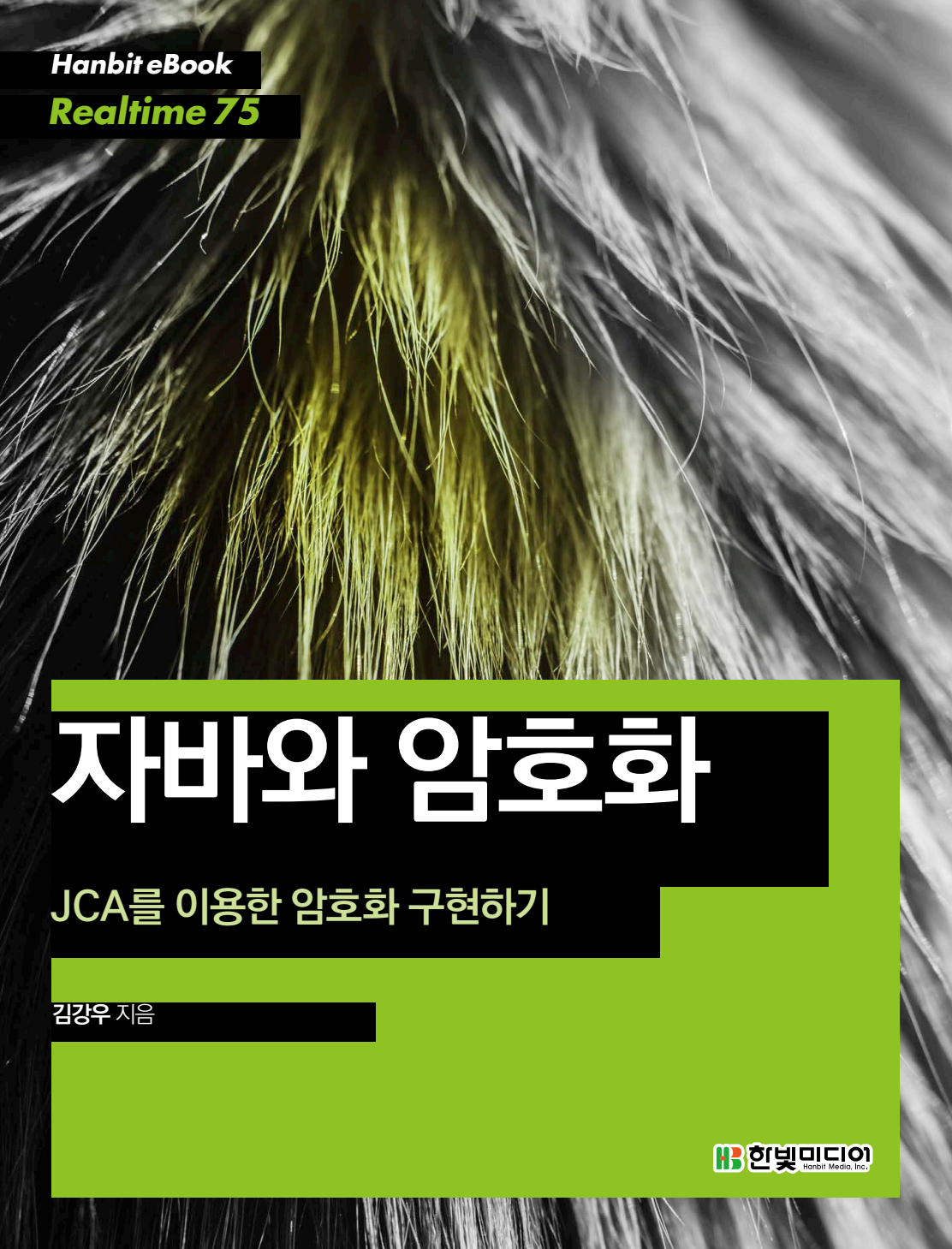


Hanbit eBook

Realtime 75

자바와 암호화

JCA를 이용한 암호화 구현하기

김강우 지음

한빛미디어 Hanbit Media, Inc.

자바와 암호화

JCA를 이용한 암호화 구현하기

자바와 암호화 JCA를 이용한 암호화 구현하기

초판발행 2014년 9월 23일

지은이 김강우 / **펴낸이** 김태헌
펴낸곳 한빛미디어(주) / **주소** 서울시 마포구 양화로 7길 83 한빛미디어(주) IT출판부
전화 02-325-5544 / **팩스** 02-336-7124
등록 1999년 6월 24일 제10-1779호
ISBN 978-89-6848-671-5 15000 / **정가** 12,000원

책임편집 김창수 / **기획·편집** 정지연
디자인 표지 여동일, 내지 스튜디오 [밈], 조판 최송실
영업 김형진, 김진불, 조유미 / **마케팅** 박상용, 김옥현

이 책에 대한 의견이나 오탈자 및 잘못된 내용에 대한 수정 정보는 한빛미디어(주)의 홈페이지나 아래 이메일로 알려주십시오.
한빛미디어 홈페이지 www.hanbit.co.kr / **이메일** ask@hanbit.co.kr

Published by HANBIT Media, Inc. Printed in Korea

지금 하지 않으면 할 수 없는 일이 있습니다.
책으로 펴내고 싶은 아이디어나 원고를 메일(ebookwriter@hanbit.co.kr)로 보내주세요.
한빛미디어(주)는 여러분의 소중한 경험과 지식을 기다리고 있습니다.

저자 소개

지은이_김강우

개발자의 탈을 쓴 원시인. 돌도끼에 몸을 의지한 채 세상을 떠돌아다니다가 현재는 롱아일랜드에서 은거 중. 인간의 감성은 사라지고, 기술의 발달로 획일성과 안락함만이 남은 개발계에서 한 줄기 빛을 찾아 헤매고 있다. 비 오는 날 창문 밖의 풍경 보는 것을 좋아하며, 의식의 표류를 즐기는 바보 중의 바보다.

저자 서문

개발자가 누릴 수 있는 즐거움 중 하나가 바로 암호학을 배우는 게 아닌가 싶습니다. 물론 현실 세계에서는 사용할 일이 없긴 하지만, 개발자에게 주어진 지적 욕구를 충족시키기에는 더할 나위 없이 좋은 분야라고 생각합니다(때론 더할 나위 없는 고통을 안겨주기도 하지만……).

이 책은 현대 암호학에서 주로 사용하는 기술에 대해서 최대한 간단히 소개하고, 자바에서 제공하는 JCA로 아주 간단(?)하게 암호화를 구현하는 방법을 설명하고 있습니다. 간단히 소개한다는 의미는, 머리를 아프게 하는 암호학 알고리즘에 대한 얘기는 전혀 없다는 뜻이므로 안심하고 읽을 수 있습니다. 빈약한 저자의 의지 때문에 못다 한 이야기가 많지만, 남김의 미는 항상 우리를 풍요롭게 하는 것 같습니다.

끝으로 머리 아픈 이 책을 편집해준 정지연 님과 2014년을 더 행복하게 보낼 수 있게 도와주신 하레 님, 아리 님, 조셉 님, 얀 님, 아키 님, 키욜 님, 리쯔 님, 안도 님, 그리고 또요 님께 감사의 말을 전합니다.

한빛 eBook 리얼타임

한빛 eBook 리얼타임은 IT 개발자를 위한 eBook입니다.

요즘 IT 업계에는 하루가 멀다 하고 수많은 기술이 나타나고 사라져 갑니다. 인터넷을 아무리 뒤져도 조금이나마 정리된 정보를 찾는 것도 쉽지 않습니다. 또한 잘 정리되어 책으로 나오기까지는 오랜 시간이 걸립니다. 어떻게 하면 조금이라도 더 유용한 정보를 빠르게 얻을 수 있을까요? 어떻게 하면 남보다 조금 더 빨리 경험하고 습득한 지식을 공유하고 발전시켜 나갈 수 있을까요? 세상에는 수많은 종이책이 있습니다. 그리고 그 종이책을 그대로 옮긴 전자책도 많습니다. 전자책에는 전자책에 적합한 콘텐츠와 전자책의 특성을 살린 형식이 있다고 생각합니다.

한빛이 지금 생각하고 추구하는, 개발자를 위한 리얼타임 전자책은 이렇습니다.

1. eBook Only - 빠르게 변화하는 IT 기술에 대해 핵심적인 정보를 신속하게 제공합니다.

   500페이지 가까운 분량의 잘 정리된 도서(종이책)가 아니라, 핵심적인 내용을 빠르게 전달하기 위해 조금은 거칠지만 100페이지 내외의 전자책 전용으로 개발한 서비스입니다. 독자에게는 새로운 정보를 빨리 얻을 수 있는 기회가 되고, 자신이 먼저 경험한 지식과 정보를 책으로 펴내고 싶지만 너무 바빠서 엄두를 못 내는 선배, 전문가, 고수 분에게는 보다 쉽게 집필할 수 있는 기회가 될 수 있으리라 생각합니다. 또한 새로운 정보와 지식을 빠르게 전달하기 위해 O'Reilly의 전자책 번역 서비스도 하고 있습니다.

2. 무료로 업데이트되는 전자책 전용 서비스입니다.

   종이책으로는 기술의 변화 속도를 따라잡기가 쉽지 않습니다. 책이 일정 분량 이상으로 집필되고 정리되어 나오는 동안 기술은 이미 변해 있습니다. 전자책으로 출간된 이후에도 버전 업을 통해 중요한 기술적 변화가 있거나 저자(역자)와 독자가 소통하면서 보완하여 발전된 노하우가 정리되면 구매하신 분께 무료로 업데이트해 드립니다.

3. 독자의 편의를 위해 DRM-Free로 제공합니다.

구매한 전자책을 다양한 IT 기기에서 자유롭게 활용할 수 있도록 DRM-Free PDF 포맷으로 제공합니다. 이는 독자 여러분과 한빛이 생각하고 추구하는 전자책을 만들어 나가기 위해 독자 여러분이 언제 어디서 어떤 기기를 사용하더라도 편리하게 전자책을 볼 수 있도록 하기 위함입니다.

4. 전자책 환경을 고려한 최적의 형태와 디자인에 담고자 노력했습니다.

종이책을 그대로 옮겨 놓아 가독성이 떨어지고 읽기 힘든 전자책이 아니라, 전자책의 환경에 가능한 한 최적화하여 쾌적한 경험을 드리고자 합니다. 링크 등의 기능을 적극적으로 이용할 수 있음은 물론이고 글자 크기나 행간, 여백 등을 전자책에 가장 최적화된 형태로 새롭게 디자인하였습니다.

앞으로도 독자 여러분의 충고에 귀 기울이며 지속해서 발전시켜 나가도록 하겠습니다.

차례

Part 2 JCA와 암호화

Part 1
암호화의 개념

암호화의 개념에 대해서 살펴보고, 현대 암호학에서 사용하는 암호화 종류와 현대 생활 전반에 사용하는 암호화 기술에 대해서 알아본다.

1 | 암호화란

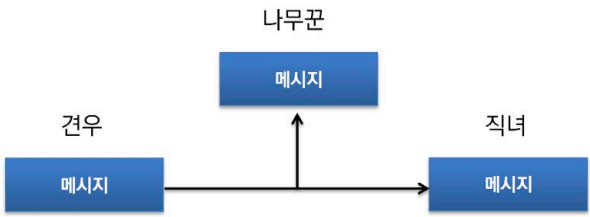
암호학cryptography은 암호화에 대한 과학이자 예술이다. 정보를 보호하기 위해서 언어학적 방법론과 수학적 방법론을 다루는 학문으로, 매우 다양한 분야에서 연구 개발되고 있다. 초기의 암호학은 메시지 보안 등의 암호화에 초점을 맞추었지만, 최근에는 인증과 전자서명 등 많은 기능을 포함하면서 그 범위가 넓어졌다. 이제 암호학은 전자상거래, 인터넷 뱅킹, 컴퓨터의 패스워드 등 현대 생활 전반에서 광범위하게 사용되고 있다.

이 책은 암호학의 일부분인 실용적인 측면에 초점을 맞췄다. 이 책의 목표는 자바를 이용하여 실제 시스템에서 암호를 적용하는 방법을 알려주는 데 있다. 사실 암호학은 매우 어려운 학문이다. 실제로 암호학을 모두 이해하는 것은 불가능하다. 하지만 제대로 사용하기 위해서는 최소한의 지식 정도는 알고 있어야 하지 않을까 해서 관련 지식을 가능하면 쉽게 풀어쓰려고 노력했다. 하지만 쉽게 쓰려고 하다 보니 중요한 점을 간과하는 누를 범했을 수도 있으므로 항상 '비판적 사고'로 이 책을 읽어주면 하는 바람이다.

1.1 암호화

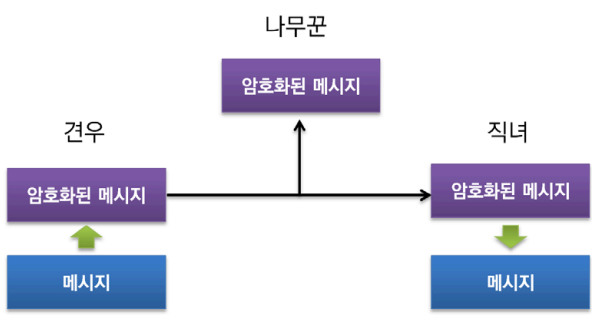
암호학의 최종 목표는 암호화다. 예를 들어, [그림 1-1]과 같이 견우와 직녀가 서로 통신하려고 한다고 가정해 보자. 그런데 일반적인 통신 채널은 안전하지 않아서 나무꾼이 통신 채널을 도청할 수 있다. 즉, 견우가 직녀에게 보낸 메시지를 나무꾼이 가로챌 수도 있다. 나무꾼의 도청을 방지하려면 견우와 직녀는 어떻게 해야 할까?

[그림 1-1]



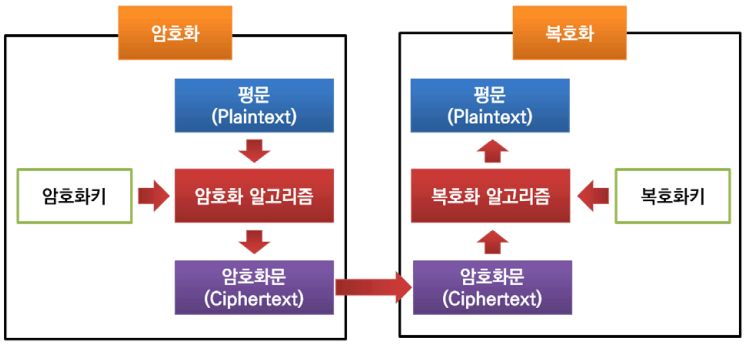
나무꾼이 견우와 직녀의 통신을 도청하지 못하게 하려면 [그림 1-2]와 같이 메시지를 암호화해서 보내면 된다. 나무꾼이 메시지를 가로챌 수도 있지만, 암호화된 메시지여서 원래 메시지를 알 수 없다.

[그림 1-2]



이처럼 암호화를 이용해서 보호해야 할 데이터(메시지)를 평문Plaintext이라 하고, 평문을 암호화 알고리즘을 이용해서 변환한 것을 암호문Ciphertext이라 한다. 이때 평문을 암호문으로 변환하는 과정을 암호화Encryption, 암호문을 평문으로 변환하는 과정을 복호화Decrypyion라고 한다. 그리고 이 과정에서 키Key가 사용된다. 자물쇠를 열기 위해서는 알맞은 열쇠가 필요하듯이 해당 키를 가진 자만이 정상적으로 암호화와 복호화를 할 수 있다.

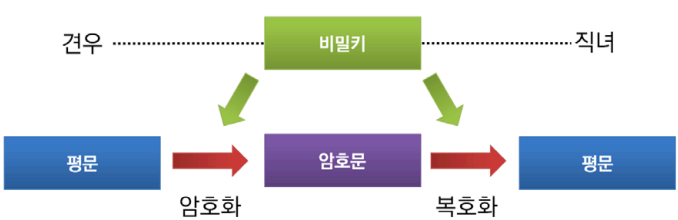
[그림 1-3]



1.2 비밀키(대칭키) 암호

견우는 메시지를 암호화해서 암호문을 직녀에게 보낼 것이고, 직녀는 암호문을 복호화해서 메시지를 확인할 것이다. 여기서 암호화할 때 사용하는 키(암호화키)와 복호화할 때 사용하는 키(복호화키)가 동일한데, 이러한 방식을 비밀키 암호화 또는 대칭키 암호화라고 한다.

[그림 1-4]



1.3 커코프 원칙

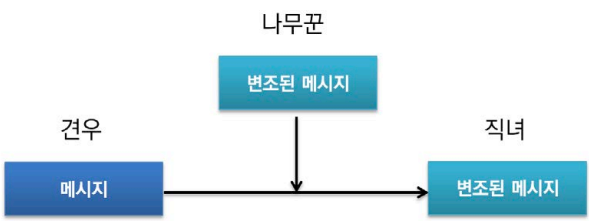
커코프의 원칙Kerckhoffs's principle이란 암호 시스템의 안전성은 암호화 알고리즘이 아니라 암호키의 비밀에만 의존해야 한다는 것을 말한다. 두 사람만을 위한 암호

시스템을 설계하지 않고 여러 사람이 사용하는 암호 시스템을 설계했다면, 암호 시스템에 속한 모든 사람은 같은 알고리즘을 사용할 것이다. 이 말은 즉, 같은 암호 시스템을 사용하고 있는 사람들은 암호 알고리즘을 알 수 있다는 뜻이다. 그래서 알고리즘의 비밀성을 유지하는 것이 매우 어렵고, 이런 상황에서 알고리즘의 비밀성이 주는 안전성은 거의 의미가 없어진다. 이런 이유로 암호 시스템의 안전성은 알고리즘이 아니라 키의 비밀성에만 의존해야 한다는 것이다.

1.4 메시지 인증 코드

견우와 직녀가 통신할 때 또 다른 문제가 발생할 수 있는데, 견우가 메시지를 보낼 때 나무꾼이 통신 채널에 끼어들어서 변조한 다른 메시지를 보낼 수 있다.

[그림 1-5]



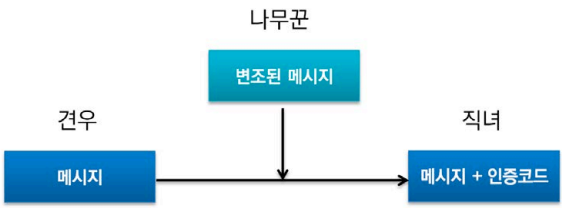
견우가 "사랑해요"라는 메시지를 직녀에게 보냈지만, 나무꾼이 중간에서 가로채서 "헤어져요"라고 바꿔서 직녀에게 보낼 수도 있다. 이런 문제를 해결하기 위해 인증이라는 개념을 사용한다.

인증하기 위해서는 암호화에 사용하는 비밀키처럼 견우와 직녀 두 사람만이 알고 있는 인증키라는 것을 사용한다. 견우가 메시지를 보낼 때 이 인증키를 가지고 메시지 인증 코드MAC, Message Authentication Code[01]라는 것을 생성한다. 이 인증 코드를 메시지와 함께 직녀에게 보내면 직녀는 수신된 메시지와 자신이 가진 인증키로 인

01 현대 암호학에서는 메시지 인증 코드을 생성하기 위해서 해시 함수를 사용하거나 블록 암호화를 사용한다.

증 코드를 생성할 것이고, 이 인증 코드를 수신된 인증 코드와 동일한지 확인한다. 인증 코드가 동일하다면 수신된 메시지는 견우가 보낸 것이다.

[그림 1-6]



나무꾼이 중간에서 변조된 메시지를 보낸다면 어떻게 될까? 나무꾼은 인증키를 알 수 없으므로 메시지에 맞는 인증 코드를 생성할 수 없고, 변조된 메시지를 받은 직녀는 메시지의 인증 코드와 자신이 가진 인증 코드가 일치하지 않음을 알게 된다. 따라서 직녀는 해당 메시지가 잘못되었다는 것을 알 수 있다.

그러나 인증 코드만 사용하는 방법에도 약간의 문제가 있다. 나무꾼이 견우의 메시지를 중간에서 삭제할 수 있다. 견우가 아무리 "사랑해요"라는 메시지를 보내도 직녀가 받지 못하면 의미가 없다. 또 다른 문제로, 기존에 견우가 보냈던 메시지와 인증 코드를 나무꾼이 기록해 두었다가 나중에 직녀에게 재전송하는 재현 공격Replay Attack이 있다. 어느 날 사랑싸움으로 화가 난 견우가 직녀에게 "헤어져"라고 메시지(물론 인증 코드도 함께)를 보냈다. 나무꾼이 이 메시지와 인증 코드를 기록해 두었다가 다음번 절묘한 순간에 직녀에게 그대로 보낸다면 직녀는 메시지와 인증 코드가 맞는 것이므로 의심하지 않을 것이고, 두 사람의 사랑은 비극적 결말로 치달을 것이다. 이런 문제점을 해결하기 위해서 메시지에 메시지 순번이나 시간값Timestamp을 포함하여 전송한다.

여기서 한 가지 의문을 가질 수도 있다. 일반적인 메시지라도 변조하는 게 가능하지만, 암호화된 메시지도 변조될까? 물론 쉬운 일은 아니지만, 전혀 불가능한 것은

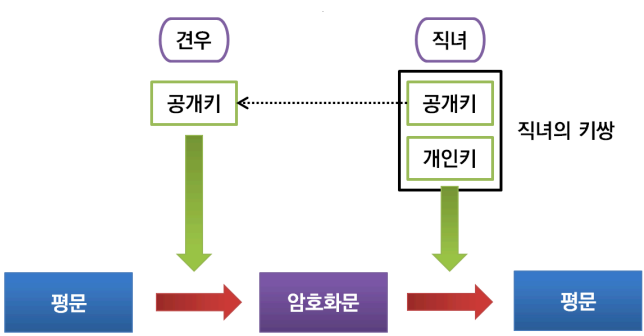
아니다. 대부분 혼동을 하는데, 메시지 암호화와 인증은 별개의 개념이다. 메시지를 암호화한다고 해서 변조를 막을 수 있는 것은 아니며, 메시지를 인증한다고 해서 비밀 전송이 보장되는 것도 아니다.

1.5 공개키 암호

암호화와 인증을 사용했으니 이제는 문제가 없을까? 사실 가장 어려운 문제가 남았다. 그건 바로 키 교환이다.

물리적으로 거리가 먼 곳에 있을 때 어떻게 키를 공유할 수 있을까? 통신 채널을 이용해서 보낼 수도 있지만, 나무꾼이 중간에서 키를 가로챌 수 있으므로 안전하지 못하다. 또한, 직녀는 견우만이 아니라 다른 친구들과도 통신한다. 직녀와 통신하는 친구가 10명이 있다면 10개의 키를 교환하고 관리해야 한다. 사람이 많아질수록 더욱 복잡해진다. 암호학에서는 이 어려운 문제를 해결하기 위해 공개키 암호를 이용한다. 공개키 암호는 비밀키 암호와 달리 암호화와 복호화에 사용하는 키가 다르다. 우선 특정 알고리즘을 사용하여 공개키와 개인키로 이루어진 키쌍KeyPair을 생성한다. 그런 다음 공개키로 메시지를 암호화하고, 개인키로 메시지를 복호화한다.

[그림 1-7]



공개키는 말 그대로 공개된 키여서 누구나 접근할 수 있다. 견우는 직녀의 공개키와 암호화 알고리즘으로 메시지를 암호화해서 보낸다. 직녀는 자신의 개인키와 복호화 알고리즘으로 메시지를 복호화해서 메시지를 얻을 수 있다. 이러한 방식을 공개키 암호화 또는 비대칭키 암호화라고 한다.

공개키 암호 방식은 키를 분배하고 관리하는 문제를 간단하게 해결한다. 직녀는 모든 사람이 사용할 수 있는 공개키 하나만 분배하면 된다. 견우뿐만 아니라 친구들과도 같은 공개키로 통신할 수 있으므로 키의 관리도 쉬워진다.

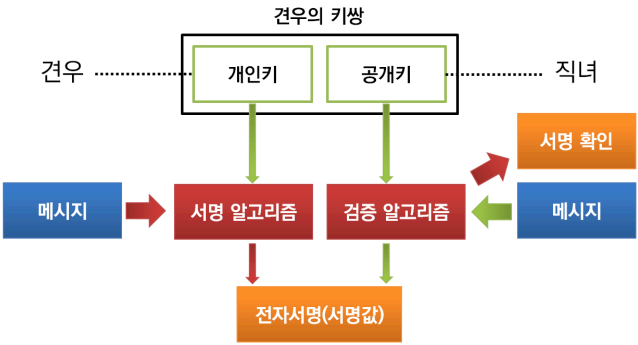
이처럼 공개키 암호화가 더 좋다면 비밀키 암호화는 필요 없을까? 공개키 암호화는 수학적인 이론에 기반을 두기 때문에 비밀키 암호화보다 그 처리가 매우 느리므로 모든 경우에 공개키 암호화를 적용하면 비용이 매우 많이 발생하여 비효율적이다. 그래서 실제 시스템에서는 공개키 암호화와 비밀키 암호화를 혼합해서 사용한다. 공개키 암호화는 메시지를 암호화할 때 사용하는 비밀키를 암호화하기 위해 사용한다. 이 방법은 공개키 암호화의 유연성과 비밀키 암호화의 효율성을 결합한 형태다.

1.6 전자서명

전자서명Digital Signatures은 메시지 인증 코드와 비슷하지만, 메시지 인증 코드와 달리 공개키 암호 방식을 사용한다. 메시지 인증 코드는 인증키라는 동일한 키를 가지고 인증과 검증을 한다. 하지만 전자서명은 개인키로 서명값을 생성하고, 공개키로 서명값을 검증하므로 누가 서명했는지 증명할 수 있다.

견우가 키 생성 알고리즘으로 키쌍을 생성하고 직녀에게 메시지를 보낼 때 서명 알고리즘으로 서명값을 생성하여 메시지와 서명값을 같이 보낸다. 직녀는 서명을 검증하기 위해 견우의 공개키를 사용한다. 직녀는 견우의 공개키와 검증 알고리즘으로 수신한 메시지와 서명값이 견우가 보낸 것인지 확인할 수 있다.

[그림 1-8]



메시지 인증 코드는 인증키를 가진 두 사람 모두 생성할 수 있다. 즉, 메시지 인증 코드로는 누가 인증 코드를 생성했는지는 알 수 없다. 또한, 검증하기 위해서도 인증키가 필요하므로 검증하려는 사람도 인증 코드를 생성할 수 있다. 하지만 전자서명은 자신의 개인키로 서명값을 생성한다. 견우의 개인키는 견우만이 가지고 있기 때문에 이론상으로 다른 사람은 서명값을 생성할 수 없다. 서명값을 검증할 때에도 견우의 공개키만 있으면 누가 서명했는지 알 수 있다. 이러한 이유로 이를 전자서명이라고 한다.

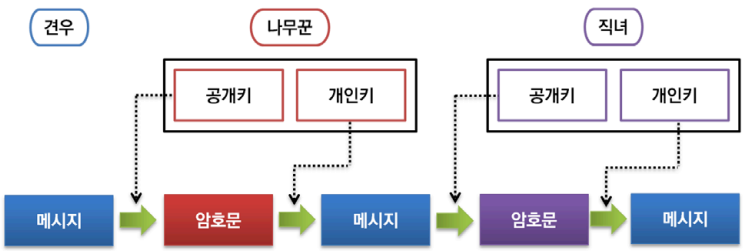
1.7 공개키 기반 구조

공개키 암호화를 실제 시스템에서 사용하기에는 약간의 문제점이 있는데, 바로 공개키를 어떻게 찾느냐다. 엄밀히 말하면 해당 공개키가 누구의 공개키인지 어떻게 확신할 수 있느냐 하는 문제다.

견우가 직녀에게 메시지를 암호화하여 보내려면 직녀의 공개키를 가져와야만 한다. 하지만 나무꾼이 중간에서 직녀를 사칭하여 자신의 공개키를 공개할 수도 있다. 이럴 경우 견우는 나무꾼이 보낸 공개키를 직녀의 공개키라 믿고 메시지를 암호화해서 보낸다. 나무꾼은 그 메시지를 중간에서 가로채서 자신의 개인키로 복호

화해서 메시지를 볼 수 있다. 그런 다음 나무꾼은 직녀의 공개키로 견우가 보낸 메시지를 다시 암호화해서 직녀에게 보낸다. 결국, 직녀는 견우가 보낸 메시지를 받아볼 수는 있지만 나무꾼도 그 메시지 내용을 알게 된다.

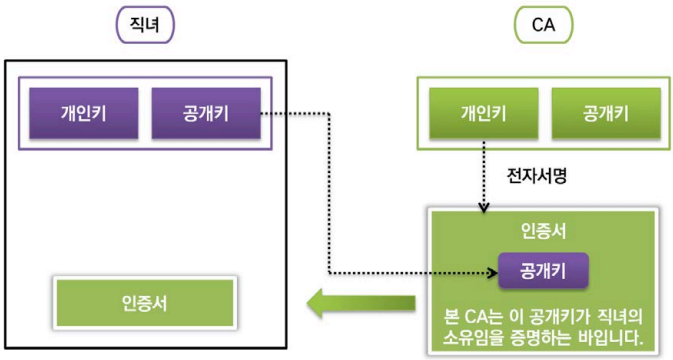
[그림 1-9]



이런 문제점을 해결하기 위해서 PKI[Public Key Infrastructure]라고 불리는 공개키 기반 구조를 사용한다. PKI는 CA[Certificate Authority]라고 하는 인증기관을 사용한다. 각 사용자는 공개키쌍을 생성한 다음, CA에 해당 공개키가 본인의 것이라는 것을 알려준다. 그러면 CA가 전자서명을 이용하여 사용자의 공개키와 몇 가지 정보를 포함해서 서명한다. 이게 바로 인증서[Certificate]다. 인증서에는 보통 "본 CA는 이 공개키가 XY의 소유임을 증명하는 바입니다"라고 명시되어 있으며, 인증서의 만기일 등 다른 유용한 정보가 포함되어 있다.

인증서를 이용하면 견우는 직녀의 공개키를 얻을 수 있다. 중간에서 나무꾼이 자신의 공개키를 직녀의 공개키처럼 속이려고 해도 견우가 CA의 공개키를 이용하여 인증서에 있는 직녀의 공개키를 검증할 수 있으므로 속일 수가 없다. 하지만 여기에는 한 가지 전제 조건이 있다. 견우는 CA의 공개키를 가지고 있어야 한다. (실제로는 애플리케이션이나 운영체제가 CA의 공개키를 가지고 있다.)

[그림 1-10]



현재 많이 사용하는 HTTPS[Hypertext Transfer Protocol over Secure Socket Layer]는 웹 통신 프로토콜인 HTTP의 보안이 강화된 버전이다. HTTPS는 TLS[Transport Layer Security]/SSL[Secure Socket Layer]을 사용하여 데이터를 암호화해서 보내는데, 이 구조가 바로 PKI를 바탕으로 한다. 웹 브라우저로 해당 사이트에 접속하면 해당 사이트의 인증서와 브라우저의 CA를 이용하여 인증서를 검증한다. 국내에서 인터넷 뱅킹 등에 많이 사용하는 공인인증서도 이 구조를 바탕으로 한다.

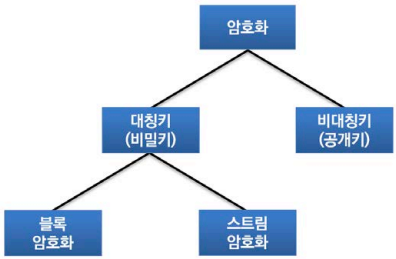
1.8 암호화 방식

현대 암호학에서 사용하는 암호화 방식은 사용하는 키의 종류와 입력 데이터의 종류에 따라 다음과 같이 분류할 수 있다.

[표 1-1] 암호화 방식의 분류

구분	종류	설명
키	대칭키(비밀키) 암호화	암호화와 복호화에 동일한 키를 사용한다.
	비대칭키(공개키) 암호화	암호화에 사용하는 키와 복호화에 사용하는 키가 다르다.
입력 데이터	블록 암호화	입력 데이터가 고정된 크기의 블록 단위 데이터다.
	스트림 암호화	입력 데이터가 연속된 스트림 데이터다.

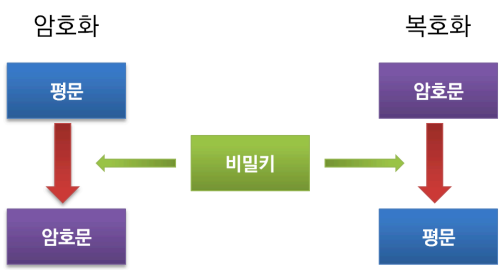
[그림 1-11] 암호화 방식의 분류



2 | 대칭키(비밀키) 암호

대칭키Symmetric-key 암호화는 암호화에 사용하는 비밀키와 복호화에 사용하는 비밀키가 동일하다는 특징이 있으며, 비밀키Secret-key 암호화라고도 한다. 이 암호 방식은 공개키 암호 방식보다 연산 속도가 빨라서 데이터 암호화에 많이 사용한다. 하지만 송신자와 수신자 모두 동일한 키를 공유해야 하므로 키 관리가 어렵다는 단점이 있다.

[그림 2-1]



대칭키 암호는 데이터를 변환하는 단위에 따라 블록 암호와 스트림 암호로 분류할 수 있다.

2.1 블록 암호화 알고리즘

블록 암호Block cipher는 고정된 크기의 블록 단위로 데이터를 암/복호화하는 함수로, 현재 많이 사용하는 블록 단위의 크기는 128비트다. 즉, 128비트의 평문을 입력받아서 128비트의 암호문을 생성하고, 복호화는 128비트의 암호문을 입력받아서 128비트의 평문을 생성한다. 이처럼 블록 암호는 평문과 암호문의 크기가 동일하다.

블록 암호에서 암/복호화하려면 키가 필요한데, 이 키를 보통 비밀키라고 부른다. 비밀키의 크기는 암호 알고리즘에 따라 다르지만, 일반적으로 128비트 또는 256비트다.

대표적인 블록 암호화 알고리즘은 다음과 같다.

DES/3DES(Triple DES)

DES(Data Encryption Standard)는 기본으로 페이스텔(Feistel) 구조[01]를 가지며, 1977년 미국 NBS(National Bureau of Standard, 현 NIST)에서 미국 표준 알고리즘으로 채택되었다. 과거에 많이 사용하던 암호 알고리즘이지만, 56비트의 제한적인 키와 64비트의 작은 블록을 사용하므로 빠른 컴퓨팅 환경에는 적합하지 않아서 현재는 잘 사용하지 않는다. 한때는 이러한 단점을 보완하기 위해 3번의 DES 암/복호화 과정을 수행하는 블록 암호인 3DES를 만들기도 했다. 기존의 DES 방식을 '암호화(Encryption) → 복호화(Decryption) → 암호화(Encryption)' 순으로 3번 사용했다. 그래서 3(Triple)DES라고 부르기도 하고, Encryption, Decryption, Encryption의 첫 글자를 따서 DESede라고 부르기도 한다. 하지만 블록 크기가 작다는 문제와 DES보다 느린 속도(1/3) 때문에 세월의 뒤안길로 사라졌다.

AES(Rijndael)

AES(Advanced Encryption Standard)는 안정성에 문제가 있는 DES를 교체하기 만들어진 미국 정부 표준이다. NIST(미국 국립표준기술연구소)는 암호학계에 대체할 암호 알고리즘 제안을 요청했고, 이에 총 15개의 제안서가 제출되었다. 그 중 Rijndael의 암호가 최종 선택되었고, 이것이 바로 AES다. AES는 2001년에 표준으로 제정되었다. 이 알고리즘은 128/192/256비트의 키 길이를 제공하고, 키 길이에 따라 10/12/14라운드(round)를 사용한다. 미국 정부 표준이라는 점에서 많이 사용하고 있다.

---

01 http://en.wikipedia.org/wiki/Feistel_cipher

SEED

SEED는 1999년 2월 한국인터넷진흥원과 국내 암호 전문가들이 순수 국내 기술로 개발한 페이스텔 구조의 128비트 블록 암호 알고리즘이다. 1999년에는 128비트 비밀키를 지원하는 SEED 128이, 2009년에는 256비트 비밀키를 지원하는 256이 개발되었다. SEED 128은 1999년 9월에 TTA(정보통신단체표준)로 제정되었으며, 2005년에는 ISO/IEC 국제 블록 암호와 IETF 표준으로 제정되었다.

ARIA

ARIA는 경량 환경 및 하드웨어 구현을 위해 최적화된 Involutional SPN Substitution-Permutation Network 구조를 가진 128비트 블록 암호 알고리즘이다. 128/192/256비트의 키 길이를 제공하며, 키 길이에 따라 12/14/16라운드를 사용한다. 그리고 대부분의 연산은 XOR 같은 단순한 바이트 단위 연산으로 구성된다. ARIA라는 이름은 Academy(학계), Research Institute(연구소), Agency(정부기관)의 첫 글자를 딴 것으로, ARIA 개발에 참여한 학/연/관의 공동 노력을 나타낸다.

2.2 블록 암호화 운영 모드

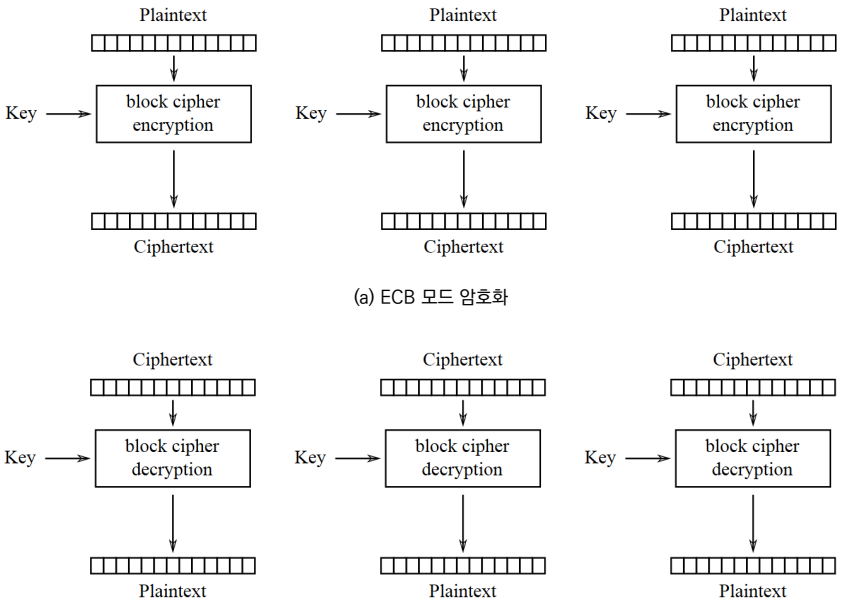
블록 암호화 운용 모드는 블록 암호를 반복적으로 사용하는 절차를 말한다. 암호화하려는 평문의 길이는 매우 다양하다. 그리고 블록 암호는 고정된 길이 단위로 동작하므로 암호화하기 위해서는 가변 길이의 데이터를 블록 단위로 나누어야 한다. 이렇게 나누어진 블록들을 어떤 방식으로 사용할지 정해야 하는데, 이 방법을 블록 암호화 운영 모드라고 한다.

ECB

ECB Electronic Codebook 모드는 주어진 평문을 블록 크기로 나누어서 차례대로 암호화하는 방식으로, 각 블록은 동일한 키를 사용한다. 평문 블록과 암호문 블록이 일

대일 대응관계인 가장 단순한 운영 모드다. 두 개의 평문 블록이 같다면 각 평문에 대응하는 암호문 블록도 동일하게 생성된다는 취약점이 있다.

[그림 2-2] ECB 모드[02]



(a) ECB 모드 암호화

(b) ECB 모드 복호화

CBC

CBC[Cipher Block Chaining] 모드는 이전 암호문 블록과 현재 평문 블록을 XOR 연산하고 그 값을 알고리즘의 입력값으로 사용하여 암호문 블록을 생성하는 방식으로, 각 블록은 동일한 키를 사용한다. 첫 번째 블록은 이전 암호문 블록이 없으므로 초기 벡터[IV, Initialization Vector][03]를 사용하여 XOR 연산한다.

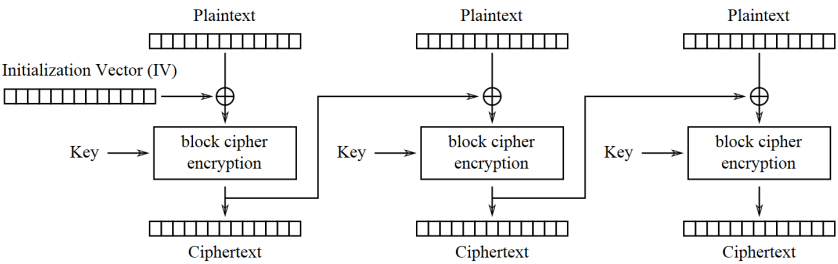
---

02 [그림 2-2] ~ [그림 2-6] 출처: http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

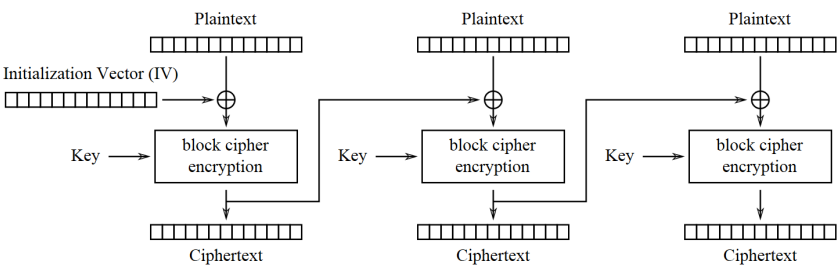
03 IV는 첫 번째 블록을 암호화할 때 사용하는 초기값을 의미한다.

이전 암호문 블록을 사용하여 현재 암호화할 평문 블록에 영향을 줌으로써 ECB 모드의 문제점을 보완했다. 즉, 똑같은 평문 블록들이 서로 다른 암호문 블록으로 생성된다. 그러나 순차적으로 처리해야 하기 때문에 병렬 처리가 불가능하다는 단점이 있다.

[그림 2-3] CBC 모드



(a) CBC 모드 암호화



(b) CBC 모드 복호화

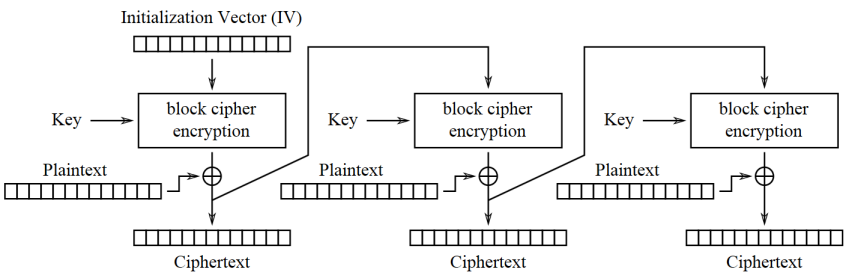
CFB

CFB[Cipher Feedback] 모드는 이전 암호문 블록을 암호 알고리즘에 입력하여 얻은 결과값과 현재 평문 블록을 XOR 연산하여 암호문 블록을 생성한다. 첫 번째 평문 블록의 암호화 과정에서는 초기 벡터(IV)를 암호 알고리즘의 입력값으로 사용한다.

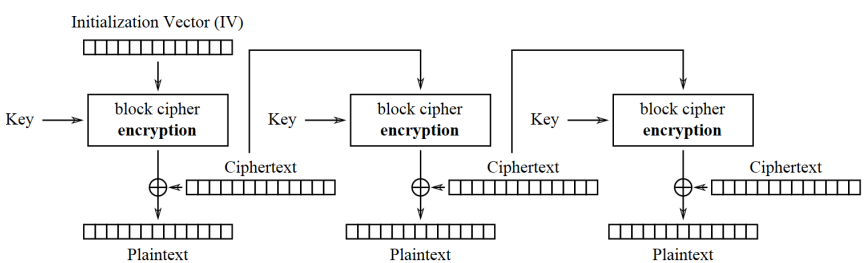
앞에서 설명한 ECB, CBC 모드는 평문을 블록 암호화했지만, CFB 모드는 평문 자

체를 암호화하지 않고, 대신 블록 암호를 이용해 만든 키 스트림을 평문과 XOR 연산함으로써 암호문을 생성한다. 이렇게 키 스트림을 생성하기 때문에 스트림 암호처럼 사용하기도 한다. 복호화 연산이 암호화 연산과 완전히 같다는 것과 패딩Padding[04]할 필요가 없다는 장점이 있다.

[그림 2-4] CFB 모드



(a) CFB 모드 암호화



(b) CFB 모드 복호화

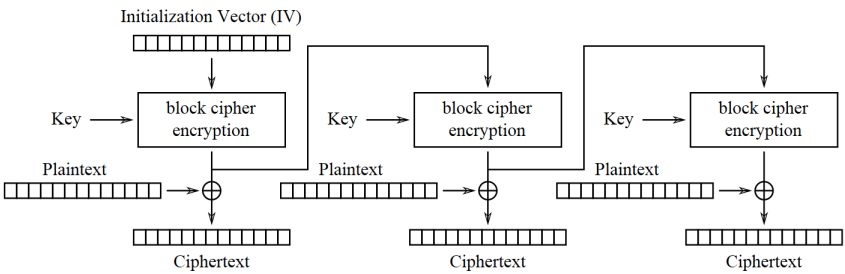
OFB

OFBOutput Feedback 모드는 이전 블록의 암호 알고리즘 출력값을 현재 암호 알고리즘의 입력값으로 하여 얻은 결과값과 평문 블록을 XOR 연산하여 암호문 블록을 생성한다. 첫 번째 평문 블록의 암호화 과정에서는 초기 벡터를 암호 알고리즘의

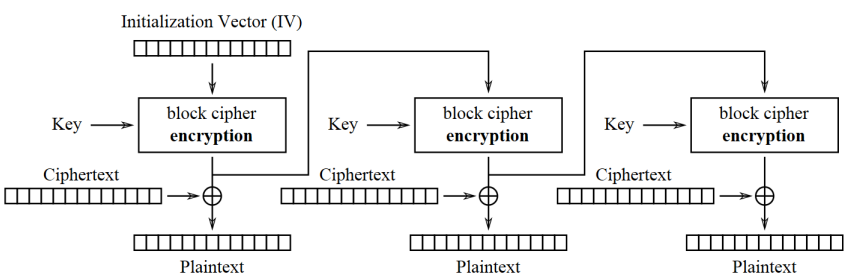
04 2.4 패딩 참조

입력값으로 사용한다. CFB 모드처럼 평문 자체를 암호화하지 않고, 블록 암호를 이용하여 만든 키 스트림을 평문과 XOR 연산함으로써 암호문을 생성한다. 이렇게 키 스트림을 생성하기 때문에 스트림 암호처럼 사용하기도 한다. 복호화 연산이 암호화 연산과 완전히 같다는 것과 패딩할 필요가 없다는 장점이 있다.

[그림 2-5] OFB 모드



(a) OFB 모드 암호화



(b) OFB 모드 복호화

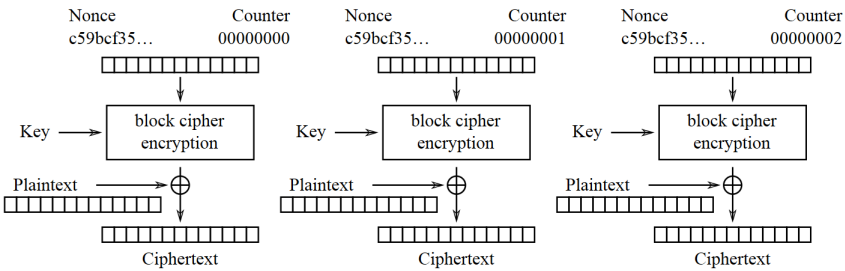
CTR

CTR[Counter] 모드는 특정 형태의 유일한 넌스[Nonce][05]와 1씩 증가하는 카운터 값을 붙인 것을 암호화하여 단일 블록 형태의 키 스트림을 만든다. 이 키 스트림을 평문 블록과 XOR 연산하여 암호문 블록을 생성한다. OFB 모드와 마찬가지로 스트림 암

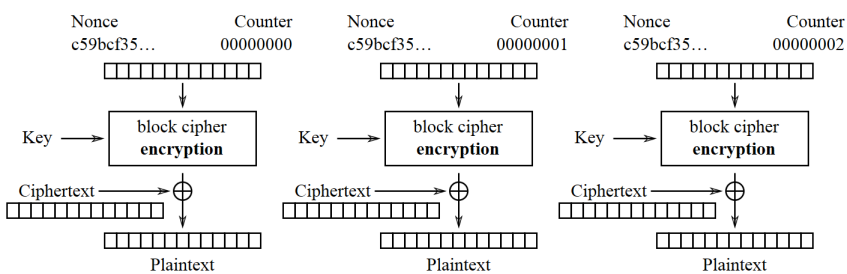
05 한 번만 사용되는 임의의 값을 의미한다.

호처럼 사용할 수 있다. 복호화 연산이 암호화 연산과 완전히 같다는 것과 패딩할 필요가 없다는 장점이 있다. 또한, 키 스트림의 계산을 병렬 처리할 수 있으므로 빠르게 계산할 수 있다는 장점도 있다.

[그림 2-6] CTR 모드



(a) CRT 모드 암호화



(b) CRT 모드 복호화

2.3 암호화와 인증을 결합한 블록 암호 운영 모드

앞에서 살펴본 블록 암호 운영 모드는 암호화 기능만을 제공한다. NIST는 CCM과 GCM으로 부르는 두 개의 모드를 표준으로 선택했는데, 이 모드들은 암호화뿐만 아니라 메시지 인증 기능까지 제공한다.

CCM

CCM(Counter with CBC-MAC)은 암호화와 메시지 인증 기능을 동시에 제공하는 운영 모드다. CCM은 메시지 인증을 위해 CBC-MAC 모드를 사용하고, 메시지 암호화를 위하여 CTR 모드를 사용한다.

GCM

CCM의 성능 향상을 위해 CTR 모드의 변종인 CWC(Carter-Wegman CTR) 모드를 만들었는데, 이 모드는 인증에 병렬 연산이 가능한 일반 해시(Hash)를 사용한다. 그리고 하드웨어 구현 시 일반 해시를 좀 더 효율적으로 적용할 수 있는 GCM(Galois/Counter Mode)을 만들었다.

2.4 패딩

블록 암호는 고정된 길이의 블록을 암호화한다. 예를 들어, 128비트의 블록 암호를 사용한다고 가정하면 128비트의 길이를 가진 평문을 사용해야 한다. 하지만 일반적으로 암호화하는 평문은 128비트 이상이고, 그 길이가 매우 다양하다. 이러한 임의의 길이를 가진 데이터를 블록 길이에 맞추는 것을 패딩(Padding)이라고 한다. 패딩에는 여러 규칙이 있지만, 가장 중요한 규칙은 패딩된 부분을 분리할 수 있어야 한다는 것이다. 즉, 패딩된 데이터에서 원래 데이터를 복구할 수 있어야 한다.

2.5 스트림 암호

스트림 암호(Stream cipher)는 블록 단위로 암/복호화하는 블록 암호와 달리, 키 스트림이라고 하는 의사난수(Pseudo- random)를 연속적으로 생성하여 암호화하려는 데이터와 결합하는 구조를 가진다. 일반적인 스트림 암호는 의사난수를 1비트 단위로 생성하고, 생성한 값과 암호화하려는 데이터를 XOR 연산하여 1비트의 암호화된 데이터를 얻는다. 이 방식은 하드웨어 구현이 간편하고 속도가 빠르므로 무선 통신

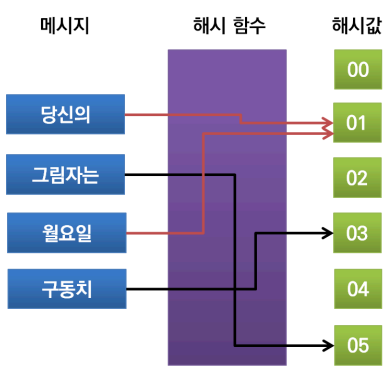
등의 환경에서 많이 사용된다. 대표적인 스트림 암호 알고리즘으로는 RC4, RC5와 A5/1, A5/2, A5/3 등이 있다.

3 | 해시 함수

해시 함수Hash Function는 임의의 크기를 가진 데이터를 입력받아서 고정된 크기의 결과값을 출력하는 함수로, 메시지 다이제스트Message Digest라고도 한다.

해시 함수는 암호학뿐만 아니라 여러 분야에서 사용되며, 단방향성과 충돌이라는 특징이 있다. 단방향성이란 결과값을 계산하기는 쉽지만 그 반대의 상황, 즉 결과값을 가지고 입력값을 구하기는 어렵다는 것을 말한다. 다시 말해, 해시 함수를 이용하여 메시지를 가지고 해시값을 생성하는 것은 쉽지만, 해시값을 가지고 메시지를 구하는것은 어렵다. 충돌이란 입력값이 달라도 동일한 결과값이 생성될 수 있다는 것을 말한다. 고정된 크기의 결과값을 가지고 있으므로 출력할 수 있는 값은 한정될 수밖에 없다. 1바이트 크기의 해시값을 생성하는 해시 함수라면 2의 8승, 256개의 해시값만을 가질 수 있다. 하지만 입력할 수 있는 값은 무한히 많으므로 입력값은 다르지만 출력값이 같아지는 경우가 존재할 수밖에 없다.

[그림 3-1]



해시 함수는 같은 입력에 대해서는 항상 같은 출력이 나오므로 입력한 데이터에 대한 지문Fingerprint을 생성함으로써 데이터의 오류나 변조를 탐지할 수 있는 무결성

을 제공한다. 또한, 암호학적 의사난수를 만들어 키를 생성하거나 전자서명에서 메시지를 축약할 때 사용한다.

3.1 해시 함수의 조건

암호학에서 사용하는 해시 함수는 다음 조건을 만족해야 한다.

- **역상 저항성**preimage resistance: 단방향 함수여야 한다. 즉, 입력값을 알면 해시값을 구하기 쉽지만, 해시값을 알아도 입력값은 구할 수 없어야 한다.
- **제2 역상 저항성**second preimage resistance: 입력값으로 구한 해시값과 동일한 해시값을 가진 다른 입력값을 찾을 수 없어야 한다. 즉, 해시값에 영향을 주지 않으면서 입력값을 변경할 수 없어야 한다.
- **충돌 저항성**collision resistance: 충돌에 대해 안전해야 한다. 즉, 완벽한 충돌 회피란 있을 수 없지만 같은 해시값을 생성하는 두 개의 입력값을 찾을 수 없어야 한다.

3.2 해시 함수 알고리즘

MD5

MD5Message Digest Algorithm 5는 1991년에 로널드 라이베스트Ronald Rivest가 고안한 128비트 해시 함수다. MD4를 대체하기 위해 만들었으나 현재는 안전성 문제로 보안 관련 용도로는 사용하지 않는다.

SHA

SHASecure Hash Algorithm는 NSA(미국 국가 안보국)에서 제안하고 NIST에서 표준화한 해시 알고리즘이다. 처음 제안된 해시 알고리즘은 SHA라 명명했으나, 지금은 대부분 SHA-0이라고 부른다. 2년 후 SHA-0의 취약성을 개선한 SHA-1을 발표하였다.

SHA2는 NIST에서 2004년에 발표한 네 개의 해시 함수(SHA-224, SHA-256, SHA-384, SHA-512)를 의미하는데, SHA-2 패밀리라고도 부른다.

SHA-3은 SHA-2를 대체하기 위해서 NIST가 공모한 해시 함수로, 기존의 해시 함수와는 다르게 직접 설계한 것이 아니라 공개적인 방법을 통해서 후보를 모집한 다음 안정성을 분석하여 걸러내는 방식으로 진행하였다. 2012년 10월 1일에 Keccak이 SHA-3으로 선정되었다.

3.3 해시 함수의 용도

체크섬

일반적인 해시 함수는 데이터의 체크섬Checksum을 생성하기 위해서 사용한다. 파일 다운로드 사이트에서 자주 보이는 체크섬이 바로 해당 파일의 해시값을 의미한다. 이 체크섬은 내려받은 파일이 정상인지 아닌지를 판단하기 위하여 사용한다.

패스워드 저장

사용자의 패스워드를 저장할 때에도 사용한다. 해시 함수는 단방향 알고리즘이라서 해시값에서 원본값을 복원할 수 없으므로 안전하게 패스워드를 저장하기 위해 사용하기도 한다. 단, 주의할 점이 있는데 단방향 알고리즘이라고 해서 원본값을 찾을 수 없는 것은 아니다. 앞에서 말한 것처럼 해시값에서 원본값을 복원할 수는 없지만, 단순무식하게 입력할 수 있는 모든 값을 해시 함수를 통해서 해시값을 만든 다음 그 해시값을 비교하면, 원본값이 무엇인지 알 수 있다. 물론 모든 해시값을 만드는 데 시간이 오래 걸리지만, 아주 단순하게 해시 함수를 사용한다면 미리 그 해시값을 만들어 놓고 비교할 수도 있다. 이런 이유로 패스워드에 사용할 수 있는 문자 종류가 적거나 패스워드의 길이가 짧을 경우에는 위험하다. 그래서 해시 함수를 이용해서 패스워드를 저장할 때에는 소금(Salt)을 살짝 뿌려주거나 이터레이션Iteration을 조절해주는 것이 좀 더 안전하다.

난수 생성

암호학에서 해시 함수는 의사난수를 생성할 때도 사용한다. 물리적 장치 없이 실제 난수를 생성하는 것은 거의 불가능하므로 의사난수라는 것을 생성해서 사용하는데, 이때 해시 함수를 많이 사용한다.

메시지 다이제스트

해시 함수의 다른 이름은 메시지 다이제스트 즉, 메시지 축약을 의미한다. 공개키 암호 알고리즘을 사용하는 전자서명은 복잡한 계산이 필요하므로 비용이 많이 든다. 그래서 일반적으로 메시지 대신 메시지의 해시값을 구한 다음 서명한다. 예를 들어, SHA-256 알고리즘은 결과값의 크기가 256비트이므로 몇만 비트인 메시지를 직접 서명하는 것보다 해시값에 서명하는 것이 훨씬 빠르다. 단, 이 방법이 안전하려면 사용하는 해시 함수가 암호학적 조건을 만족해야만 한다.