하드웨어 암호화 함수를 이용한 스크립트 응용 프로그램의 보안성 향상

(Improvement of Confidentiality for Script Applications using HW Cryptography Functions)

최 창 기 † 한 태 숙 ††
(Changki Choi) (Taisook Han)

요 약 오픈 플랫폼 환경은 외부 개발자가 응용 프로그램을 개발하여 내부 임베디드 시스템에서 실행되는 구조로 되어 있다. 스크립트 응용 프로그램 기반의 오픈 플랫폼 환경의 경우 스크립트 응용 프로그램은 누구나 내용을 확인할 수 있어 기밀성에 취약하다. 본 논문에서는 응용 프로그램의 코드 노출 및 무단복제를 방지하기 위하여, 하드웨어 암호화 함수를 통한 응용 프로그램의 암호화와 암호화에 사용된 키의 안전한 관리 솔루션을 제시한다. 이 솔루션을 통해 오픈 플랫폼 환경의 보안 신뢰도를 향상 시킴으로써 개발자는 다양한 응용프로그램을 개발하여 사업과 수익의 다변화를 이루고, 환경 제공자는 다양한 응용 프로그램을 도입할 수 있어 환경 경쟁력을 향상시킬 수 있다.

키워드 : 오픈 플랫폼, 스크립트 언어 응용 프로그램, 하드웨어 암호화, 임베디드 시스템, 보안, 기밀성, 복사방지

Abstract A major feature of open platform environment is that applications made by external developers can be executed on various internal embedded systems. Script applications of open platform are vulnerable in confidentiality because they can be exposed to an attacker easily. This paper suggests a way to encrypt script applications and manage the encryption key using HW cryptography on embedded system. This solution can protect source codes and unauthorized use of applications. With our solution for open platform environment to improve the reliability of the security, developers can provide various applications and expand their business model, and platform providers can provide various applications and improve their competitiveness.

Key words : Open Platform, Script Language Application, HW Cryptography, Embedded System, Security, confidentiality, Copy Protection

1. 서 론

현재 전 세계적인 스마트 열풍속에는 다양한 응용 프로그램을 특정 에코시스템에 제공하여 다양한 사용자의 만족도를 극대화 하고자 에코시스템 환경 제공자는 오픈 플랫폼을 통한 불특정 다수의 개발자 참여를 유도하고 있다. 이런 목적의 여러 개발환경 중 HTML, Java Script와 같은 Plain Text로 구성된 스크립트 응용 프로그램을 활용하는 환경도 있으며 이 환경은 스크립트의 취약한 기밀성에도 불구하고 개발자에게는 개발의 용이함을, 시스템 제공자에게는 저비용으로 쉬운 하위 호환성을 보장해 주는 장점이 있다.

스크립트 응용 프로그램 기반 오픈 플랫폼에서는 필연적으로 응용 프로그램의 코드 유출, 무단 복제 등의 보안성에 대한 문제가 부각되며 이 보안성을 얼마나 간편하고 안전하게 보장하는가에 따라 개발자, 사용자, 환경제공자 모두에게 높은 만족도를 줄 수 있다. 특히, 응용 프로그램 개발을 통해 수익을 창출하는 응용 프로그램 개발자에게 보안성은 개발자 개인 또는 개발자가 속

† 정 회 원 : KAIST 전산학과
hancck@gmail.com
†† 종신회원 : KAIST 전산학과 교수
han@cs.kaist.ac.kr
(Corresponding author임)
논문접수 : 2011년 11월 16일
심사완료 : 2012년 1월 26일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제18권 제4호(2012.4)

한 회사의 수익에 막대한 영향을 미칠 수 있다. 그리고 제조사와 같이 해당 환경을 제공하는 회사는 유통되는 응용 프로그램의 보안성을 얼마나 편하고 안전하게 제공하는가에 따라 해당 환경의 개발 및 유지 보수에 드는 비용을 절감할 수 있으며, 보안 신뢰도가 높을수록 다수의 개발자가 다양한 응용 프로그램을 제공할 수 있으므로 해당 환경 자체의 경쟁력이 높아질 수 있다.

기밀성 및 코드 유출을 방지하기 위해서는 응용 프로그램 개발 후 응용 프로그램을 암호화 하고 암호화된 응용 프로그램을 전달하여야 하며, 이를 최종 실행 단계에서 복호화 하여 사용하여야 한다. 하지만, 대칭키 암/복호화를 사용할 경우 암/복호화에 사용되는 동일한 키 값이 개발용 SDK와 임베디드 시스템에 동시 탑재되어 있어야 하므로, 어느 한쪽에서 해당 키가 유출될 경우 기밀성을 확보할 수 없다. 대칭키 유출을 회피하기 위해 비대칭키 암호화를 사용할 경우 유통되는 과정의 키는 유출에 대한 걱정이 없으나, 복호화를 위한 키는 임베디드 시스템에 내장되어야 한다[1]. 하지만 이 방식 또한 대칭키 방식과 같이 복호화를 위해 임베디드 시스템에 내장된 키의 유출을 막는 문제가 남아 있으며, 비대칭키 암호화 방식은 키 사이즈가 커서 복호화에 많은 시간이 걸린다는 단점이 있어 임베디드 시스템에서 응용 프로그램 전체를 복호화 하는데 적절하지 않다.

이러한 문제점을 해결하기 위해 본 논문에서는 PC SDK에서 개발된 응용 프로그램의 암호화와, 대칭키의 안전한 전달, 하드웨어 암호화 함수를 이용한 대칭키 유출 방지 및 응용 프로그램의 무단 복제 방지 시스템을 제안한다.

이 시스템에서는 응용 프로그램 전달시 네트워크 단말의 보안성 향상을 통한 전체 시스템의 보안성을 향상시키는 방법이 아닌, 응용 프로그램 개발을 위해 배포하는 SDK와 응용 프로그램의 복호화 및 실행을 처리하는 임베디드 시스템만의 암/복호화 솔루션을 제시한다. 이로써 응용 프로그램의 배포시에 인증된 네트워크 경로가 아닌 일반 e-mail 이나 이동식 저장 장치와 같이 안전하지 않은 경로를 통해 전달하더라도 해당 응용 프로그램의 기밀성을 보장할 수 있으며, 대칭키 및 비대칭키 암호 방식을 상황에 맞게 혼용 사용하여 복호화시 성능도 보장할 수 있다. 또한 제안된 시스템을 실제 오픈 플랫폼 환경에 적용하여 솔루션의 실효성을 검증한다.

개발 환경이 오픈 플랫폼의 한 형태인 제조사가 개발용 툴 킷(SDK), 응용 프로그램 서버, 임베디드 시스템을 관리하고, 외부 개발자가 응용 프로그램을 개발하여 제조사에게 전달하는 구조로 된 플랫폼에서 본 논문에서 제안한 솔루션을 적용하여 응용 프로그램의 보안성을 향상시켜 줄 수 있다. 보안성 향상은 개발자에게는 환경의 신뢰감을 주어 다양한 응용 프로그램을 제작할 수 있도록 하여 사업 다변화 기회를 제공하고, 환경 제공자(제조사)에게는 초기 개발 비용, 하위 호환성 유지를 위한 비용, 관리 및 운영 비용을 절감시켜 주는 장점이 있다.

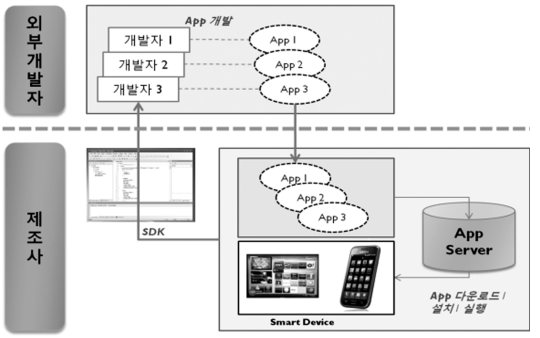

그림 1 스마트 Device 용 오픈 플랫폼 환경

임베디드 시스템을 생산하는 제조사는 그림 1과 같이 SDK(Software Development Kit)를 외부 개발자에게 제공한다. 외부 개발자는 SDK를 이용하여 제조사의 임베디드 시스템에서 동작하는 응용 프로그램(이하 App)을 제작하여 이를 제조사에 전달하게 되고 제조사는 이 App 을 인증하고 서버에 등록하며, 여러 임베디드 시스템에서는 이 App을 다운로드하여 설치 및 실행한다.

Script 언어 App을 기반으로 하는 환경에서는 Script의 취약한 기밀성으로 코드 노출, 무단 복제 등의 공격이 용이하다. 취약한 기밀성으로 인해 개발자에게는 다양한 App 제작을 제약하고 환경을 제공하는 제조사에게는 다양한 App 수급을 어렵게 한다.

따라서 App을 제작하는 개발자는 자신의 App이 외부 공격자에 의해 코드 및 데이터가 유출되지 않아야 하고, App 자체가 무단 복제되어 구매하지 않은 사용자의 기기에서는 동작되지 않기를 요구한다. SDK, 임베디드 시스템, App 인수 및 배포를 관리하는 제조사 입장에서는 모든 개발자를 잠재적인 공격자로 가정하고 App의 취약한 기밀성을 보완해 주는 솔루션을 제공하여야 한다. 이 때, 제공되는 솔루션은 임베디드 시스템에서 메모리 제약, 성능 과부하 및 구현 용이성 측면에서 무리가 없는 기술이어야 한다.

이를 위해 대칭키, 비대칭키 암호화 함수를 혼용 사용하여 App과 App 암호화에 사용된 대칭키를 보호하고, 기기에서는 하드웨어 암호화 함수를 이용하여 비대칭 개인키 관리 및 설치된 App을 기기별로 재암호화 한 후 관리하여 기밀성과 무단 복제 가능성을 차단하는 솔루션을 제공하여 보안성이 높고, 성능 부하가 적은 시스템을 제시하고자 한다.

이를 위해 본 연구에서는 2장에서는 시스템의 상세 요구사양 분석을 통해 본 연구에서 해결하고자 하는 내용을 정리하고, 3장에서는 하드웨어 암호화 함수를 통한 보안성 향상 시스템을 제시하고, 4장에서는 이론적인 기술 및 실제 사용되는 기존 기술에 대해 분석하고 5장에서는 구현 및 실험 결과를 소개하고, 6장에서는 결론과 향후 계획을 설명한다.

2. 시스템 상세 요구사양 분석

보안에는 크게 기밀성, 무결성, 인증성, 부인방지 네 가지 측면이 있다[2]. 이중 Script App을 기반으로 하는 오픈 플랫폼 환경에서는 코드 노출에 대한 불안감으로 인해 기밀성 측면의 요구사양이 가장 높다. 또한, App 판매를 통해 수익을 창출하는 개발자 입장에서는 App의 무단 복제 방지와 관련된 인증성 측면의 요구사양이 높다. 이 두 가지 측면의 보안 요구사양을 기준으로 PC용 SDK와 임베디드 기기의 특징을 고려하여 제안하는 솔루션이 만족해야 하는 상세 요구사양을 분석하면 아래와 같다.

첫째, 기밀성을 위해 가장 편하게 접근할 수 있는 방법은 대칭키 암호화 방식이다. 제조사가 SDK에 대칭키 암호화 시스템을 제공하여 App 을 암호화 할 경우 임의의 개발자는 SDK 자체에서 대칭키 값을 쉽게 추출할 수 있다. 또한 임베디드 시스템에 기록된 대칭키 값도 난이도의 차이만 있을 뿐 얼마든지 추출할 수 있다고 가정하여야 하므로 단순한 대칭키 암호화 시스템은 현실적으로 기밀성 문제를 해소한다고 볼 수 없다. 이러한 문제를 회피하고자 비대칭키 암호화 시스템을 사용하여 App을 암/복호화할 경우 큰 크기의 비대칭키 값과 복잡한 로직에 따른 성능 과부하로[3] 임베디드 시스템에서 App을 복호화 하는데 많은 시간이 소요되어 제조사에게 현실적인 솔루션이 될 수 없다. 더불어 비대칭키 암호화 방식으로 SDK에서 공개키로 App을 암호화하고 임베디드 시스템에서 개인키로 App을 복호화 할 경우 개인키 값은 임베디드 시스템의 펌웨어 어딘가에 기록되어야 하고, 기록할 때 어떠한 Obfuscation 방식을 도입한다 하더라도 공격자에 의해 유출될 가능성이 있어 완벽한 보안 솔루션이 될 수 없다.

둘째, 무단 복제 방지를 위해서는 구매한 사용자 또는 기기를 인증하여 App 실행여부를 승인 또는 부인해 주면 된다. 하지만 사용자 또는 기기를 인증하기 위해서는 외부에 있는 서버에 네트워크 등을 활용하여 인증여부를 문의하는 방식이 일반적으로 사용되며, 만약 네트워크가 유효하지 않은 상황이라면 해당 App의 실행여부를 판단할 수 없게 된다. 이렇게 인증 여부를 확인할 수 없는 예외 상황인 경우 항상 승인으로 판단하면 보안성이 취약해지고, 항상 부인할 경우는 사용성이 저하되는 문제점이 있다. 결국, 임베디드 기기 스스로 구매한 사용자 또는 기기임을 확인할 수 있는 방법이 있어야 한다.

셋째, 보안성 측면 외에 운영적 측면으로 다수의 개발자에게 배포되는 SDK와 SDK 배포수 이상의 다양한 App, 다양한 임베디드 기기에 서로 다른 키를 임의로 배포, 관리, 운용하는 솔루션은 기본적으로 키 배포 및 관리의 어려움 때문에 초기 개발비용과 기간이 많이 소요된다. 또한 암호화된 정보를 최종적으로 해석하는 임베디드 기기는 제한된 저장공간을 갖고 있지만 개발자 수 또는 App 수 만큼의 다른 키를 해석하기 위해서는 그와 대응되는 제한되지 않은 다수의 키 정보를 임베디드 기기에 미리 저장해 두어야 하므로 기술의 현실성이 떨어진다.

결국, 제조사 입장에서는 성능을 위해서는 대칭키 암호화 방식을 사용하여야 하고, 키 유출을 방지하기 위해서는 비대칭키 암호화 방식을 사용하여야 한다는 기본 방식을 결정할 수 있다. 여기에, SDK 뿐만 아니라 임베디드 시스템의 펌웨어에도 복호화에 필요한 키 값을 함부로 기록할 수 없다는 요구와, 임베디드 시스템 스스로 무단복제 방지를 위한 구매 사용자 인증 시스템이 있어야 한다는 요구사항이 추가된다. 더불어, 소수의 지정된 키를 이용하여 불특정 다수의 개발자 및 App을 대상으로 복호화를 처리할 수 있어야 한다는 요구도 추가된다.

이번 연구에서는 이러한 요구조건을 모두 만족하기 위해 키 값 추출이 불가능한 것으로 알려진 하드웨어 암호화 함수를 이용한 솔루션을 제안한다. 또한, 이번 연구 범위에서 하드웨어 암호화 모듈의 키 값 추출은 불가능한 것으로 가정하며, 보안성 측면중 기밀성과 무단 복제방지 측면을 제외한 인증성, 무결성, 부인방지 측면은 고려하지 않는다.

3. 하드웨어 암호화 함수 기반 보안성 향상 시스템

그림 2와 같이 일반적인 시스템에서 Script App은 코드 노출, App 무단 복제 가능성이 높다. 이 문제점을 해결하기 위해 그림 3과 같이 App을 암호화하여 전달하고, 최종 설치시 기기별로 서로 다른 키 값을 활용하여 다시 App 을 암호화하여 저장한다면 코드 노출, 무단 복제 문제점은 해결된다.

하지만, 이러한 솔루션으로 대칭키 방식을 그대로 사용할 경우 PC상의 SDK 뿐만 아니라 임베디드 시스템 펌웨어에 기록되어 있는 대칭키 키값도 유출될 수 있다. 비대칭키 방식만을 사용할 경우에는 비대칭키 방식의 복잡한 알고리즘과 큰 키 사이즈로 인한 복호화시 성능 문제가 이슈가 된다.

따라서, 임베디드 시스템의 성능을 보장하면서 보안

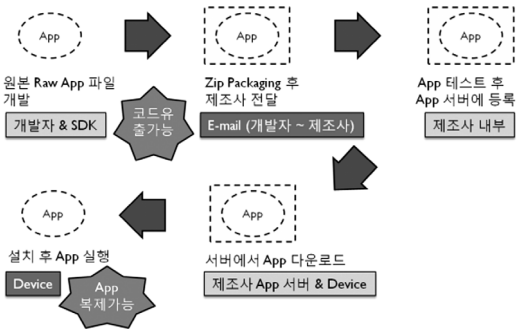

그림 2 Script App 전달 시스템의 보안 문제점

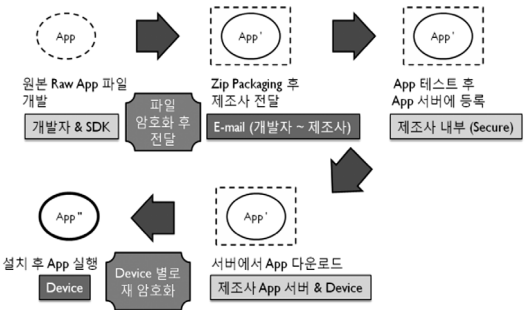

그림 3 Script App 전달 시스템의 보안 해결방향

문제를 해결하기 위해서는 대칭키/비대칭키 방식을 혼용하여야 한다. 즉, 성능을 위해서는 대칭키(s) 암호화를, 키값 노출 방지를 위해서는 비대칭키(a1, a2)를 사용하여야 한다. 이를 위해,

SDK에서는

1) 대칭키(s)를 생성하여 대칭키로 App을 암호화하고 (App′)
2) 비대칭 공개키(a2)로 대칭키(s)를 암호화 하여(s′)
3) App′과 s′을 제조사에 전달한다.

제조사에서는 해당 App을 App 서버에 등록하고

임베디드 시스템에서는

1) App′과 s′을 다운로드 하고
2) 비대칭 개인키(a1)로 s′을 복호화하여 s 값을 추출하고
3) s 값을 이용하여 App′을 복호화하여 원본 App을 추출한다.

위의 경우 한 쌍의 비대칭키 만을 이용하기 때문에 개발 및 유지보수가 용이하며, 크기가 큰 App은 대칭키로 암/복호화 하고, 대칭키 값은 비대칭키 방식으로 암/복호화 하여 대칭키 유출에 안전한 방식이다[4].

하지만, 아직도 비대칭 개인키(a1)를 임베디드 시스템의 펌웨어에 기록해 두어야 하므로 비대칭 개인키(a1)가 유출될 수 있다는 문제점이 남아 있으며, 무단 복제를 방지하기 위하여 임베디드 시스템 별로 서로 다른 대칭키 값으로 재 암호화를 해야한다는 요구를 만족하지 못한다.

따라서, 임베디드 시스템 펌웨어 코드내 키 유출 문제와 기기별 재 암호화 이슈를 해결하기 위해 키 값 노출 가능성이 없는 임베디드 시스템 메인 칩셋에서 제공하는 하드웨어 암호화 모듈을 이용하는 솔루션을 제안한다.

칩셋 제조사는 Device 별로 서로 다른 결과를 내는 Device Unique Confidential Symmetric암호화 함수(이하 Fi())와 동종 Model 별로 같은 결과를 내는 Model Common Confidential을 이용한 Symmetric암호화 함수(이하 Fm())을 제공한다. 이 두가지 키값은 HW 내에 내장되어 있고 칩셋 제조사별로 높은 수준으로 보호하고 있어 이번 연구에서는 이 값이 노출되지 않음을 가정한다.

이 두가지 하드웨어 암호화 함수(Fi(), Fm())를 이용하여 App 복호화에 필요한 비대칭 개인키(a1) 값을 모델 대칭키 암호화 함수(Fm)로 암호화하여 펌웨어에 기록하고, App의 무단 복제를 방지하기 위해 Fi()로 App을 암호화하여 기기별로 저장하고, 실시간 복호화하여 실행하는 시스템을 통해 스크립트 기반 App의 코드 노출 및 기기간 무단 복제를 방지할 수 있다.

임베디드 시스템 개발자는 개발 단계에서 Fm()을 이용하여 비대칭 개인키(a1)을 암호화하고(a1′) 이 값을 임베디드 시스템 펌웨어에 기록하여 둔다. 이 후,

임베디드 시스템에서는

1) App′과 s′을 다운로드 하고
+) 암호화된 비대칭 개인키(a1′)를 Fm()를 통해 복호화하여 a1을 추출하고
2) 비대칭 개인키(a1)로 s′을 복호화하여 s값을 추출하고
3) s 값을 이용하여 App′을 복호화하여 원본 App을 추출한다.
+) 원본 App은 Fi()를 통해 재 암호화하여 App″ 상태로 설치된다.
+) App″을 실행할 때는 다시 Fi()를 통해 실시간으로 App을 복호화하여 사용한다.

위처럼 키 값 유출 가능성이 없는 여러 키와 암호화 방식을 혼용하여 사용함으로써 임베디드 시스템에서 복호화 성능 이슈가 없고, App의 기밀성 및 무단 복제 가능성을 차단할 수 있는 솔루션을 제공할 수 있다.

4. 관련 연구

Plain Text를 암호화하여 Client Device에서 복호화하여 사용하는 시스템은 많은 연구가 이루어졌다. 단순하게 대칭키로 암호화하고 대칭키를 미리 공유하는 방

법은 대칭키 유출에 대한 보완책이 전혀 없어 J. C. Smith[5]는 대칭키 기반으로 Plain Text를 암호화 하고 대칭키를 공개키로 암호화하여 전달한 후 Client에서 대칭키를 개인키로 복호화하여 사용하는 대칭키와 비대칭키 암호화 시스템을 혼용 사용하는 시스템을 제안하였다. 하지만 개인키의 안전한 관리에 대한 기술이 전혀 거론되지 않아 n 쌍의 Key Pair가 필요한 시스템으로 예상할 수 있어 한 쌍의 Key Pair만 사용해야 하는 현재 시스템의 기본 요구사양을 만족할 수 없다. R. Perlman[6]는 개인키의 안전한 관리를 위한 시스템을 제안하였다. 서버에서 개인키를 관리하고 사용자를 인증한 후 안전한 경로를 통해 개인키를 배포하는 방식으로 네트워크가 유효하지 않은 상황에서는 사용자를 인증할 수 없어 현재 시스템에는 적합하지 않다. 즉, 기존 연구에서는 대칭키를 해석할 수 있는 개인키를 N개를 준비하고 각 Key Pair를 관리하거나, Key를 관리하는 서버를 두고 안전한 네트워크 경로를 통해 Key를 발급받는 등의 제약이 있다. 하지만 제안한 솔루션은 대칭키를 보호하는데 한쌍의 비대칭키만 있으면 되며, 네트워크가 유효하지 않아도 비대칭키를 사용할 수 있다.

이론적인 비교 외에 현재 오픈 플랫폼 환경으로 App을 배포하는 시스템을 비교하면 다음과 같다. 현재 오픈 플랫폼 환경을 운영하는 곳은 아이폰으로 대표되는 애플과 안드로이드 환경을 제공하는 구글이 있다. 기밀성, 무단 복제 방지 측면에서 제안한 솔루션과 두 회사의 방식을 비교하면 아래와 같다.

• 기밀성

표 1과 같이 애플, 구글 모두 컴파일 언어를 사용하여 App의 코드에 대한 중간 수준의 기밀성은 유지하나, Reverse Engineering 통해 코드 노출 가능성이 있어 개발자는 App 개발 시 App 내에 사업적으로 중요한 키 값이 필요할 경우 키 노출 방지를 위한 추가 작업이 필요하다. 그러나 제안한 솔루션은 기본적으로 취약한 기밀성의 Plain Text를 암호화 하여 reverse engineering이 불가능해지고, 결국 컴파일 언어보다 기밀성이 우수해져 App의 코드 내 키 값 노출 등의 우려를 해소할 수 있다.

• 무단복제 방지

애플[7], 구글[8] 모두 구매한 기기에서 파일을 꺼낼 수 없게 하거나(구글) 구매하지 않은 고객에게 복사되지 않도록 하는(애플) 등 파일의 이동을 제약하여 복사 방지를 시도하고 있다. 애플은 이와 함께 구매하지 않은 고객이 온라인 상태로 앱을 실행할 경우 앱 서버에서 구매한 기기인지 여부를 판별하여 원 구매자에게 알림 메시지를 전달하는 방법을 추가적으로 사용하고 있다. 이러한 방법은 구매하지 않은 사용자가 App을 사용할 수 있는 가능성이 남아 있다. 그러나, 제안한 솔루션은 기본적인 방식으로 앱을 꺼낼 수도 없고, 앱을 꺼내서 타 기기에 옮기더라도 해당 기기의 Fi()의 결과가 다르므로 타 기기에서는 앱이 실행되지 않아 근본적인 복사 방지 효과가 있다.

5. 구현 및 실험 결과

시판중인 Smart Device와 해당 기기의 PC용 SDK에 제안한 솔루션을 구현하여 실효성을 검증하였다.

Windows XP 상에서 동작하는 SDK는 Plain Text 상태로 package되었던 App을 전달하였다. 여기에 암호화 모듈을 추가하여 전달용 App을 암호화 할 수 있도록 하였다. 이 때 대칭키(s)를 암호화하는 공개키(a2)를 위한 비대칭키(a1, a2)는 사전에 1쌍을 발급하여 공개키(a2)는 미리 SDK 내에 저장하여 두고 모든 대칭키(s)는 이 공개키(a2)를 통해 암호화 한다.

개발에 사용한 대칭키 암호화는 AES128 함수를, 비대칭키 암호화는 RSA1024 함수를 사용하였으며 키 생성, 대칭키/비대칭키 암호화 함수는 openssl0.9.8i 버전을 사용하였다.

Smart Device Platform 에서 대칭키 및 비대칭키 암/복호화 함수를 구현하고, 구현된 암/복호화 함수를 사용할 수 있는 인터페이스를 추가하여 App Manager가

표 1 애플, 구글과 기밀성 비교

	애플	구글	현솔루션
언어	Object C	Java	Script (JavaScript, HTML)
기밀성	컴파일 통해 binary화 - 기밀성 중간	컴파일 통해 binary 화 - 기밀성 중간	암호화된 Text - 기밀성 높음
비고	reverse engineering 통한 코드 노출 가능	reverse engineering 통한 코드 노출 가능	암호화 통해 reverse engineering 불가

표 2 애플, 구글과 무단복제 방지 비교

	애플	구글	현솔루션
방식	- 복사통제 - 온라인검증	- 복사통제	- 기기별 암호화
비고	- 타 기기로 옮길 수 있고 네트워크를 끊으면 사용 가능	- 타 기기로 옮길 수 있으면 사용 가능	- 타 기기로 옮겨도 사용 불가

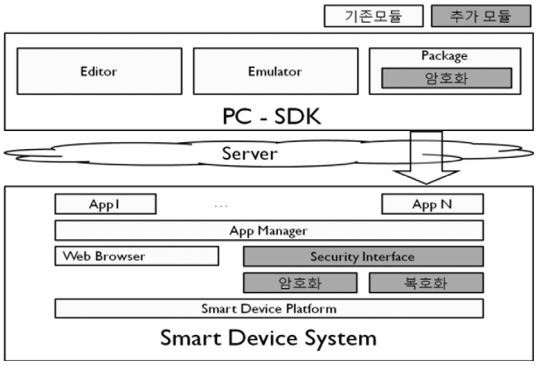

그림 4 PC용 SDK와 Smart Device System

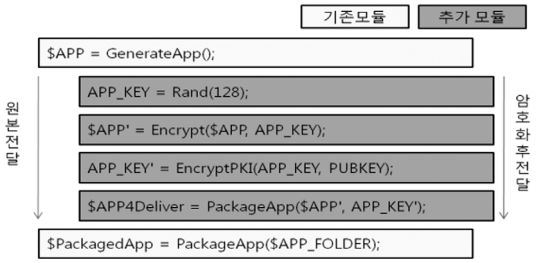

그림 5 PC용 SDK 내 추가 Pseudo Code

App을 설치하는 과정에서 비대칭키를 이용한 대칭키 추출, 대칭키를 이용한 원본 파일 획득, 최종 설치시 기기별로 서로 다른 결과를 내는 암호화 함수 Fi()를 이용한 저장과, App 실행 시 Web Browser의 file load 시 다시 Fi()를 이용한 복호화 로직을 추가하였다. Device 펌웨어에는 미리 발급한 비대칭키 중 개인키가 저장되어 있으며 이 개인키는 Model Common Confidential를 이용한 Fm()을 이용하여 암호화 되어 Device 펌웨어에 저장되어 있다.

Device 내 HW Crypto AES 함수는 Smart Device Platform에서 제공하는 HW AES 함수를 사용하였으며, SW 암호화 함수는 openssl0.9.8i에서 제공하는 것을 사용하였다.

그림 6과 같은 구조로 App의 암호화 및 암호화된 App의 실시간 복호화 통한 실행이 정상적으로 동작되는 것을 확인할 수 있었다.

이 구조에서 암/복호화에 사용되는 가장 큰 입력은 App 자체이다. App 자체는 500KB~15MB 정도의 크기를 갖고 있으며 평균 약 1.2MB 정도의 크기를 갖는다. 또한 이 App을 설치시 총 2번의 암호화 또는 복호화를 하여야 하는 구조이다. 이 때 사용할 암호화 함수를 결정하기 위해 HW AES 함수와 openssl을 이용한 SW AES/RSA 함수를 이용하여 Data Encoding 성능을 측정하였다.

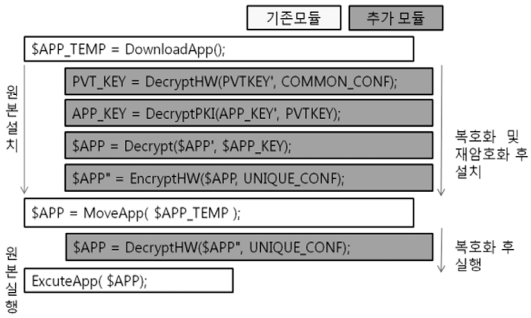

그림 6 Device Platform 내 추가 Pseudo Code

표 3 AES(HW/SW), RSA(SW) 성능 측정(단위:ms)

입력크기	AES128_HW	AES128_SW	RSA1024_SW PUB_Enc
0.1 MB	10	10	640
1.0 MB	80	90	7690
10 MB	770	860	66970
20 MB	1510	1850	136520
30 MB	2290	3140	196680

표 4 암호화 App 설치 시 시간측정(용량:MB, 시간:ms)

App 개수	총 용량	암호 미사용	암호 사용	추가 시간	추가 비율
1	1.3	13,230	14,540	1,310	10%
5	6.9	73,750	80,280	6,530	9%
10	13.4	151,430	167,880	16,450	11%

성능 측정 결과, 표 3과 같이 HW AES128 보다 SW AES128 버전은 평균 1.2배정도 느리며, RSA1024 SW 버전은 약 90배 정도 느리다. 따라서, App의 실시간 암/복호화에는 RSA 함수를 사용하는 것은 비현실적이며, HW와 SW AES128의 성능 차이는 그다지 크지 않으므로 구현 난이도와 시스템 의존 상황에 따라 HW/SW 함수 중에서 선택하여 사용하면 될 것으로 판단된다.

가장 성능이 우수한 HW AES를 기준으로 App 설치 시 2번의 암/복호화(복호화 1번, 암호화 1번)를 추가하여 설치 시 소요 시간을 측정한 결과는 표 4와 같다.

암호화 솔루션을 적용한 경우 평균 10% 정도의 시간이 더 소요되는 것을 확인할 수 있다. 높은 기밀성을 보장하고 무단 복제를 방지할 수 있는 솔루션으로서 10%의 시간 오버헤드는 감수할 수 있는 것으로 판단된다.

6. 결론 및 향후 연구

본 논문에서는 응용 프로그램의 코드 노출 및 무단복제를 방지하기 위하여 하드웨어 암호화 함수를 통한 응용 프로그램의 암호화와 암호화에 사용된 키의 안전한 관리 솔루션을 제시하였다. App의 대칭키 암호화를 통

해 Plain Text 기반의 취약한 기밀성을 극복하였고, 기기별 재 암호화를 통해 무단 복제 방지를 방지하였다. 또한 암호키 노출 방지를 위해 비대칭키 암호방식을 혼용 사용하였으며, 펌웨어에 기록되어야 하는 키는 하드웨어 암호화 함수로 재 암호화하여 외부 공격자의 공격에 대해 안전한 시스템을 제공한다. 또한, 기기에서는 HW 대칭키 암호화 함수를 이용하여 App을 암/복호화하여 과도한 오버헤드가 발생하지 않고, 전달되는 키 값의 보호를 위해 한 개의 비대칭키를 사용함으로서 비대칭키 발급과 관리가 간소화 되었다.

이 결과, 컴파일 언어보다 취약했던 기밀성은 오히려 강화되었다. 이는 App 개발자가 App에서 노출되지 않아야 하는 키 값을 App내에서 사용해야 할 경우 컴파일 언어에서는 reverse engineering 통한 키 노출 가능성이 있어 네트워크 서버를 두고 안전한 채널로 접속해서 키 값을 받아 오는 등 복잡한 절차를 거쳐야 하지만, 제안한 솔루션에서는 코드 내에 키 값을 적어 두어도 키 노출 가능성이 없으므로 개발자는 이러한 보안 문제를 고려하지 않고 App 개발에만 집중할 수 있는 장점을 제공한다.

이번 연구에서는 기밀성과 무단 복제 방지에 초점을 맞추어 연구를 하였다. 하지만, 현재 시스템이 Device Unique Confidential과 Model Common Confidential의 추출불가를 가정하고 있으므로 이 값이 추출되었을 때 두 값에 의해 보호되고 있던 키 값(a2, Private Key)을 갱신할 수 있는 시스템에 대한 추가 연구가 필요하며, App내 의도적으로 포함된 악성코드에 대한 탐지 및 최소의 키만을 이용한 앱의 무결성 검증과 App 을 전달한 개발자를 인증하는 인증 시스템에 대해서 추가 연구가 필요하다.

참 고 문 헌

[1] R. D. Pietro, and L. V. Mancini, "Security and privacy issues of handheld and wearable wireless devices," *Comm. ACM*, vol.46, no.9, pp.74-79, Sep. 2003.

[2] W. Stallings, Cryptography and Network Security: Principles and Practices, Pearson Education, 2004.

[3] Y. H. Jung, and T. Y. Choe, "Encryption Efficiency Analysis using Embedded Board," *Proc. of the KIISE Korea Computer Congress 2006*, vol.33, no.1(C), pp.289-391, Jun. 2006. (in Korean)

[4] D. Wang, and X. Li, "Improved method to increase AES system speed," *ICEMI 2009. 9th International Conference*, pp.3-49-3-52, Aug. 2009.

[5] J. C. Smith, J. Bandini, "Method and system for dynamic server document encryption," http://www.patentstorm.us/patents/6061448.html, United States Patent, no.6061448(US), May. 2000.

[6] R. Perlman, and C. Kaufman, "Secure password-based protocol for downloading a private key," *Proc. ISOC Network and Distributed System Security Symposium*, 1999.

[7] Wikipedia, "FairPlay - DRM created by Apple," http://en.wikipedia.org/wiki/FairPlay, Oct. 2011.

[8] Android Pub, "[안드로이드 마켓] 보호된 어플(Protected App)의 apk 추출하기", http://www.android-pub.com/210599, Mar. 2010. (in Korean)

최 창 기

2001년 한양대학교 수학과 졸업(학사) 2012년 KAIST 소프트웨어 대학원 졸업(석사). 관심분야는 Embeded System, Security

한 태 숙

1976년 서울대학교 전자공학과 학사. 1978년 KAIST 전산학과 석사. 1990년 Univ. of North Carolina at Chapel Hill 박사 1991년~현재 KAIST 전산학과 교수. 관심분야는 프로그래밍언어 이론, Secure 소프트웨어, 임베디드 시스템