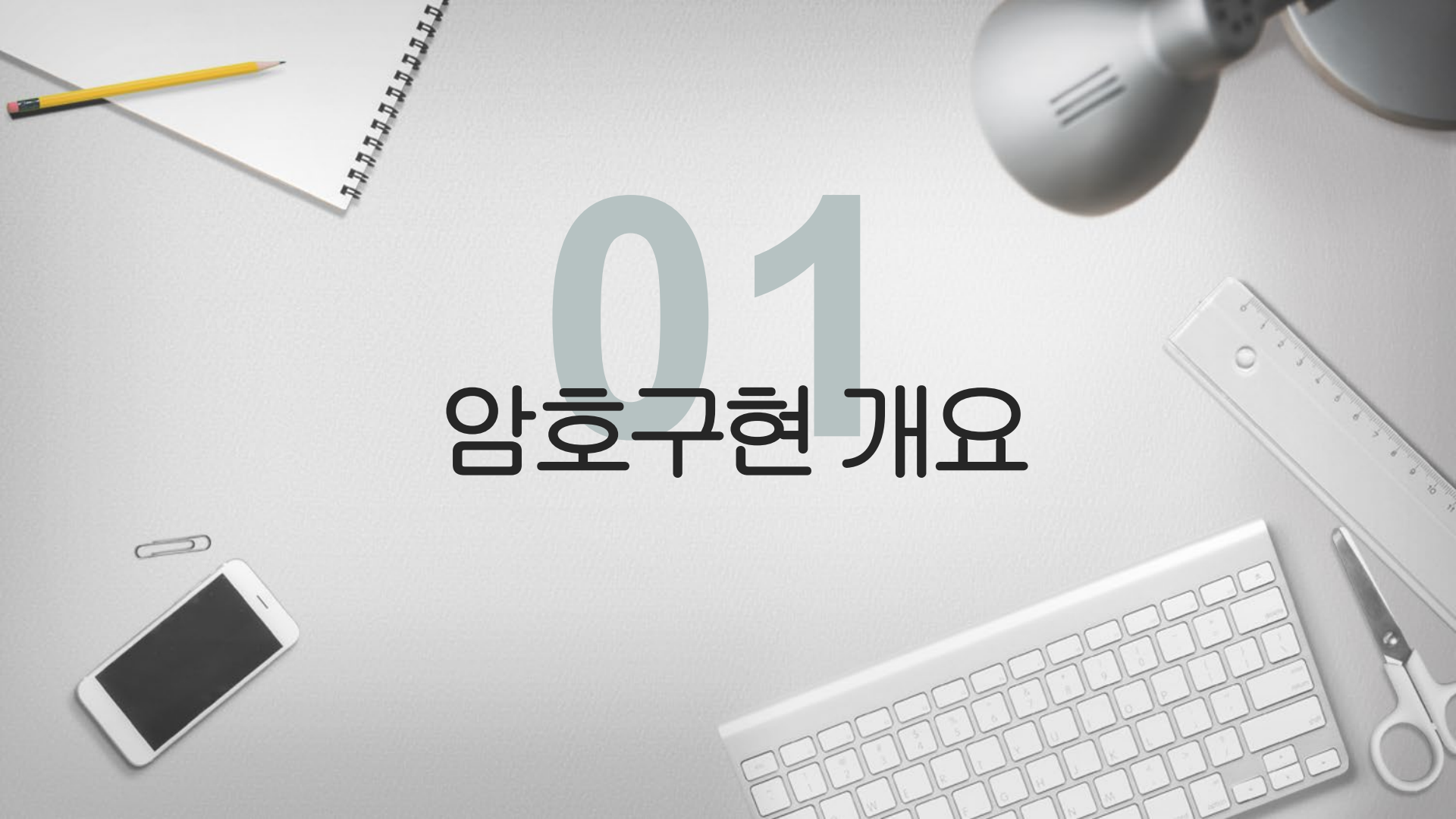


01

암호구현 개요

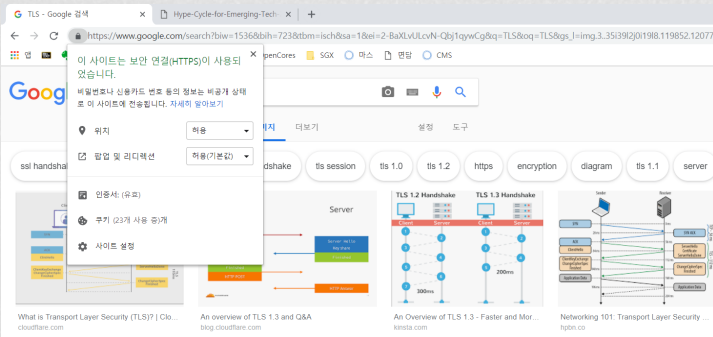
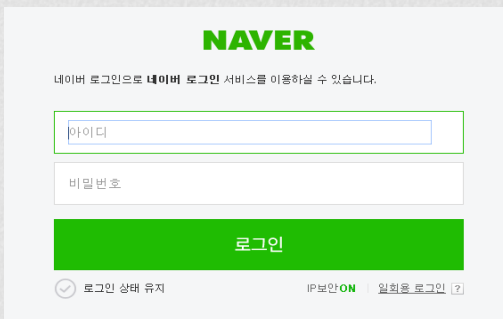
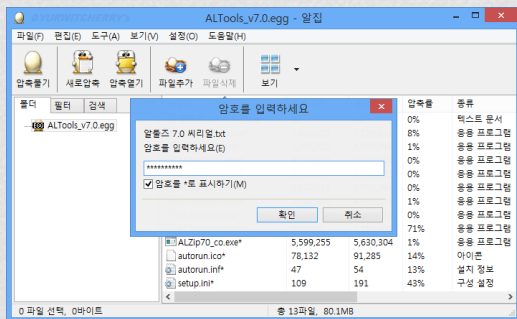


실생활 속의 암호



암호란 무엇인가?

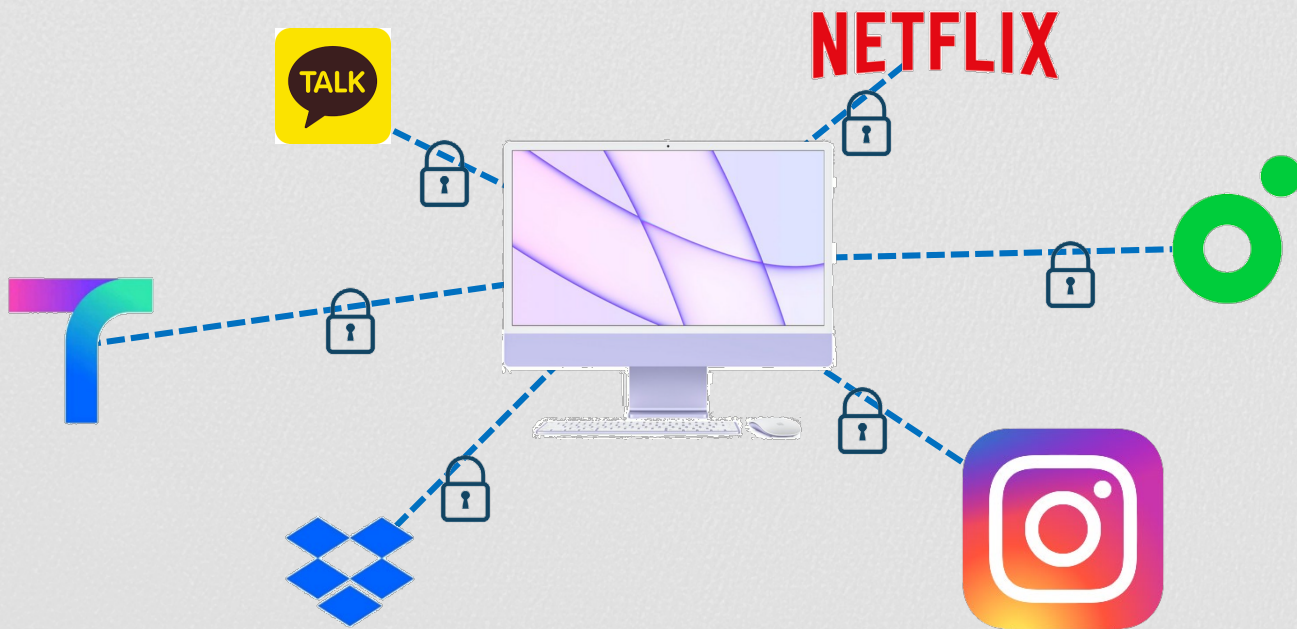
- ▶ 중요한 정보를 적법한 사용자만이
안전하고 신속하게 접근가능하도록 해주는 기술



암호 활용 분야

컴퓨터 네트워크와 암호

- ▶ 컴퓨터를 통해 제공되는 온라인 서비스들은 암호화 과정을 통해 네트워크 안전성을 보장받고 있음 (TLS/SSL, SSH, SFTP)



암호의 악의적인 사용



랜섬웨어 복호화 방법?

- ▶ 안전한 암호화를 사용하였다면 복호화는 불가능



암호의 위력

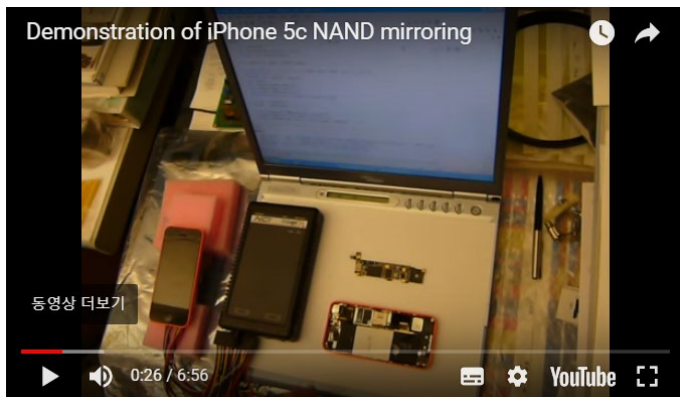
개인의 스마트폰을 해킹하기 위해서는 국가적 비용 필요

- ▶ 암호화는 개인이 가질 수 있는 강력한 비대칭 전략 무기

Academic beats FBI by unlocking Apple iPhone for £75

A Cambridge researcher shows the FBI there was no need to launch a bitter court battle with Apple over a terrorist's phone.

06:00, UK,
Wednesday 21 September 2016





matrixSSL

wolfSSL

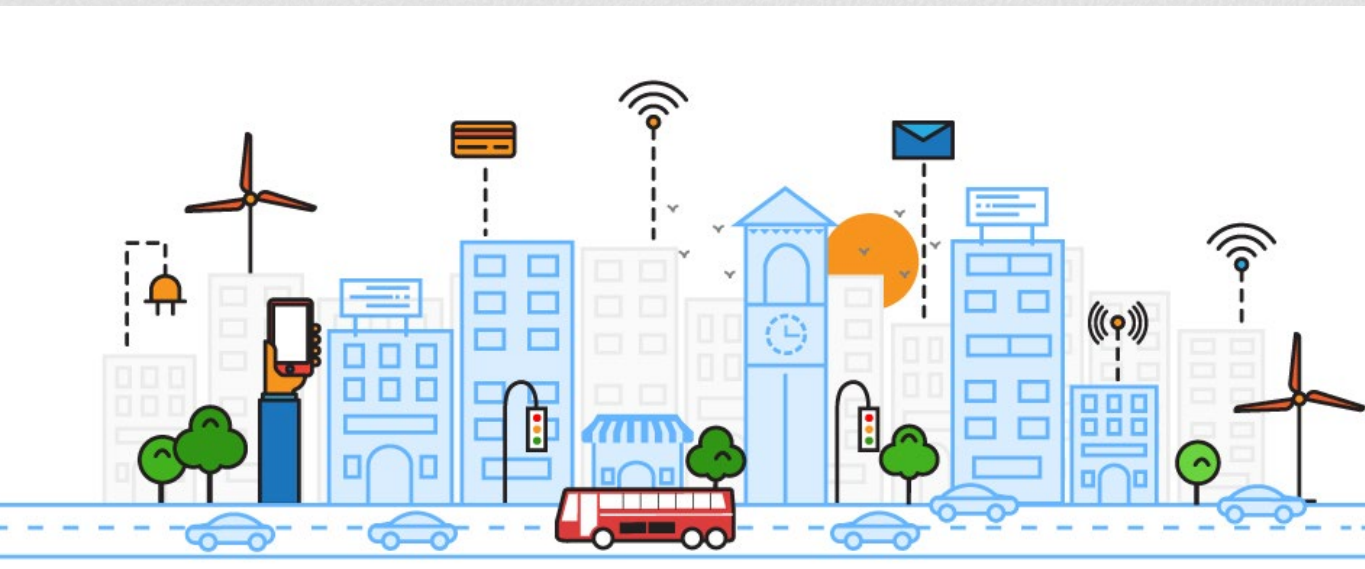
알고리즘	기능	예시
대칭키	암호화 (운영모드), 난수 생성기, 해시 연산	ARIA, LEA, AES
비대칭키	키교환, 전자 서명	ECC, RSA, PQC
해시	메시지 다이제스트, 난수 생성기	LSH, SHA- 2/ 3
난수 생성기	진짜 난수, 가짜 난수	HW- RNG, DRBG

OpenSSL
Cryptography and SSL/TLS Toolkit

arm
MBED

암호 구현?

- ▶ 수학적 안전성에 기반한 암호 알고리즘을 실생활에서 사용하는 다양한 사물인터넷 장비 상에서 실현시키는 작업
- ▶ 발전소, 신호등, 스마트폰, 신용카드, 스마트홈, 자동차, 스마트 공장



사물인터넷의 분류

고성능 사물



저전력 사물

대표적인 임베디드 프로세서 (ARM)

Cortex- M (Microcontroller): 저전력 설계

- ▶ IoT, 산업용 및 가정용 기기에 활용

Cortex- A (Application): 복잡한 컴퓨팅

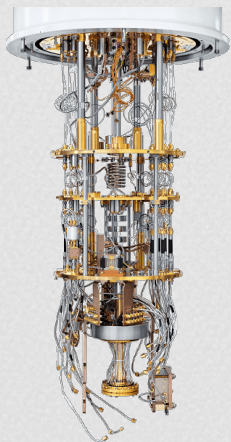
- ▶ 운영체제 플랫폼과 SW 어플리케이션 지원

Cortex- R (Real Time): 실시간성

- ▶ M 아키텍처 기반으로 설계된 제품; 실시간 및 안전성 특화



컴퓨터 구조에 따른 분류



GPU



CPU

Quantum Computer

비고	Quantum Computer	GPU	CPU
목적	양자 알고리즘 구동	병렬 연산 / 그래픽 연산	순차 연산 / 범용 연산
개발사	IBM, Google	Nvidia	ARM, AMD, Intel

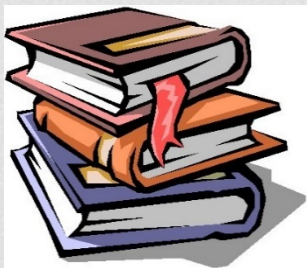
암호 구현을 유튜브 방송에 비유



유튜브	암호설계/ 구현	난이도
스토리	암호 알고리즘 개발 (몽고메리 곱셈기)	상
녹화	암호 구현 (프로그래밍)	중
편집	암호 구현 개선 (어셈, 병렬화)	중
웹페이지	컴퓨터 (Intel, ARM)	-

암호 구현 추진도

 레퍼런스 코드가 있는 경우



대략적인
동작 방식 이해

레퍼런스
코드 **분석**

레퍼런스 코드 기반
프로그램 **수정**

암호 구현 추진도



레퍼런스 코드는 어디에?

▶ KISA 암호알고리즘 소스코드

해시함수 SHA-3

전자서명 KCDSA, EC-KCDSA

모바일환경에 적합한 암호알고리즘 소스코드

해시함수 LSH

해시함수 SHA-256

블록암호 LEA

블록암호 ARIA

블록암호 HIGHT

블록암호 SEED

패스워드 안전성 검증 라이브러리

<https://seed.kisa.or.kr/kisa/reference/EgovSource.do>

암호 구현 추진도



레퍼런스 코드는 어디에?


▶ NIST 공모전 코드

Round 3 Finalists: Public-key Encryption and Key-establishment Algorithms

Algorithm	Algorithm Information	Submitters	Comments
Classic McEliece <i>(merger of Classic McEliece and NTS-KEM)</i>	Zip File (97MB) Website	Martin R. Albrecht Daniel J. Bernstein Tung Chou	Submit Comment View Comments

암호 구현 추진도



 Search or jump to... Pulls Issues Marketplace Explore

Learn Git and GitHub without any code!

Using the Hello World guide, you'll start a branch, write comments, and open a pull request.

Read the guide

kmackay / micro-ecc

Watch 81 Star 819 Fork 310

<> Code Issues 14 Pull requests 3 Actions Projects Wiki

master

Go to file Add file Code

kmackay Merge pull request #177 from jaroban/... yesterday 221

examples/ecc_t...	Fix for #148	25 days ago
scripts	Add fast multiply asm for AVR (#50)	5 years ago
test	Add test vectors for public key gener...	20 days ago
.gitignore	Adding ignore rules.	7 years ago
LICENSE.txt	Add license.	7 years ago
README.md	Update README.md	3 years ago
asm_arm.inc	Improve Thumb multiplication perfor...	5 years ago
asm_arm_mult_...	Add faster ARM multiplication code ...	5 years ago

ECDSA and ECDH for 8-bit, 32-bit, and 64-bit processors.

Readme

BSD-2-Clause License

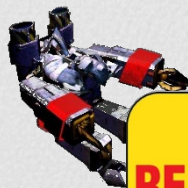
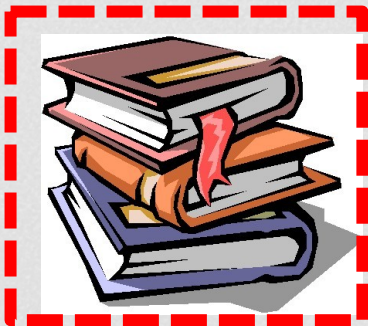
Releases 1

v1.0 Latest on 23 May 2017

Packages

암호 구현 추진도

📌 레퍼런스 코드가 없는 경우



REFERENCE



동작 방식

레퍼런스 코드

레퍼런스 코드 기반

완벽히 이해

작성

프로그램 수정

암호 구현의 특징



입력 중 1-비트가 어긋나면 출력 전체가 달라지는 특징

- ▶ 디버깅이 용이 (출력값 확인) 하다는 장점 (?)을 가짐

SHA256 예시

입력: 0

출력: 0x5feceb66ffc86f38d952786c6d696c79c\
2dbc239dd4e91b46729d73a27fb57e9

입력: 1

출력: 0x6b86b273ff34fce19d6b804eff5a3f574\
7ada4eaa22f1d49c01e52ddb7875b4b

암호 구현 완결성 확인 방법



테스트 벡터 활용

- ▶ 암호에 대한 무작위 입력과 정확한 출력의 묶음 (문제 & 답)

KEY = 00000000000000000000000000000000

IV or PLAINTEXT = 00000000000000000000000000000000

PLAINTEXT or IV	CIPHERTEXT
f34481ec3cc627bacd5dc3fb08f273e6	0336763e966d92595a567cc9ce537f5e
9798c4640bad75c7c3227db910174e72	a9a1631bf4996954ebc093957b234589
96ab5c2ff612d9dfaae8c31f30c42168	ff4f8391a6a40ca5b25d23bedd44a597
6a118a874519e64e9963798a503f1d35	dc43be40be0e53712f7e2bf5ca707209
cb9fceec81286ca3e989bd979b0cb284	92beedab1895a94faa69b632e5cc47ce
b26aeb1874e47ca8358ff22378f09144	459264f4798f6a78bacb89c15ed3d601
58c8e00b2631686d54eab84b91f0aca1	08a4e2efec8a8e3312ca7460b9040bbf

데이터 표현 주의

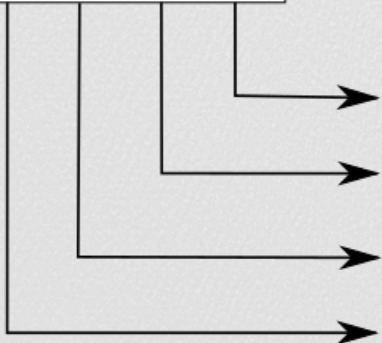
Little-endian 그리고 Big-endian

- ▶ 컴퓨터 구조에 따라 상이한 엔디안을 가짐

Little-endian

32-bit integer

0A0B0C0D



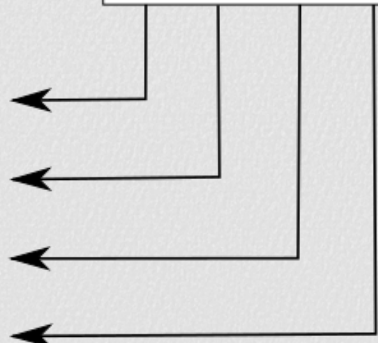
Memory

⋮		⋮
0D	a	0A
0C	$a+1$	0B
0B	$a+2$	0C
0A	$a+3$	0D
⋮		⋮

Big-endian

32-bit integer

0A0B0C0D



암호 구현 완결성 확인 방법



수학 계산기로 테스트 벡터 생성

MAGMA
COMPUTER • ALGEBRA

AboutCalculatorOrderingFAQDownloadDocumentationCitationsConferencesLinksContactCAGLogin

Magma Calculator

Enter your code in the box below. Click on "Submit" to have it evaluated by Magma.

```
SetSeed(1);  
a:=Random(2^128,2^129);  
c:=a+a;  
  
c:Hex;
```

ClearSubmit

0x2101481327CA4257C9CA7063282EB506208081A5A85A8014FE46830C2C06B5FB9

Calculations are restricted to 120 seconds.
Input is limited to 50000 bytes.
Running Magma V2.23-3.
Seed: 1554197083; Total time: 0.190 seconds; Total memory usage: 32.09MB.

암호 구현 완결성 확인 방법



레퍼런스 코드와 1대1 비교


```
// Diffusion Layer
void DL(const Byte *i, Byte *o) {
    Byte T;

    T = i[3] ^ i[4] ^ i[9] ^ i[14];
    o[0] = i[6] ^ i[8] ^ i[13] ^ T;
    o[5] = i[1] ^ i[10] ^ i[15] ^ T;
    o[11] = i[2] ^ i[7] ^ i[12] ^ T;
    o[14] = i[0] ^ i[5] ^ i[11] ^ T;
    T = i[2] ^ i[5] ^ i[8] ^ i[15];
    o[1] = i[7] ^ i[9] ^ i[12] ^ T;
    o[4] = i[0] ^ i[11] ^ i[14] ^ T;
    o[10] = i[3] ^ i[6] ^ i[13] ^ T;
    o[15] = i[1] ^ i[4] ^ i[10] ^ T;
    T = i[1] ^ i[6] ^ i[11] ^ i[12];
    o[2] = i[4] ^ i[10] ^ i[15] ^ T;
    o[7] = i[3] ^ i[8] ^ i[13] ^ T;
    o[9] = i[0] ^ i[5] ^ i[14] ^ T;
    o[12] = i[2] ^ i[7] ^ i[9] ^ T;
    T = i[0] ^ i[7] ^ i[10] ^ i[13];
    o[3] = i[5] ^ i[11] ^ i[14] ^ T;
    o[6] = i[2] ^ i[9] ^ i[12] ^ T;
    o[8] = i[1] ^ i[4] ^ i[15] ^ T;
    o[13] = i[3] ^ i[6] ^ i[8] ^ T;
}

// Right-rotate 128 bit source string s by n bits and XOR it to target string t
void RotXOR(const Byte *s, int n, Byte *t) {
    int i, q;

    q = n / 8;
    n %= 8;
    for (i = 0; i < 16; i++) {
        t[(q + i) % 16] ^= (s[i] >> n);
        if (n != 0)
            t[(q + i + 1) % 16] ^= (s[i] << (8 - n));
    }
}
```

Google 학술검색



☒ 모든 언어 ☐ 한국어 웹

COVID-19에 관한 기사

CDC

NEJM

JAMA

Lancet

Cell

BMJ

Nature

Science

Elsevier

Oxford

Wiley

medRxiv

거인의 어깨에 올라서서 더 넓은 세상을 바라보라 - 아이작 뉴턴

암호 구현의 최고의 스승 (gcc)

gcc - S helloworld.c

▶ gcc가 변환한 어셈블리 분석

```
push    %rbp
mov     %rsp,%rbp
sub     $0x10,%rsp
lea     0x51(%rip),%rdi          # 0x100000f60
movl    $0x0,-0x4(%rbp)
mov     $0x0,%al
callq   0x100000f34 <dyld_stub_printf>
mov     $0x0,%ecx
mov     %eax,-0x8(%rbp)
mov     %ecx,%eax
add     $0x10,%rsp
pop     %rbp
retq
```

```
#include <stdio.h>

int main()
{
    printf("Hello world!");
    return 0;
}
```


암호 구현 주안점 I



암호화 연산은 컴퓨터 상의 자원을 소모하며 수행

- ▶ 전력 (실행시간), 저장 공간 (RAM, ROM)



동일한 암호도 구현 기법에 따라 상이한 연산 복잡도 도출

- ▶ 알고리즘 변환, 최신 컴퓨터 구조 활용 (SIMD, SIMT)



일반적으로 컴퓨터 자원들은 상호 **trade-off** 관계를 가짐

- ▶ 세가지 컴퓨터 자원을 모두 최적화하는 암호 구현은 어려움

→ 목표로 하는 상황에 맞추어 최적화 자원 선정 필요

타겟에 따른 암호 구현 (고성능 사물 VS 저전력 사물) I



Product	Processor	Frequency	RAM / ROM	OS
iPhone 13 Pro	64-bit ARMv8-A	3.22GHz	6GB/ 1TB	iOS
Airpod	32-bit ARMv6-M	100MHz	128KB/ 1MB	RTOS

타겟에 따른 암호 구현 (고성능 사물 VS 저전력 사물) II

연산 속도:

암호화 연산 가속화 (고성능/저전력)

코드 크기:

암호화 코드 최소화 (저전력)

램 사용량:

변수 및 사전 테이블 최소화 (저전력)

부채널 공격 방어:

Branch/Cache-free (고성능/저전력)



암호 구현과 진로

암호 구현 관련 능력

- ▶ 프로그래밍 능력 (어셈블리), 컴퓨터 구조, 암호학적 지식 (수학)

진로

- ▶ 일반 IT 기업, 연구소

