Arxan[6], Cloakware[7] 등에서는 RSA 암호뿐만 아니라 ECC 암호까지 화이트박스 암호 기법으로 구현하여 솔루션을 판매하고 있음을 홍보하고 있지만, 논문 등에서는 관련 자료들이 공개된 바 없다. 또한 이들 회사에서는 AES, DES의 화이트박스 암호 솔루션을 제공하고 있으며, Syncrosoft[8]도 이와 같은 암호 솔루션을 제공한다.

Ⅲ. 화이트박스 암호 응용 기술

화이트박스 암호 기술은 암복호 연산 속도가 느리고 필요한 메모리 용량이 커질 수 있어서 모든 응용 분야에서 기존의 암호 기술을 대체하는 기술은 아니다. 하지만 일부 응용에 적용시 보안 하드웨어 없는 상태의 소프트웨어 보안 기술로서의 장점이 극대화 될 수 있다. 본 절에서는 이러한 화이트박스 암호 응용 기술에 대하여 논한다.

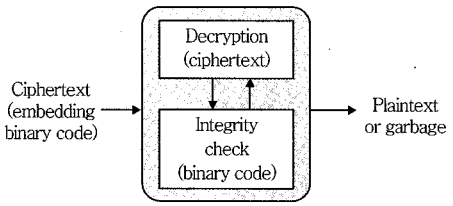
1. 소프트웨어 보호 응용 서비스

지금까지의 사용자 단말은 외부 침입자에 의한 공격 시도로부터 자신의 단말을 안전하게 지키기 위해서 HSM, 스마트카드 등의 보안 하드웨어를 활용하거나, 자신의 단말이 안전한지 확인하기 위해 바이러스 진단 및 치료를 수행한다. 즉, 사용자는 자신의 단말에 대하여 해킹을 시도하지 않는다는 가정 하에서 외부의 침입을 방지하기 위한 보안 위협 모델이었다. 하지만 신뢰할 수 없는 개방형 단말 플랫폼 환경에서는, 사용자가 잠재적인 공격자로서 단말에 설치되어 있는 소프트웨어에 대하여 역공학 분석, 비밀정보의 변조, 권한 제어/검증 기능 무력화 또는 불법 복제를 할 가능성이 있다. 내부의 비밀정보와 권한 제어 및 검증 기능 등의 소프트웨어 실행을 안전하게 지키기 위해서, 화이트박스 암호 기술은 매우 유용하며 이러한 개념은 (그림 9)와 같이 표현할 수 있을 것이다.



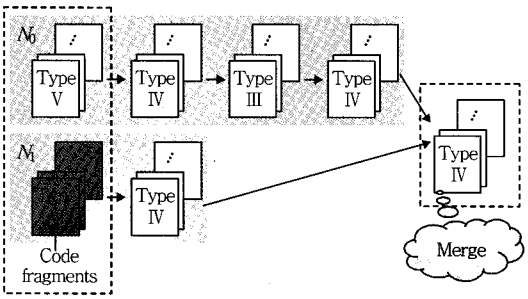
(그림 9) 화이트박스 암호를 이용한 소프트웨어 보호

소프트웨어 임의조작 방지에 화이트박스 암호를 적용[9]한 기본적인 아이디어는 화이트박스 암호에 숨겨진 소프트웨어(실행 바이너리코드)에 변조가 발생되면 화이트박스의 암호 키까지 변경되도록 화이트박스 암호 테이블을 구성함으로써 암호 연산 결과값이 의미 없는 값이 되도록 한다. 이러한 원리를 간단히 도시하면 (그림 10)과 같으며, 복호화를 진행하면서 바이너리 코드의 무결성을 확인한다는 의미에서 '이중 해석(dual interpretation)' 메커니즘으로 부르기도 한다.

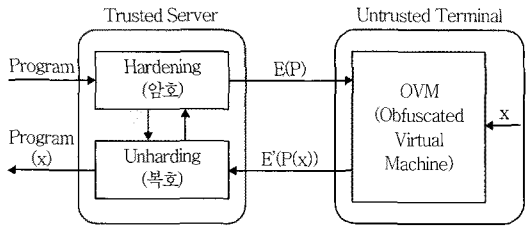


(그림 10) 화이트박스 암호 기반 소프트웨어 임의조작 방지 방법

이중 해석 메커니즘은 기존에 제안된 WB-AES [3]의 동작 전체를 N 네트워크라고 할 때, N 네트워크는 N_0 네트워크와 N_1 네트워크의 XOR 결과값과 동일하다는 특성을 이용하였다. 화이트박스 암호 테이블 구성 시에 무결성 보장을 원하는 실행 바이너리 코드 조각을 Type6 테이블로 구성하고 원래의 암호키 테이블인 Type2 테이블을 변형하여 Type5 테이블로 구성한다. 이와 같은 테이블 구성은 (그림 11)에 도시하였다.



(그림 11) 화이트박스 암호 이중해석을 위한 룩업테이블 구성



(그림 12) WBRPE 구조

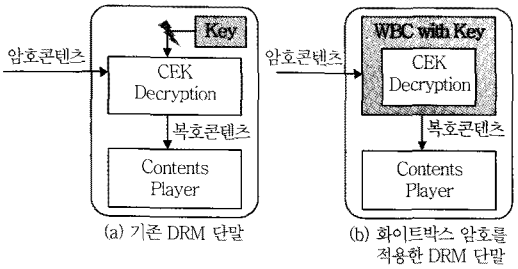
이 기법은 암호 키에 의한 암복호 과정 중에만 무결성을 제공하고, 룩업테이블에 포함되는 바이너리 코드의 크기에 제한이 있으므로, 이 부분에 대한 보완이 필요하다.

또 다른 방법으로는 WBRPE[10]가 제안되었으며, 개략적인 구조는 (그림 12)에 도시하였다. 이 기법은 신뢰할 수 없는 원격 단말은 신뢰 서버로부터 암호화된 E(P)을 수신하여 E'(P(x))를 계산하고, 이를 다시 신뢰 서버로 전송하여 P와 결과값 P(x)가 변조되지 않았음을 검증하도록 한다. 그러나 OVM을 화이트박스 암호를 이용하여 구성하는 자세한 기법에 대한 제시가 없고 이론적인 검증 수준이므로, 다른 보안 기법들로 보완되어야 할 것으로 판단된다.

2. 콘텐츠 보호 응용 서비스

가. DRM 서비스 시나리오

DRM은 디지털 콘텐츠에 대한 불법적인 사용을 차단하기 위해서 콘텐츠는 암호화하고, 콘텐츠 암호키(CEK) 등의 권한정보는 특정 단말 또는 사용자만 볼 수 있도록 별도로 암호화하여 전송한다. 하지만 아직까지는 표준화된 단일 DRM 솔루션도 없고 특정 벤더에 종속적인 보안 하드웨어를 채용할 수도 없어서, DRM 클라이언트에 권한정보를 해독할 수 있는 사용자의 비밀키를 내장하는 경우가 많다. 이 경우, 콘텐츠 제공자가 콘텐츠를 암호화해서 전송하더라도 사용자가 DRM 클라이언트 소프트웨어를 분석하면



(그림 13) 화이트박스 암호를 이용한 DRM 서비스

사용자 비밀키를 알아낼 수 있고, 불법적인 사용과 배포에 무방비 상태가 된다. 하지만 사용자 비밀키를 화이트박스 암호로 적절히 숨겨 놓는다면 소프트웨어 분석에 의해서도 쉽게 비밀키를 알아낼 수 없으며, 이러한 과정을 (그림 13)에 도시하였다.

나. 기타 응용 서비스

이외에도 화이트박스 암호는 포렌식 워터마킹과 소프트웨어를 특정 단말에서만 수행 가능하도록 하는 노드-락킹 서비스 등에도 응용 가능하다.

콘텐츠의 불법 배포자를 추적하는 용도로 콘텐츠 내에 사용자들을 모두 구별해 낼 수 있는 정보를 포렌식 워터마크로 삽입한다. 콘텐츠 제공자는 포렌식 마크를 구분하기 위해서 사용자를 식별하여 추적할 수 있는 유일한 정보가 필요하고 이 정보가 바로 화이트박스 암호에 내장된 암호 키 정보일 것이다. 화이트박스 암호를 활용한 노드-락킹은 하드웨어 식별자 정보를 암호 키와 같이 숨겨서 식별자 정보가 정확한 경우에만 정상적으로 수행하도록 하는 기법이며, 불법 사용자에게 하드웨어 식별자를 어떤 방법으로 숨길 수 있느냐는 별도로 해결하여야 하는 문제이다.

지금까지 화이트박스 암호 기반 응용 보안 기술을 살펴본 바와 같이, 화이트박스 암호는 기존의 암호 기술을 대체하는 기술이 아니라 타 보안 기술과 연계된 상태에서 소프트웨어 보안을 강화하는 하나의 대안으로써는 의미가 크다고 볼 수 있다.

Ⅳ. 결론

암호 기술에서는 안전성이 검증된 공개된 알고리듬을 사용하되 암호 키를 안전하게 관리함으로써 전체 시스템의 안전성을 보장한다. 이러한 암호 체계는 보안 하드웨어를 가정한 블랙박스 공격에 안전하며, 화이트박스로 동작하는 소프트웨어에 대한 보안에서는 기존 암호 체계의 안전성을 기반으로 하되, 화이트박스의 특성을 반영하는 형태로 기술의 변화가 필요할 것이다. 이러한 기술이 바로 화이트박스 암호이다. 본 고에서는 이러한 특징을 갖는 화이트박스 암호 기술의 등장 배경, 기술적인 동작 메커니즘 및 그 응용 분야에 대하여 간략히 살펴보았다.

본 고에서 설명한 바와 같이, 새롭게 등장한 화이트박스 암호 기술은 메모리, 수행 속도 등의 제약 등으로 일부 응용에 국한될 수는 있으나, 저비용의 원격 단말의 소프트웨어적인 보안 솔루션으로는 매우 유용할 것이다. 다만, 관련 기술에 대한 연구 및 개발이 아직까지는 미비한 상태이므로, 안전한 화이트박스 암호구현 기술 및 응용에 대한 활발한 연구 개발이 필요하다. 또한 화이트박스 테이블 구성 모델, 인코딩 및 디코딩의 안전성 검증, 성능 검증 기준 등에 대한 표준화도 필요할 것으로 판단된다.

● 용 어 해 설 ●

화이트박스 공격: 공격자는 암호 연산 과정과 중간값을 모두 알아낼 수 있어서 암호키 해석 및 유추가 가능함(대부분의 소프트웨어는 화이트박스로, 디버거 등으로 분석 공격 가능함)

화이트박스 암호(White-Box Cryptography, WBC): 암호키가 암복호 알고리듬 내부에 암호 기술로 숨겨져 있어서, 화이트박스인 소프트웨어 동작 과정을 분석하더라도 암호키 해석을 어렵게 하는 새로운 암호 기술

약어 정리

AES	Advanced Encryption Standard
CCA	Chosen-Ciphertext Attack
CEK	Contents Encryption Key
CPA	Chosen-Plaintext Attack
DES	Data Encryption Standard
DRM	Digital Right Management
ECC	Elliptic Curve Cryptography
HSM	Hardware Security Module
OVM	Obfuscated Virtual Machine
RSA	Rivest, Shamir, and Adleman
TPM	Trusted Platform Module
WBC	White-Box Cryptography
WBRPE	White-Box Remote Execution Program

참 고 문 헌

[1] M. Joye, "On White-Box Cryptography," *Proc. 1st Int'l Conf. Security of Information and Networks(SIN 07)*, Trafford Publishing, 2008, pp.7-12.

[2] J. Alex Halderman et al., "Lest We Remember: Cold-boot Attacks on Encryption Keys," *Communications of the ACM*, Vol.52, No.5, May 2009.

[3] S. Chow et al., "White-Box Cryptography and an AES Implementation," *Proc. 9th Ann. Workshop Selected Areas in Cryptography(SAC 02)*, LNCS 2595, Springer, 2002, pp.250-270.

[4] S. Chow et al., "White-Box DES Implementation for DRM Applications," *Proc. 2nd ACM Workshop Digital Rights Management*, LNCS 2696, Springer, 2002, pp.1-15.

[5] Joan Daemen and Vincent Rijmen, "The Design of Rijndael:AES-The Advanced Encryption Standard," Springer, 2001.

[6] Arxan, http://www.arxan.com

[7] Cloakware, http://www.cloakware.com

[8] Syncrosoft, http://www.syncrosoft.com

[9] W. Michiels and P. Gorissen, "Mechanism for Software Tamper Resistance: An Application of White-Box Cryptography," Proc. 7th ACM Workshop Digital Rights Management, ACM Press, 2007, pp.82-89.

[10] Amir Herzberg, Haya Shulman, Amitabh Saxena, and Bruno Crispo, "Towards a Theory of White-box Security," Cryptology ePrint Archive, Report 2008/087, 2008.