MD5

https://youtu.be/1Q_-IPHdUcE

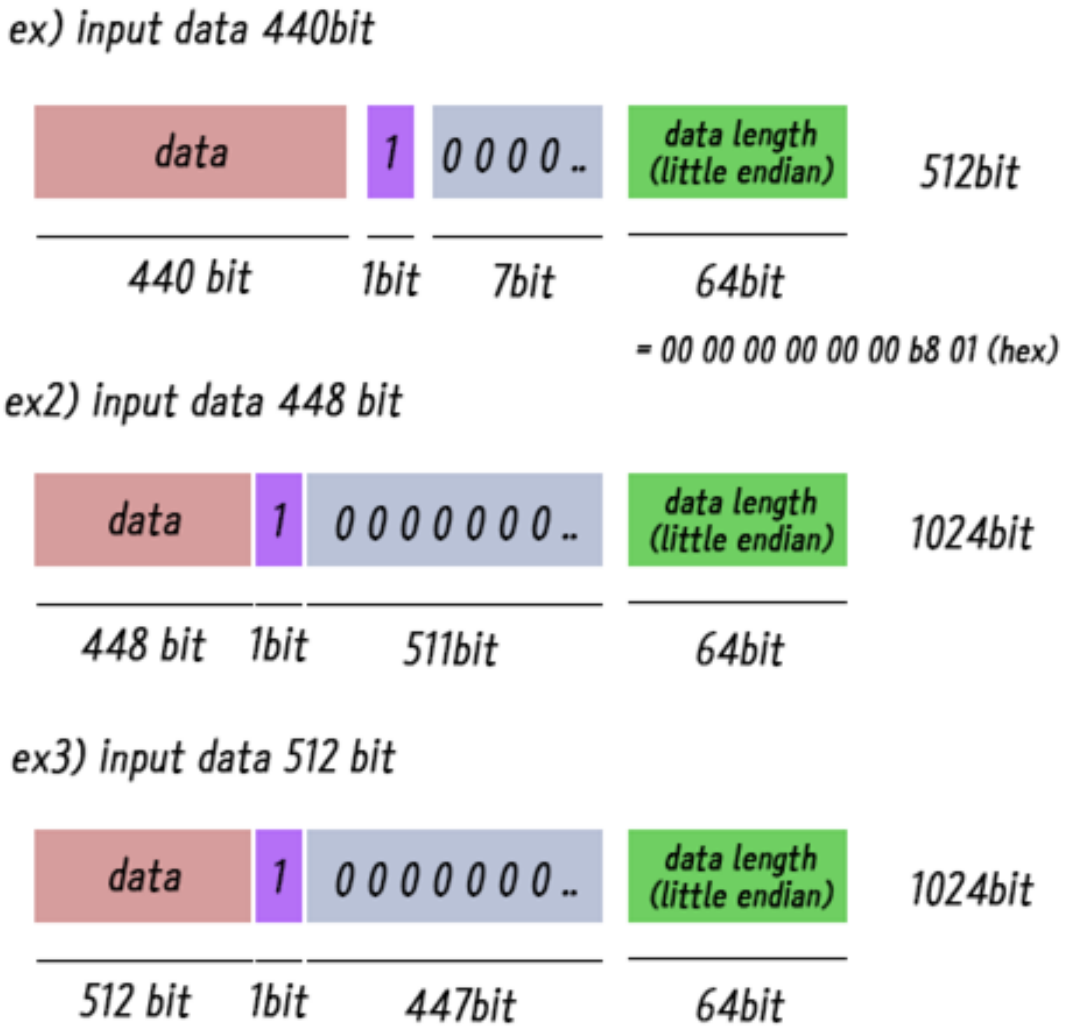
MD5

128비트 암호화 해시 함수

MD5는 임의의 길이의 메시지 입력
128비트짜리 고정 길이의 출력

입력 메시지는 512 비트 블록들로 쪼개짐
메시지를 우선 패딩하여
512로 나누어떨어질 수 있는 길이가 되록

첫 단일 비트, 1을 메시지 끝부분에 추가
512의 배수의 길이보다 64 비트가 적은 곳까지 0
64 비트는 메시지의 길이를 나타내는 64 비트 정수



MD5

32 비트 워드 네 개로 이루어진 하나의 128 비트 스테이트(state)에 대해 동작

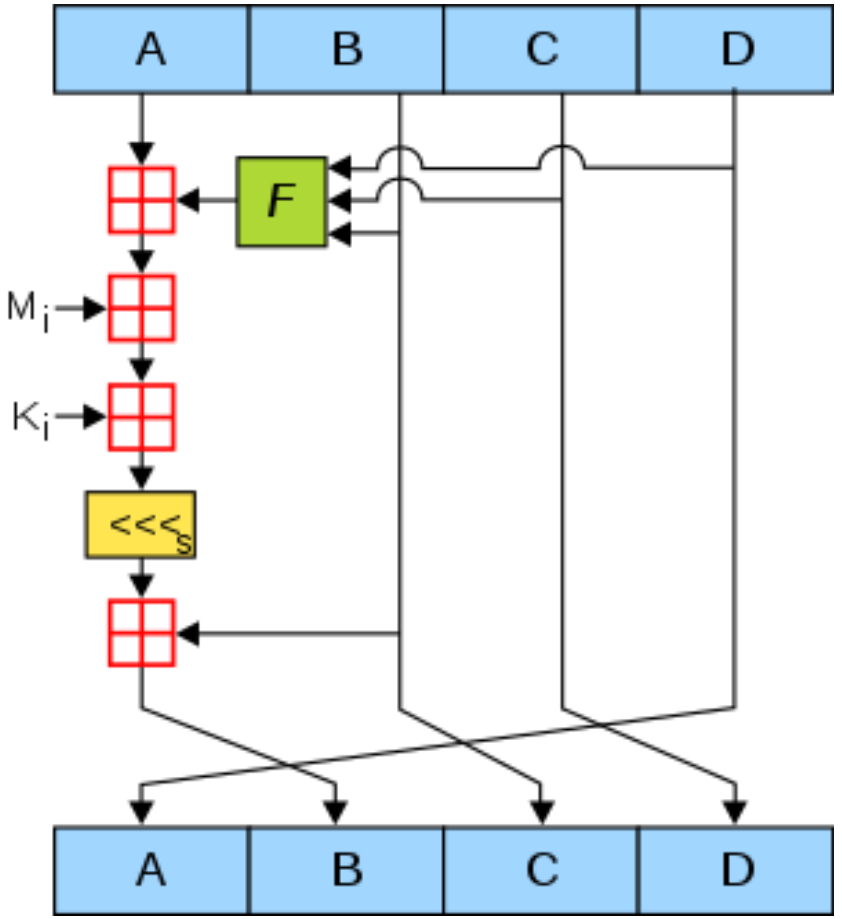
A,B,C,D는 소정의 상수값으로 초기화

메인 MD5 알고리즘은 각각의

512 비트짜리 입력 메시지 블록에 대해 차례로 동작

함수 F는 4가지

각 라운드마다 각각 다른 F가 쓰인다



$$F(B,C,D) = (B \wedge C) \vee (\neg B \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \neg D)$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \neg D)$$

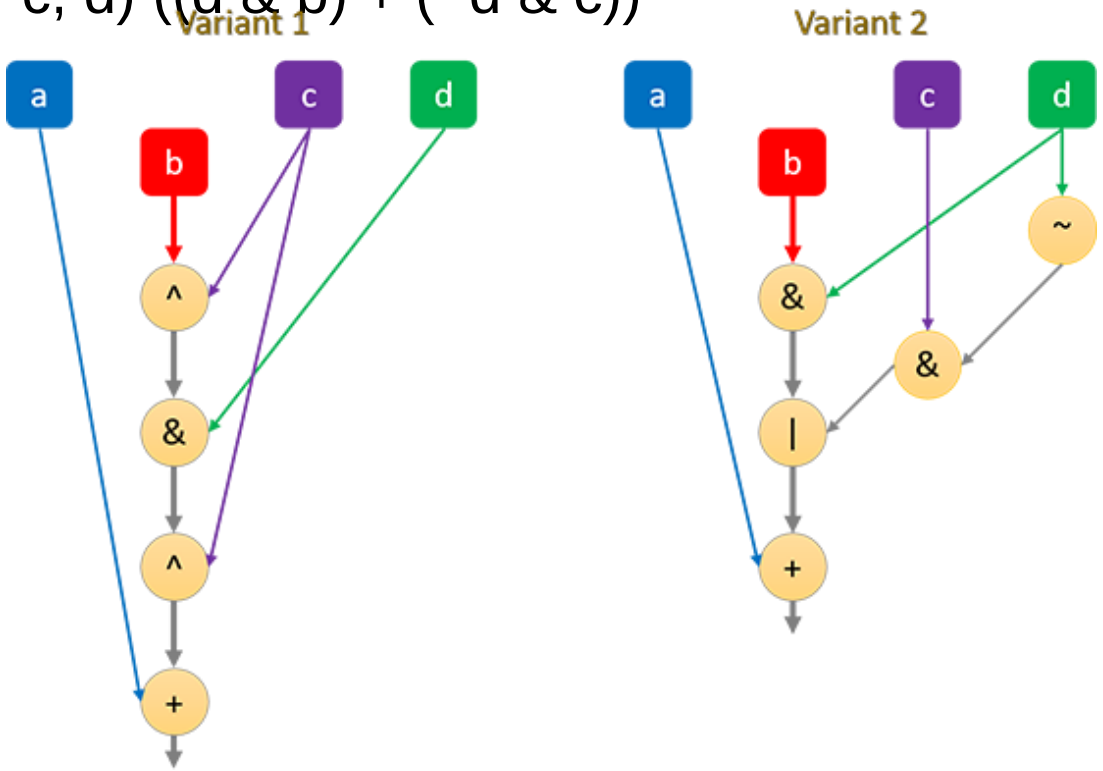
최적화

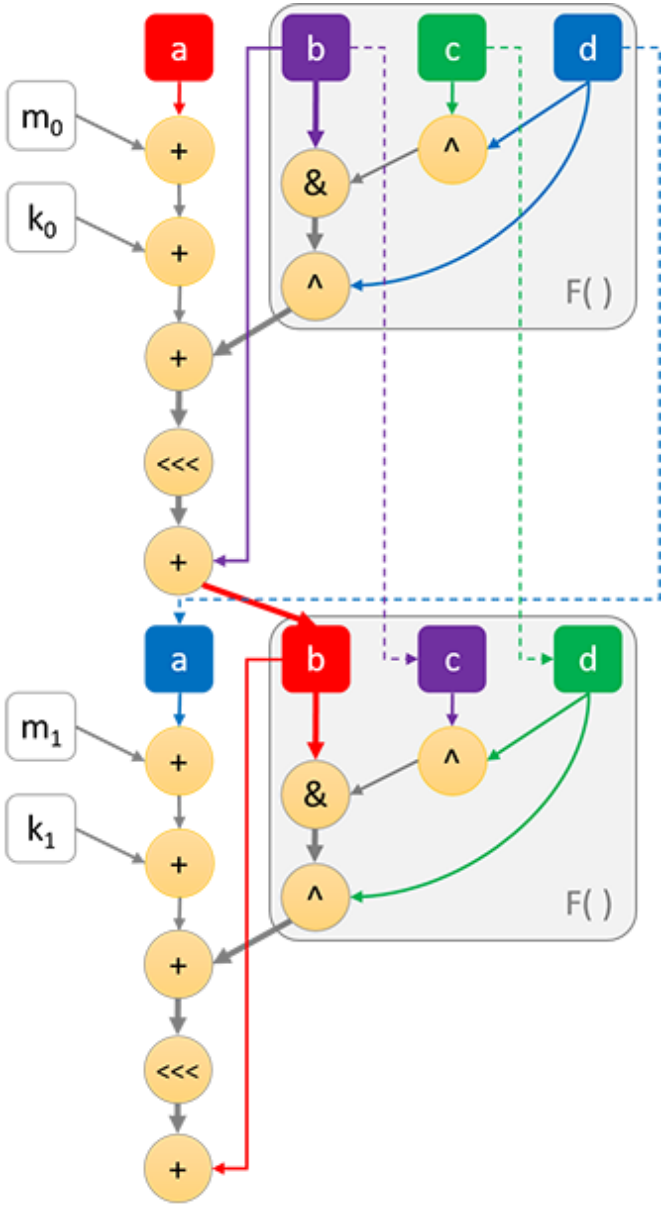
https://github.com/animetosho/md5-optimisation#readme

몇가지 트릭사용 5%, AVX512를 지원하는 프로세서의 경우 23% 성능향상

G 코드 단축

#define G(b, c, d) ((d & b) + (~d & c))





Hashcat MD5 최적화

- 5개의 해시캣 최적화 기법

  - Zero-based optimizations
  - Initial-step optimizations
  - Early-exit optimizations
  - Precomputing
  - Partial reversing / Meet-in-the-middle

Zero-based optimization

- 사용자가 설정한 암호 문자열은 알고리즘 입력 데이터 블록으로 전달 할때 패딩이 이루어진다.
  - 예를 들어, ‘password’라는 암호는 두개의 32-bit 정수 사용
  - 일반적으로 512-bit로 고정되어 있어 암호 이외의 값은 0으로 패딩
  - 16개로 구성된 버퍼의 w[0], w[1]에 ‘password’에 대한 값으로 채워지고 나머지는 0으로 채워짐

```
0000000: 70617373 776f7264 00000000 00000000  password........
0000010: 00000000 00000000 00000000 00000000  ................
0000020: 00000000 00000000 00000000 00000000  ................
0000030: 00000000 00000000 00000000 00000000  ................
```

  - 암호의 길이가 8인 경우, w[2] ~ w[15]는 0가 되어 w[2] ~ w[15]를 사용하는 모든 함수 호출을 제거 할 수 있음

Zero-based optimization

- MD5 함수의 단일 단계 함수 연산 방법

```
#define FF(a, b, c, d, x, s, ac) a += F (b, c, d) + x + ac; …
```

  - 위의 함수는 16번 호출.
  - 각 호출은 입력 벡터의 다른 요소를 사용.
  - 암호문의 길이 < 12 일때, 총 x / 128 ADD 명령어 저장.

Zero-based optimization

```
KERNEL_FQ void m00000_s04 (KERN_ATTR_VECTOR ())
{
  /**
   * base
   */

  const u64 gid = get_global_id (0);

  if (gid >= gid_max) return;

  u32 w[16];

  w[ 0] = pws[gid].i[ 0];
  w[ 1] = pws[gid].i[ 1];
  w[ 2] = pws[gid].i[ 2];
  w[ 3] = pws[gid].i[ 3];
  w[ 4] = 0;
  w[ 5] = 0;
  w[ 6] = 0;
  w[ 7] = 0;
  w[ 8] = 0;
  w[ 9] = 0;
  w[10] = 0;
  w[11] = 0;
  w[12] = 0;
  w[13] = 0;
  w[14] = pws[gid].i[14];
  w[15] = 0;

  const u32 pw_len = pws[gid].pw_len & 63;

  /**
   * main
   */

  m00000s (w, pw_len, pws, rules_buf, combs_buf, words_buf_r,
}
```

```
KERNEL_FQ void m00000_s08 (KERN_ATTR_VECTOR ())
{
  /**
   * base
   */

  const u64 gid = get_global_id (0);

  if (gid >= gid_max) return;

  u32 w[16];

  w[ 0] = pws[gid].i[ 0];
  w[ 1] = pws[gid].i[ 1];
  w[ 2] = pws[gid].i[ 2];
  w[ 3] = pws[gid].i[ 3];
  w[ 4] = pws[gid].i[ 4];
  w[ 5] = pws[gid].i[ 5];
  w[ 6] = pws[gid].i[ 6];
  w[ 7] = pws[gid].i[ 7];
  w[ 8] = 0;
  w[ 9] = 0;
  w[10] = 0;
  w[11] = 0;
  w[12] = 0;
  w[13] = 0;
  w[14] = pws[gid].i[14];
  w[15] = 0;

  const u32 pw_len = pws[gid].pw_len & 63;

  /**
   * main
   */

  m00000s (w, pw_len, pws, rules_buf, combs_buf,
}
```

Initial-step optimizations

상수를 확장하여 처음 몇 라운드를 단순화 하는 기법

- 짧은 입력일 경우 input[ ] 이 0이 되어 일부 단계를 뛰어넘을 수 있음

```
A = 0x67452301; B = 0xefcdab89; C = 0x98badcfe; D = 0x10325476;
A += input[0] + 0xd76aa478 + F(B, C, D);  => 0xd76aa477 + input[ 0 ];
A = ROTATE_LEFT(A, 7);
A += B;
D += input[1] + 0xe8c7b756 + F(A, B, C); => D = 0xf8fa0bcc + input[ 1 ] + (( 0x77777777 & A) ^ 0x98badcfe );
D = ROTATE_LEFT(D, 12);
D += A;
C += input[2] + 0x242070db + F(D, A, B); => C = 0xbcdb4dd9 + input[ 2 ] + (((A ^ 0xefcdab89 ) & D) ^ 0xefcdab89 );
C = ROTATE_LEFT(C, 17);
C += D;
B += input[3] + 0xc1bdceee + F(C, D, A);
B = ROTATE_LEFT(B, 22);
B += C;
```

***B,C,D 가 알려져 있으므로 사전 연산 가능부분**

Initial-step optimizations

```
/**
 * base
 */
const u32 F_w0c00 =    0u + MD5C00;
const u32 F_w1c01 = w[ 1] + MD5C01;
const u32 F_w2c02 = w[ 2] + MD5C02;
const u32 F_w3c03 = w[ 3] + MD5C03;
const u32 F_w4c04 = w[ 4] + MD5C04;
const u32 F_w5c05 = w[ 5] + MD5C05;
const u32 F_w6c06 = w[ 6] + MD5C06;
const u32 F_w7c07 = w[ 7] + MD5C07;
const u32 F_w8c08 = w[ 8] + MD5C08;
const u32 F_w9c09 = w[ 9] + MD5C09;
const u32 F_wac0a = w[10] + MD5C0a;
const u32 F_wbc0b = w[11] + MD5C0b;
const u32 F_wcc0c = w[12] + MD5C0c;
const u32 F_wdc0d = w[13] + MD5C0d;
const u32 F_wec0e = w[14] + MD5C0e;
const u32 F_wfc0f = w[15] + MD5C0f;

const u32 G_w1c10 = w[ 1] + MD5C10;
const u32 G_w6c11 = w[ 6] + MD5C11;
const u32 G_wbc12 = w[11] + MD5C12;
const u32 G_w0c13 =    0u + MD5C13;
const u32 G_w5c14 = w[ 5] + MD5C14;
const u32 G_wac15 = w[10] + MD5C15;
const u32 G_wfc16 = w[15] + MD5C16;
const u32 G_w4c17 = w[ 4] + MD5C17;
const u32 G_w9c18 = w[ 9] + MD5C18;
const u32 G_wec19 = w[14] + MD5C19;
const u32 G_w3c1a = w[ 3] + MD5C1a;
const u32 G_w8c1b = w[ 8] + MD5C1b;
const u32 G_wdc1c = w[13] + MD5C1c;
const u32 G_w2c1d = w[ 2] + MD5C1d;
const u32 G_w7c1e = w[ 7] + MD5C1e;
const u32 G_wcc1f = w[12] + MD5C1f;

const u32 H_w5c20 = w[ 5] + MD5C20;
const u32 H_w8c21 = w[ 8] + MD5C21;
const u32 H_wbc22 = w[11] + MD5C22;
const u32 H_wec23 = w[14] + MD5C23;
const u32 H_w1c24 = w[ 1] + MD5C24;
const u32 H_w4c25 = w[ 4] + MD5C25;
const u32 H_w7c26 = w[ 7] + MD5C26;
const u32 H_wac27 = w[10] + MD5C27;
const u32 H_wdc28 = w[13] + MD5C28;
const u32 H_w0c29 =    0u + MD5C29;
const u32 H_w3c2a = w[ 3] + MD5C2a;
const u32 H_w6c2b = w[ 6] + MD5C2b;
const u32 H_w9c2c = w[ 9] + MD5C2c;
const u32 H_wcc2d = w[12] + MD5C2d;
const u32 H_wfc2e = w[15] + MD5C2e;
const u32 H_w2c2f = w[ 2] + MD5C2f;

const u32 I_w0c30 =    0u + MD5C30;
const u32 I_w7c31 = w[ 7] + MD5C31;
const u32 I_wec32 = w[14] + MD5C32;
const u32 I_w5c33 = w[ 5] + MD5C33;
const u32 I_wcc34 = w[12] + MD5C34;
const u32 I_w3c35 = w[ 3] + MD5C35;
const u32 I_wac36 = w[10] + MD5C36;
const u32 I_w1c37 = w[ 1] + MD5C37;
const u32 I_w8c38 = w[ 8] + MD5C38;
const u32 I_wfc39 = w[15] + MD5C39;
const u32 I_w6c3a = w[ 6] + MD5C3a;
const u32 I_wdc3b = w[13] + MD5C3b;
const u32 I_w4c3c = w[ 4] + MD5C3c;
const u32 I_wbc3d = w[11] + MD5C3d;
const u32 I_w2c3e = w[ 2] + MD5C3e;
const u32 I_w9c3f = w[ 9] + MD5C3f;
```

Precomputing

이전 해시 연산연산의 일부를 LUT에 저장하여 호출하는 기법으로 높은 수준의 최적화 제공

- SHA-256를 계산하고 일부 문자만 변경하는 candidate generator를 가정
  - 변경되는 문자는 패스워드의 마지막 문자
- 즉, w[0]와 같은 첫 4 문자는 수정되지 않음

```
t1 = H + S1(E) + Ch(E,F,G) + K[0] + W[0];
t2 = S0(A) + Maj(A,B,C);
H = G; G = F; F = E; E = D + t1; D = C; C = B; B = A; A = t1 + t2;
```

- A ~ H 및 K[0], w[0]는 상수이므로 값들을 사용한 값은 모두 연산 결과가 동일함. 따라서 미리 계산한 값을 LUT에 저장하여 호출하는 것이 가능

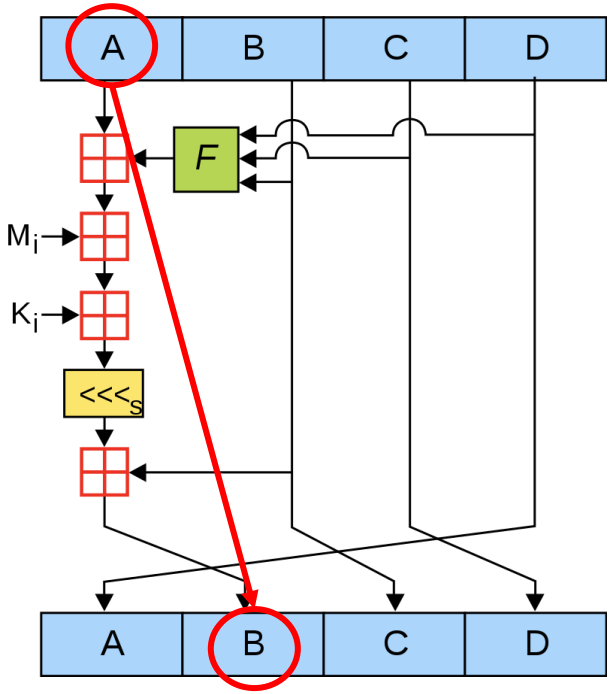
- 메모리를 얼마나 사용하는가?, 메모리에서 얼마나 빠르게 값을 찾는가? 에 따라 속도 차이를 보인다.

Early-exit optimizations

연산이 완료된 해시값의 일부만 비교하는 최적화 기법

- MD5 의 61라운드에서 state[A]가 마지막으로 연산되어 state[B]에 저장됨. 그 후 마지막 라운드(64)까지 자리 이동만 있을 뿐, 값의 변화는 없음

- 따라서 62라운드 진행전 61라운드에서 연산된 state[A]의 값을 Target hash(target[0]) 값과 비교 후 같지 않다면 더이상 라운드를 진행하지 않고 loop를 빠져나옴.



Early-exit optimizations

```
/**
 * reverse
 */

u32 a_rev = digests_buf[DIGESTS_OFFSET].digest_buf[0];
u32 b_rev = digests_buf[DIGESTS_OFFSET].digest_buf[1];
u32 c_rev = digests_buf[DIGESTS_OFFSET].digest_buf[2];
u32 d_rev = digests_buf[DIGESTS_OFFSET].digest_buf[3];

MD5_STEP_REV (MD5_I_S, b_rev, c_rev, d_rev, a_rev, w[ 9], MD5C3f, MD5S33);
MD5_STEP_REV (MD5_I_S, c_rev, d_rev, a_rev, b_rev, w[ 2], MD5C3e, MD5S32);
MD5_STEP_REV (MD5_I_S, d_rev, a_rev, b_rev, c_rev, w[11], MD5C3d, MD5S31);
MD5_STEP_REV (MD5_I_S, a_rev, b_rev, c_rev, d_rev, w[ 4], MD5C3c, MD5S30);
MD5_STEP_REV (MD5_I_S, b_rev, c_rev, d_rev, a_rev, w[13], MD5C3b, MD5S33);
MD5_STEP_REV (MD5_I_S, c_rev, d_rev, a_rev, b_rev, w[ 6], MD5C3a, MD5S32);
MD5_STEP_REV (MD5_I_S, d_rev, a_rev, b_rev, c_rev, w[15], MD5C39, MD5S31);
MD5_STEP_REV (MD5_I_S, a_rev, b_rev, c_rev, d_rev, w[ 8], MD5C38, MD5S30);
MD5_STEP_REV (MD5_I_S, b_rev, c_rev, d_rev, a_rev, w[ 1], MD5C37, MD5S33);
MD5_STEP_REV (MD5_I_S, c_rev, d_rev, a_rev, b_rev, w[10], MD5C36, MD5S32);
MD5_STEP_REV (MD5_I_S, d_rev, a_rev, b_rev, c_rev, w[ 3], MD5C35, MD5S31);
MD5_STEP_REV (MD5_I_S, a_rev, b_rev, c_rev, d_rev, w[12], MD5C34, MD5S30);
MD5_STEP_REV (MD5_I_S, b_rev, c_rev, d_rev, a_rev, w[ 5], MD5C33, MD5S33);
MD5_STEP_REV (MD5_I_S, c_rev, d_rev, a_rev, b_rev, w[14], MD5C32, MD5S32);
MD5_STEP_REV (MD5_I_S, d_rev, a_rev, b_rev, c_rev, w[ 7], MD5C31, MD5S31);
MD5_STEP_REV (MD5_I_S, a_rev, b_rev, c_rev, d_rev,     0, MD5C30, MD5S30);

const u32 pre_cd = c_rev ^ d_rev;

MD5_STEP_REV1(MD5_H_S, b_rev, c_rev, d_rev, a_rev, w[ 2], MD5C2f, MD5S23);
MD5_STEP_REV1(MD5_H_S, c_rev, d_rev, a_rev, b_rev, w[15], MD5C2e, MD5S22);
```

```
if (MATCHES_NONE_VV (pre_c, c)) continue;

MD5_STEP0(MD5_H2, b, c, d, a,     H_w6c2b, MD5S23);
MD5_STEP0(MD5_H1, a, b, c, d,     H_w9c2c, MD5S20);
MD5_STEP0(MD5_H2, d, a, b, c,     H_wcc2d, MD5S21);

if (MATCHES_NONE_VV (pre_d, d)) continue;

MD5_STEP0(MD5_H1, c, d, a, b,     H_wfc2e, MD5S22);
MD5_STEP0(MD5_H2, b, c, d, a,     H_w2c2f, MD5S23);

MD5_STEP (MD5_I , a, b, c, d, w0, I_w0c30, MD5S30);
MD5_STEP0(MD5_I , d, a, b, c,     I_w7c31, MD5S31);
MD5_STEP0(MD5_I , c, d, a, b,     I_wec32, MD5S32);
MD5_STEP0(MD5_I , b, c, d, a,     I_w5c33, MD5S33);
MD5_STEP0(MD5_I , a, b, c, d,     I_wcc34, MD5S30);
MD5_STEP0(MD5_I , d, a, b, c,     I_w3c35, MD5S31);
MD5_STEP0(MD5_I , c, d, a, b,     I_wac36, MD5S32);
MD5_STEP0(MD5_I , b, c, d, a,     I_w1c37, MD5S33);
MD5_STEP0(MD5_I , a, b, c, d,     I_w8c38, MD5S30);
MD5_STEP0(MD5_I , d, a, b, c,     I_wfc39, MD5S31);
MD5_STEP0(MD5_I , c, d, a, b,     I_w6c3a, MD5S32);
MD5_STEP0(MD5_I , b, c, d, a,     I_wdc3b, MD5S33);
MD5_STEP0(MD5_I , a, b, c, d,     I_w4c3c, MD5S30);
MD5_STEP0(MD5_I , d, a, b, c,     I_wbc3d, MD5S31);
MD5_STEP0(MD5_I , c, d, a, b,     I_w2c3e, MD5S32);
MD5_STEP0(MD5_I , b, c, d, a,     I_w9c3f, MD5S33);

COMPARE_S_SIMD (a, d, c, b);
```

Reversing a.k.a. Meet-in-the-middle

- MD5 Digest를 생성하는데 사용한 원본 값이 있을 때 적용 가능
  - 특정 지점까지 해시 연산을 진행하여 MD5 Digest를 생성
  - 예상 값을 사용하여 새로 생성한 Digest와 기존 Digest를 비교
  - 두 값이 일치하지 않는다면, 예상 값은 원본과 다른 것이므로 연산 중단

- 연산을 끝까지 수행하지 않고 키 일치 여부를 알 수 있음
  - 필요 없는 연산은 중간에 중단하기에 속도 향상 가능

Reversing a.k.a. Meet-in-the-middle

- 패스워드의 처음 4자만 바꾸고 일정하게 유지하는 생성기를 가정
  - w[0] : 변경, w[1]~w[15] : 유지

```
AAAApass

AAABpass

ZZZZpass
```

- 해시 연산 시, w[0]를 사용하지 않는 단계까지 연산 후 Digest 저장

- 내부 반복에서 정상적인 해시 연산을 진행
  - 기존에 Digest를 저장한 단계까지 수행

- 생성한 Digest를 기존 Digest와 비교

- 값이 일치하지 않는다면 w[1]~w[15]가 다른 것이므로 계산이 필요하지 않음

Q & A