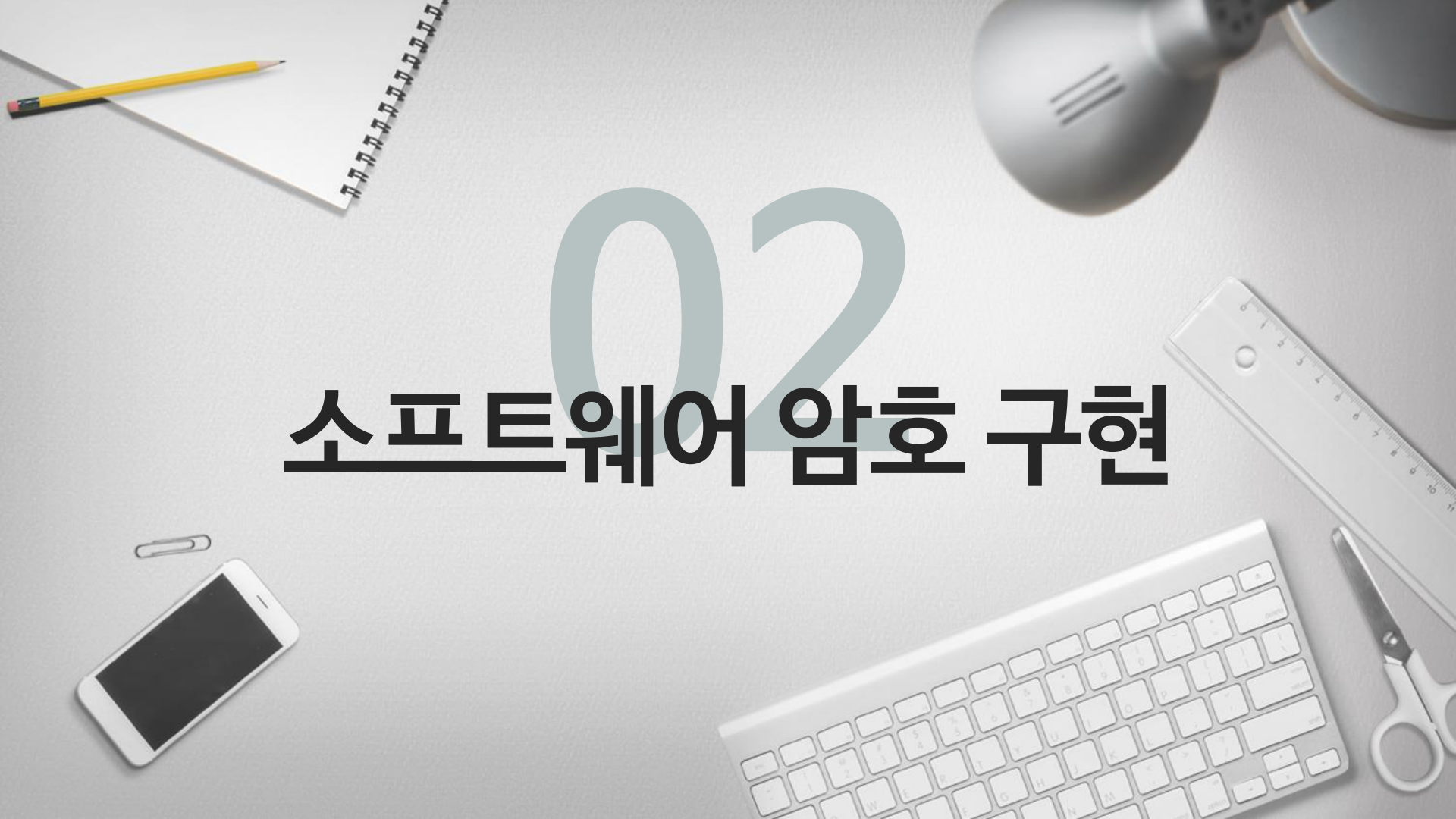


02

# 소프트웨어 암호 구현



# 소프트웨어 vs 하드웨어



# 소프트웨어 기반 암호 구현

특징	내용
플랫폼	저전력, 고성능
구현 중점	속도, 코드 크기, RAM 크기
확장성	Python, Java, C, 어셈블리
병렬 처리	SIMD, SIMT, Superscalar, Warp
Crypto 가속기	AES, SHA, ECC, RNG
부채널 방지	Simple Power Analysis, Timing Attack, Fault Attack
TEE	Intel SGX, ARM Trustzone

# 암호 구현 첫걸음

암호 구조

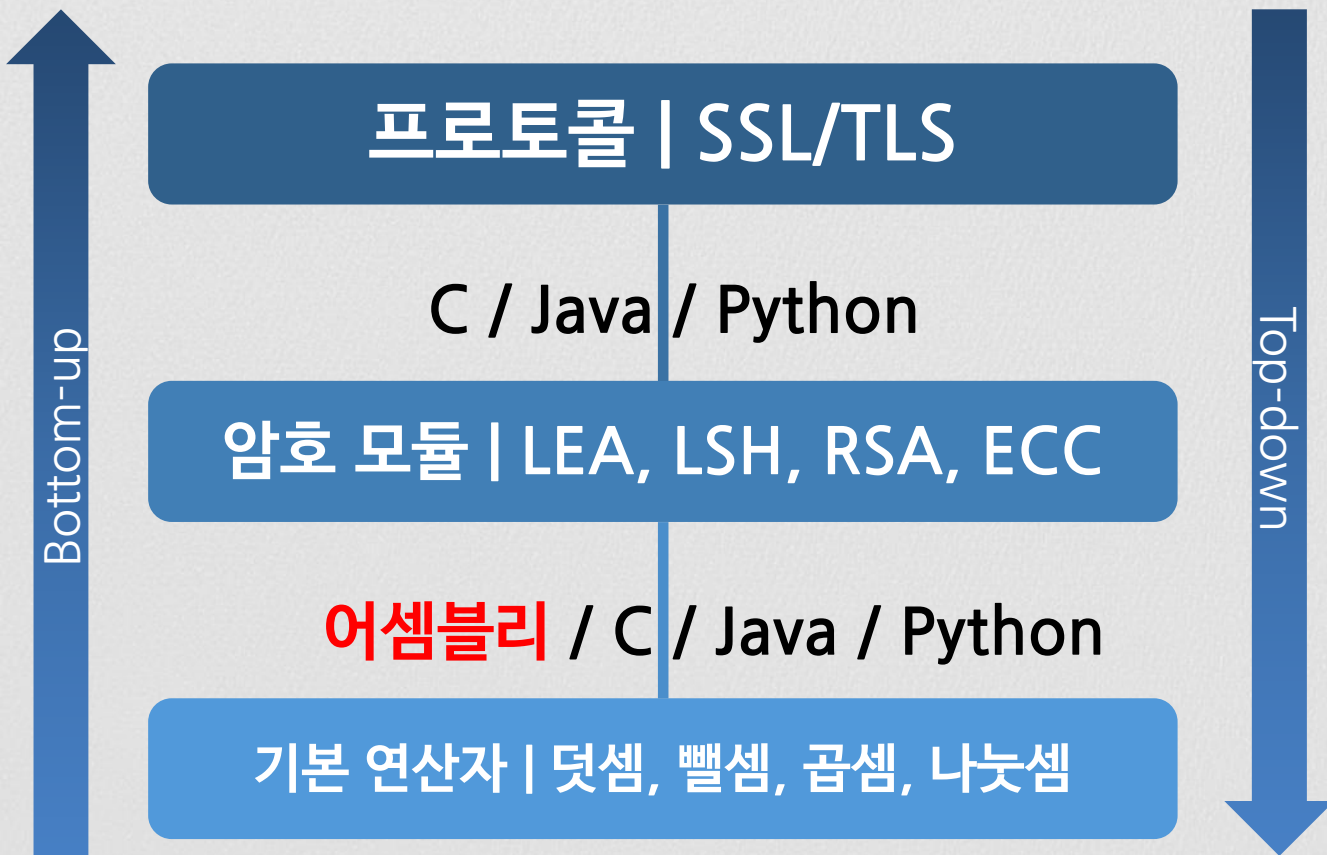
프로그래밍 언어

플랫폼

벤치 마크

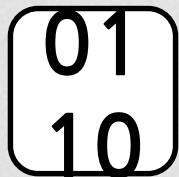


# 암호 구조



# 프로그래밍 언어

저급 언어



고급 언어

# 프로그래밍 언어 (64-비트 곱셈 연산 비교)

```
void digit_x_digit(const digit_t a, const digit_t b, digit_t* c)
{ // Digit multiplication, digit * digit -> 2-digit result
    register digit_t al, ah, bl, bh, temp;
    digit_t albl, albh, ahbl, ahbh, res1, res2, res3, carry;
    digit_t mask_low = (digit_t)(-1) >> (sizeof(digit_t)*4), mask_high = (digit_t)(-1) << (sizeof(digit_t)*4);

    al = a & mask_low;           // Low part
    ah = a >> (sizeof(digit_t) * 4); // High part
    bl = b & mask_low;
    bh = b >> (sizeof(digit_t) * 4);

    albl = al*bl;
    albh = al*bh;
    ahbl = ah*bl;
    ahbh = ah*bh;
    c[0] = albl & mask_low;       // C00

    res1 = albl >> (sizeof(digit_t) * 4);
    res2 = ahbl & mask_low;
    res3 = albh & mask_low;
    temp = res1 + res2 + res3;
    carry = temp >> (sizeof(digit_t) * 4);
    c[0] ^= temp << (sizeof(digit_t) * 4); // C01

    res1 = ahbl >> (sizeof(digit_t) * 4);
    res2 = albh >> (sizeof(digit_t) * 4);
    res3 = ahbh & mask_low;
    temp = res1 + res2 + res3 + carry;
    c[1] = temp & mask_low;       // C10
    carry = temp & mask_high;
    c[1] ^= (ahbh & mask_high) + carry; // C11
}
```

C언어

mul C0, A, B  
umulh C1, A, B

어셈블리

- 지원하는 명령어 셋
  - SISD, SIMD, Integer, Floating, Crypto
- 레지스터의 수
  - 일반 용도, 특수 용도
- 파이프라이닝
  - 파이프라이닝 구조
- 메모리 크기
  - ROM과 RAM 크기

## Cortex-M4 [\[ edit \]](#)

Conceptually the Cortex-M4 is a Cortex-M3 plus **DSP** instructions, and optional floating-point unit (FPU). A core with an FPU is known as Cortex-M4F.

Key features of the Cortex-M4 core are:<sup>[6]</sup>

- ARMv7E-M architecture<sup>[10]</sup>
- 3-stage pipeline with branch speculation.
- Instruction sets:
  - Thumb-1 (entire).
  - Thumb-2 (entire).
  - 32-bit hardware integer multiply with 32-bit or 64-bit result, signed or unsigned, add or subtract after the multiply. 32-bit Multiply and MAC are 1 cycle.
  - 32-bit hardware integer divide (2–12 cycles).
  - Saturation arithmetic support.
  - DSP extension: Single cycle 16/32-bit MAC, single cycle dual 16-bit MAC, 8/16-bit SIMD arithmetic.
- 1 to 240 interrupts, plus NMI.
- 12 cycle interrupt latency.
- Integrated sleep modes.

### Silicon options:

- Optional floating-point unit (FPU): single-precision only IEEE-754 compliant. It is called the FPv4-SP extension.
- Optional memory protection unit (MPU): 0 or 8 regions.

## Cortex-M4

### Architecture and classification

Microarchitecture	ARMv7E-M
Instruction set	Thumb-1, Thumb-2, Saturated, DSP, Divide, FPU (SP)



Silicon Labs (Energy Micro)  
Wonder Gecko STK Board with  
EFM32WG990



TI Stellaris Launchpad  
Board with LM4F120



# 플랫폼 간 특징 분석

비고	Cortex-M3	Cortex-M4
Architecture	ARMv7-M (Harvard)	ARMv7E-M (Harvard)
Instruction Set	Thumb, Thumb-2	Thumb, Thumb-2, DSP, SIMD, FPU
Pipeline Stage	3	3
Instruction/Data Cache	X	X
Interrupt Priority	8-256	8-256
Single Cycle Multiply	부분적으로 제공	전체 제공
Hardware Divide	O	O
발표일자	2004	2010



**Arduino DUE**  
(84Mhz, 512KB ROM,  
96KB RAM)



**STM32F4DISCOVERY**  
(168Mhz, 1MB ROM,  
192KB RAM)

# 플랫폼

## 8/16/32/64-비트 프로세서들의 차이점

▶ 레지스터의 저장 단위

8-bit

16-bit

32-bit

64-bit

128-bit / 256-bit / 512-bit?

## 8/16/32/64-비트 프로세서들의 차이점

▶ 예시) 64-비트 덧셈

프로세서	AVR	MSP	ARMv6/7	ARMv8
연산 가능 최대 길이	8-비트	16-비트	32-비트	64-비트
연산 횟수	8 번	4번	2번	1번

# 벤치마크



## 벤치마크의 목적

- ▶ 특정 프로그램 (암호)를 객관적인 지표 (구현 환경 및 기법)에 따라 성능 분석



## 벤치마크 플랫폼

- ▶ 객관성 증진과 테스트 용이성을 위해 개발



## 대표적인 벤치마크


벤치마크	SUPERCOP	FELICS	pqm4	OpenSSL
플랫폼	Intel, ARM	AVR, MSP430, ARM M3	ARM M4	Intel, ARM
암호	모든 암호	블록 암호	양자 내성 암호	모든 암호



# SUPERCOP

## 유럽 ECRYPT 프로젝트의 일환으로 개발

- ▶ 현재 가장 유명한 암호 벤치 마크 프레임워크



**eBACS: ECRYPT Benchmarking of  
Cryptographic Systems**

**ECRYPT II**  
↓↑↔⊗⊕⊖⊗⊕⊖⊗⊕⊖

[ECRYPT II](#)

<b>General information:</b>		<a href="#">Introduction</a>	<a href="#">eBASH</a>	<a href="#">eBASC</a>	<a href="#">eBAEAD</a>	<a href="#">eBATS</a>	<b>SUPERCOP</b>	<a href="#">XBX</a>	<a href="#">Computers</a>
<b>How to submit new software:</b>	<a href="#">Tips</a>	<a href="#">hash</a>	<a href="#">stream</a>	<a href="#">aead</a>		<a href="#">dh</a>	<a href="#">kem</a>	<a href="#">encrypt</a>	<a href="#">sign</a>
	<b>List of primitives measured:</b>	<a href="#">sha3</a>	<a href="#">hash</a>	<a href="#">stream</a>	<a href="#">caesar</a>	<a href="#">aead</a>	<a href="#">dh</a>	<a href="#">kem</a>	<a href="#">encrypt</a>
<b>Measurements indexed by machine:</b>	<a href="#">sha3</a>	<a href="#">hash</a>	<a href="#">stream</a>	<a href="#">caesar</a>	<a href="#">aead</a>	<a href="#">dh</a>	<a href="#">kem</a>	<a href="#">encrypt</a>	<a href="#">sign</a>
<b>List of subroutines:</b>	<a href="#">verify</a>	<a href="#">decode</a>	<a href="#">encode</a>	<a href="#">sort</a>	<a href="#">core</a>	<a href="#">hashblocks</a>	<a href="#">scalarmult</a>		

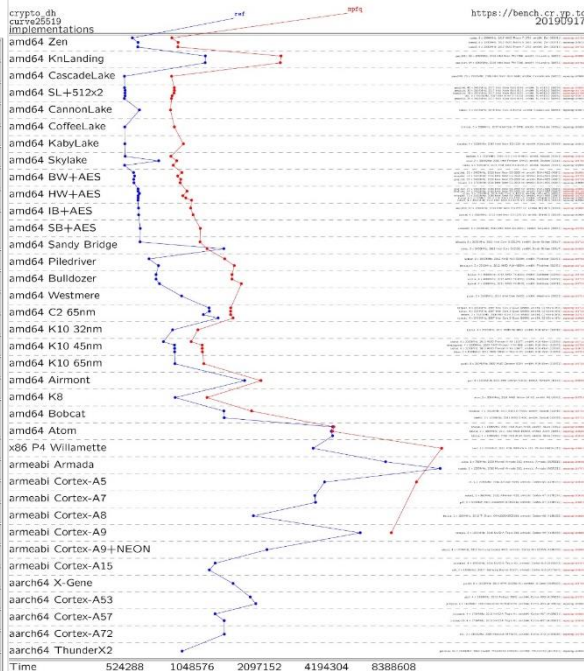
# SUPERCOP



## 특정 암호에 대해 다양한 플랫폼 상의 성능 지표 도출 가능

### Architecture notes

Designer	Microarchitecture	CPU ID
AMD	K7	622
AMD	K8	20f10 (Egypt), 40f13 (Santa Rosa), 40fb2
AMD	K10 65nm	100f23 (Agena, Barcelona)
AMD	K10 45nm	100f42 (Deneb, Shanghai), 100f63 (Geneva), 100fa0 (Thuban)
AMD	K10 32nm	300f10 (Llano)
AMD	Bobcat	500f20 (Brazos)
AMD	Bulldozer	600f12 (Zambezi)
ARM	Cortex-A8	412fc081 (Freescall i.MX515), 412fc085 (Freescall i.MX515), 413fc082 (TI Sitara AM3359, TI Sitara AM3703)
ARM	Cortex-A9	411fc090 (NVIDIA Tegra 250)
ARM	Cortex-A9+NEON	413fc090 (VIA WonderMedia 8850)
ARM	Cortex-A15	410fc0f4 (Samsung Exynos 5)
Intel	P6	672, 683
Intel	PM	6d8
Intel	P4 NetBurst	f0a (Willamette), f12 (Willamette), f25 (Northwood)
Intel	P4 Prescott	f41, f43, f64
Intel	C2 65nm	6f2 (Allendale, Conroe), 6f6 (Allendale, Conroe, Woodcrest), 6f8 (Kenstfield), 6fb (Kentsfield, Merom, Clovertown, Tigerton), 6fd (Allendale, Conroe)
Intel	C2 45nm	10676 (Harpertown, Wolfdale), 10677 (Yorkfield), 1067a (Penryn, Wolfdale)
Intel	Atom	106c2 (Diamondville), 106ca (Pineview)
Intel	Nehalem	106a5 (Bloomfield, Gainestown), 106a5 (Clarksfield, Lynnfield), 206e6 (Beckton)
Intel	Westmere	20652 (Arrandale), 20655 (Arrandale)
Intel	Westmere+AES	206c2 (Gulftown)
Intel	Sandy Bridge	206a7 in Core i3
Intel	SB+AES	206a7 in Core i5/i7
Intel	Haswell	306c3 in Core i3
Intel	HW+AES	306c3 in Core i5/i7
Qualcomm	Scorpion	510f02d2 (Snapdragon S3 APQ8060)
Qualcomm	Krait	511f04d0, 511f04d3 (Snapdragon S4 APQ8060A)



## 룩셈부르크 대학에서 시행한 블록암호 구현 공모전

- ▶ NIST 경량 암호 공모전에도 큰 영향력을 미치고 있음

## 저전력 프로세서를 타겟으로 시행

- ▶ 8-비트 AVR, 16-비트 MSP, 32-비트 ARM M3

## 평가 기준

- ▶ 코드 크기, RAM, 실행속도



## 지금까지 평가의 객관성 문제

- ▶ 서로 상이한 플랫폼 상에서의 구현 결과
- ▶ 8-비트 플랫폼에 매우 적합한 암호 구현 vs 모든 플랫폼에 골고루 좋은 성능인 암호 구현
- ▶ 연산 속도만 빠른 구현 vs 모든 성능이 골고루 좋은 암호
- ▶ 특정한 프로토콜에 특화된 암호 vs 모든 프로토콜에 골고루 좋은 암호





## 평가 기준의 객관성 확보 → Figure-of-Merit (FOM)

- ▶ 코드 크기, RAM, 실행속도, 그리고 플랫폼까지 고려한 평가 기준



## 다양한 프로토콜에 대한 평가 → 시나리오 기반 평가

- ▶ Communication Protocol  
(CBC mode of operation: 1,024 bits data)
- ▶ Challenge-Response Authentication  
(CTR mode of operation: 128 bits data)



$$FOM(i_1, i_2, i_3) = \frac{p_{i1}, AVR + p_{i2}, MSP + p_{i3}, ARM}{3}$$

심볼	설명
$M$	실행 속도, 메모리 사용량, 코드 크기
$i$	구현물
$d$	플랫폼
$p_{i,d}$	Performance indicator
$w_m$	평가지표의 상대적 가중치

$$p_{i,d} = \sum_{m \in M} w_m \frac{v_{i,d,m}}{\min_i(v_{i,d,m})}$$

심볼	설명
$m$	평가지표
$v_{i,d,m}$	$i$ 암호 구현을 $d$ 플랫폼에 구현했을 때 $m$ 평가지표의 값
$\min_i(v_{i,d,m})$	전체 $i$ 암호 구현에 대한 $v_{i,d,m}$ 최소값

# FELICS

Results for scenario 1 - I: Encryption + Decryption (including key schedule). Encrypt 128 bytes of data using CBC mode. For each cipher, an optimal implementation on each architecture is selected.

Cipher Info				Results									
				AVR			MSP			ARM			
Cipher	Block [b]	Key [b]	Sec.	Code [B]	RAM [B]	Time [cyc.]	Code [B]	RAM [B]	Time [cyc.]	Code [B]	RAM [B]	Time [cyc.]	
Chaskey	128	128	0.87	1318	227	21349	900	222	19058	450	236	8740	4.0
Chaskey-LTS	128	128	0.43	1318	227	33829	904	222	29170	450	236	11556	4.6
Speck	64	96	0.73	956	292	40666	576	290	36698	328	304	16100	5.1
Speck	64	128	0.74	864	300	45686	592	298	37850	402	312	17084	5.3
Simon	64	96	0.71	1074	361	64440	758	362	50932	466	376	24591	7.0
Simon	64	128	0.70	1112	373	67404	780	374	53112	530	388	23404	7.2
RECTANGLE	64	128	0.72	1108	351	65604	846	406	54564	627	432	36313	8.2
RECTANGLE	64	80	0.72	1142	350	67513	832	400	54431	631	426	37336	8.3
LEA	128	128	0.63	1650	629	61755	1154	630	51582	496	664	17410	8.3
SPARX	64	128	0.62	1188	390	66330	986	394	43437	1158	424	39680	8.9
SPARX	128	128	0.68	1726	751	84390	1118	760	67042	1118	788	67430	13.5
SKINNY	64	128	-1	1086	337	92891	1006	342	112402	788	372	95924	13.9
HIGHT	64	128	0.81	1404	331	95348	1258	330	129188	1440	380	89382	14.8
SKINNY	128	128	-1	1124	545	77451	1158	546	142974	866	572	95658	15.5
AES	128	128	0.70	3000	406	58973	2684	408	87850	3052	452	72828	15.7
Fantomas	128	128	-1	3524	229	141269	4164	234	57430	2926	280	95629	18.0
Robin	128	128	1	2484	229	183957	3170	238	76878	3700	1292	66398	18.5
Lilliput	64	80	-1	1884	474	82138	1940	470	132945	2322	572	117500	18.8
Robin*	128	128	-1	4458	239	153685	3312	238	88838	3864	1296	80046	20.2
RoadRunneR	64	128	0.58	2176	206	131430	3218	218	224521	2344	424	122499	21.8
RC5-20	64	128	0.80	3630	363	252858	4094	372	409826	568	384	30948	22.0
RoadRunneR	64	80	0.5	2200	329	152373	3088	338	252491	2584	394	103621	22.1
PRIDE	64	128	-1	2128	366	125036	2566	212	282041	1280	428	168659	24.8
LBlock	64	80	0.72	2666	480	196797	1632	324	316539	2232	534	160926	27.5
PRESENT	64	80	0.84	2150	446	246023	1838	450	205430	2116	478	274107	32.9
PRINCE	64	128	0.83	1930	365	300799	2028	236	451509	1660	444	232538	35.8
TWINE	64	80	0.64	3724	615	273413	3796	564	393320	2563	469	228923	37.5
Piccolo	64	80	0.56	1936	314	412551	1354	310	354195	1212	382	335247	41.2
LED	64	80	-1	4882	560	2213031	7004	252	2505640	3908	638	580575	144.7

## NIST PQC의 주요 평가 요소

- ▶ 암호 보안성, 연산 성능

## pqm4

- ▶ Cortex-M4 상에서의 성능 평가
- ▶ 연산 속도, ROM, 그리고 RAM 사용량 평가

### What NIST wants

- Performance (hardware+software) will play more of a role
  - More benchmarks
  - For hardware, NIST asks to focus on **Cortex M4** (with all options) and **Artix-7**
    - pqc-hardware-forum
- Continued research and analysis on **ALL** of the 2<sup>nd</sup> round candidates
- See how submissions fit into applications/procotols. Any constraints?







## 가장 유명한 암호 라이브러리

### ▶ 자체 벤치마크 제공

```
[2.3.2-RELEASE][admin@vpnpraca.praca]/root: /usr/bin/openssl speed -evp aes-128-cbc -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing aes-128-cbc for 3s on 16 size blocks: 69022 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 64 size blocks: 64313 aes-128-cbc's in 3.05s
Doing aes-128-cbc for 3s on 256 size blocks: 63053 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 1024 size blocks: 61382 aes-128-cbc's in 3.02s
Doing aes-128-cbc for 3s on 8192 size blocks: 41874 aes-128-cbc's in 3.00s
OpenSSL 1.0.1s-freebsd 1 Mar 2016
built on: date not available
options:bn(64,64) rc4(8x,int) des(idx,cisc,16,int) aes(partial) idea(int) blowfish(idx)
compiler: clang
The 'numbers' are in 1000s of bytes per second processed.
type            16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
aes-128-cbc      368.12k    1350.90k    5380.52k    20843.16k    114343.94k
```

AES-NI disabled

```
[2.3.2-RELEASE][admin@vpnpraca.praca]/root: /usr/bin/openssl speed -evp aes-128-cbc -elapsed
You have chosen to measure elapsed time instead of user CPU time.
Doing aes-128-cbc for 3s on 16 size blocks: 35851979 aes-128-cbc's in 3.03s
Doing aes-128-cbc for 3s on 64 size blocks: 12119352 aes-128-cbc's in 3.00s
Doing aes-128-cbc for 3s on 256 size blocks: 3722287 aes-128-cbc's in 3.05s
Doing aes-128-cbc for 3s on 1024 size blocks: 1030609 aes-128-cbc's in 3.01s
Doing aes-128-cbc for 3s on 8192 size blocks: 118720 aes-128-cbc's in 3.00s
OpenSSL 1.0.1s-freebsd 1 Mar 2016
built on: date not available
options:bn(64,64) rc4(8x,int) des(idx,cisc,16,int) aes(partial) idea(int) blowfish(idx)
compiler: clang
The 'numbers' are in 1000s of bytes per second processed.
type            16 bytes    64 bytes    256 bytes    1024 bytes    8192 bytes
aes-128-cbc      189239.31k    258546.18k    312748.46k    350867.49k    324184.75k
```

AES-NI enabled

# 성능 측정 (Intel / ARM)

```
int64_t cpucycles(void)
{ // Access system counter for benchmarking
#ifdef (OS_TARGET == OS_WIN) && (TARGET == TARGET_AMD64 || TARGET == TARGET_x86)
    return __rdtsc();
#elif (OS_TARGET == OS_WIN) && (TARGET == TARGET_ARM)
    return __rdpmccntr64();
#elif (OS_TARGET == OS_LINUX) && (TARGET == TARGET_AMD64 || TARGET == TARGET_x86)
    unsigned int hi, lo;

    asm volatile ("rdtsc\n\t" : "=a" (lo), "=d"(hi));
    return ((int64_t)lo) | (((int64_t)hi) << 32);
#elif (OS_TARGET == OS_LINUX) && (TARGET == TARGET_ARM || TARGET == TARGET_ARM64)
    struct timespec time;

    clock_gettime(CLOCK_REALTIME, &time);
    return (int64_t)(time.tv_sec*1e9 + time.tv_nsec);
#else
    return 0;
#endif
}
```

# 성능 측정 (Intel / ARM)

```
cycles = 0;

for (n=0; n<BENCH_LOOPS; n++)
{
    cycles1 = cpucycles();

    // 수행하고자 하는 연산 삽입

    cycles2 = cpucycles();
    cycles = cycles+(cycles2-cycles1);
}

cycles = cycles/BENCH_LOOPS;
```

성능 측정 예제

# 성능 테스트 유의 사항



## 구현 완결성

- ▶ 벤치마크 시 구현 완결성 확인을 함께 수행해야 함
- ▶ 그렇지 않을 경우 불필요한 연산으로 간주
- ▶ 구성 (결과값 확인 코드 → 벤치마크 코드)



## 최적화 옵션

- ▶ -O3와 같은 경우 구현 완결성 실패 가능성 높음
- ▶ -O2가 가장 안정적인 최적화 옵션
- ▶ 어셈블리인 경우 최적화 옵션은 의미 없음



# 성능 테스트 유의 사항

## 주파수

- ▶ 연산 주파수에 따라 성능 변화 → 메모리와 CPU 속도 차이로 인한 지연
- ▶ 스마트폰 벤치마크 시 주의 사항 → 충전하여 테스트

## 시뮬레이터와 실제 디바이스

- ▶ 무조건 실제 디바이스가 정확한 성능 지표를 도출

## 연산 수행 횟수

- ▶ 연산 횟수는 많을 수록 안정적 → function call과 루프 오버헤드 무시 가능