높은 보안 강도를 위한 DB 재암호화 성능 분석

https://youtu.be/CHw2KsElmrA

한성대학교 HANSUNG UNIVERSITY

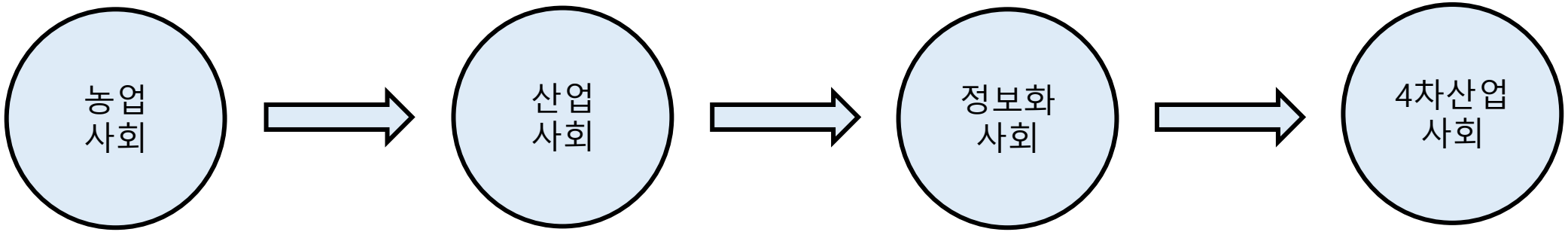
CryptoCraft LAB

x. DB에서의 암호 알고리즘 전환 연구 목표

- 산업사회 이후 데이터의 양과 사용이 크게 증가
- 데이터를 잘 주고 받는 것도 중요하지만, 데이터를 잘 저장하는 것도 중요

~18세기 후반
농업이 주요 산업이며, 인구 증가와 함께 정착 생활이 시작

20세기 ~ 현재
인터넷과 디지털 기술의 발전으로 인해 정보의 생산과 유통이 중요



18세기 ~ 20세기
기계의 발명과 산업혁명으로 인해 제조업이 발달, 도시화가 진행

앞으로
인공지능, 빅데이터, 사물인터넷 등의 기술이 발전

x. DB에서의 암호 알고리즘 전환 연구 목표

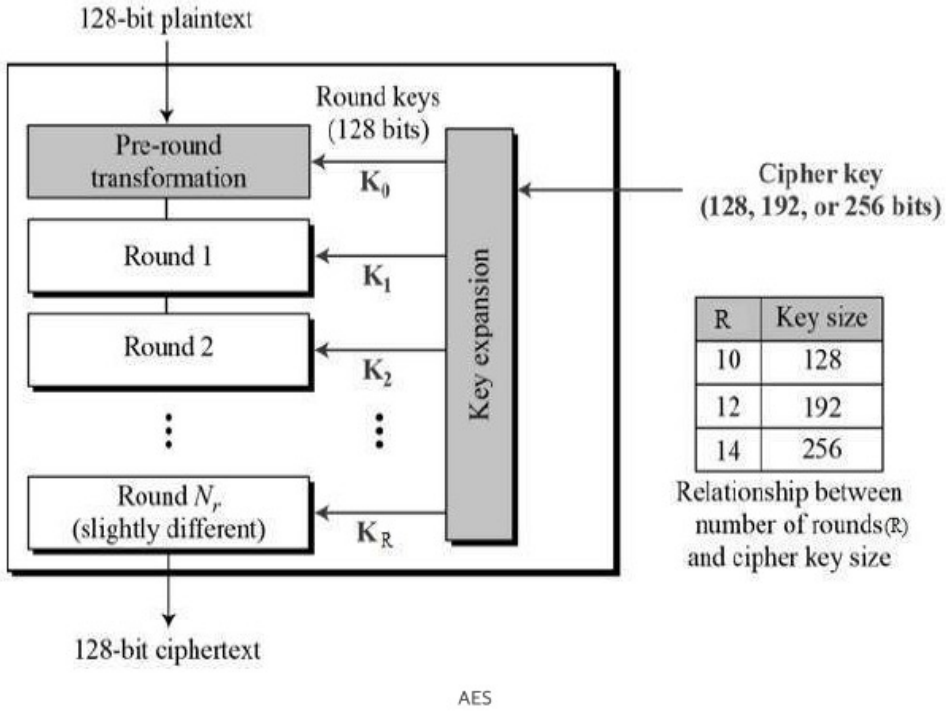
- 데이터베이스 보안은 DB와 그 안의 정보를 다양한 위험으로부터 보호하는 활동
- 앞선 연구는 데이터를 안전하게 주고 받기 위한 통신 과정에서의 PQC 알고리즘으로 전환 시 성능 테스트 진행
- 데이터를 안전하게 읽고 쓰기 위해 데이터베이스에서 사용되는 암호 알고리즘의 보안 강도도 높아져야 함
- 데이터베이스 재암호화 성능 테스트
  - 기존의 AES-128로 암호화된 데이터에 대한 AES-256으로 재암호화

x. DB에서의 암호 알고리즘 전환

- AES-128과 AES-256 차이

키 길이	라운드 수	라운드 키의 수
128	10	11
256	14	15

- 키 길이가 길어지면서 라운드키의 수가 많아지고, 키 확장 과정에서 추가적인 연산 발생



R	Key size
10	128
12	192
14	256

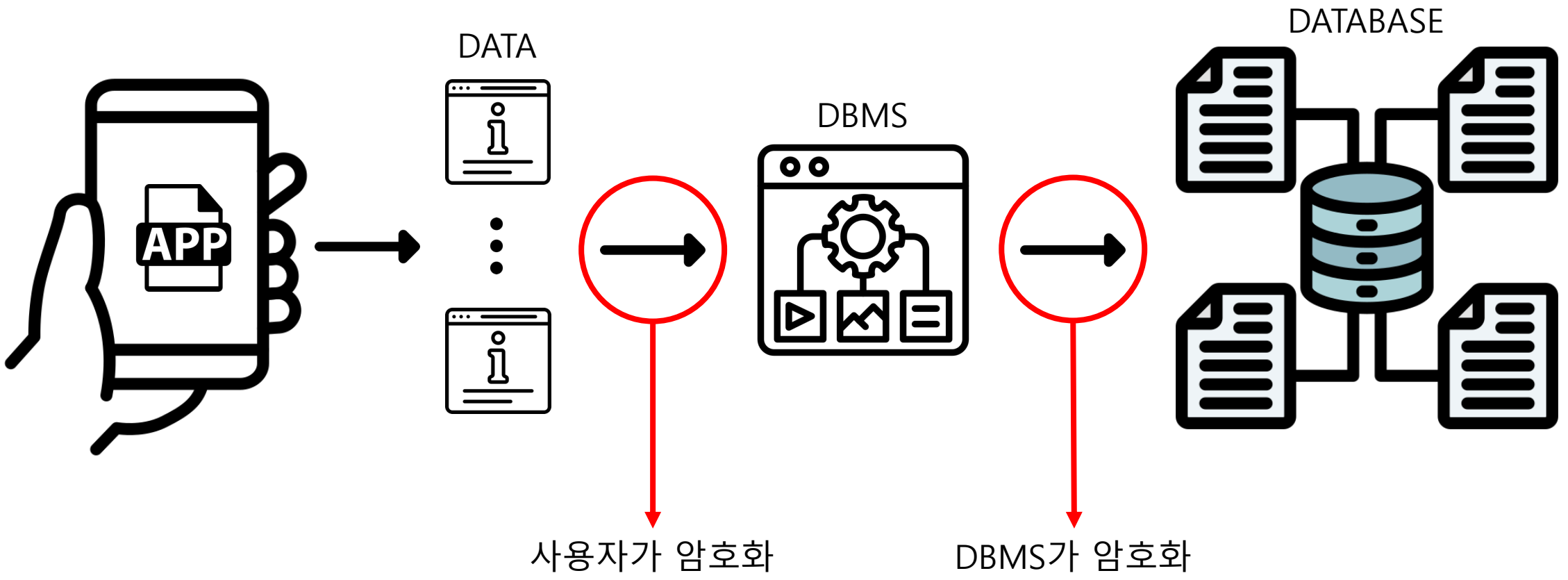
Relationship between number of rounds(R) and cipher key size

AES

https://yjshin.tistory.com/353

x. DB에서의 암호 알고리즘 전환

- DB에서의 암호화 방식
  - 사용자가 암호화 하는 경우
  - 사용자가 전달한 데이터를 DBMS에서 암호화하는 경우



x. DB에서의 암호 알고리즘 전환

- DBMS(Database Management System)
  - 데이터베이스 관리 시스템
  - 데이터를 구조화하고 저장하며, 데이터에 접근하고 조작하는 기능을 제공하는 소프트웨어
    - 데이터베이스를 효율적으로 관리하고 조작할 수 있도록 다양한 기능과 도구를 제공
- Stackoverflow에서 해마다 개발자를 대상으로 하는 설문 결과
  - PostgreSQL과 MySQL는 전문 개발자들이 많이 사용하고, MySQL과 SQLite는 학습용으로 많이 사용하는 것으로 조사됨.

프로그래밍 언어 순위

전체 DBMS 순위

- 대중적이고 널리 알려진 MySQL을 사용

참고 : https://survey.stackoverflow.co/2023/#most-popular-technologies-database-learn

x. DB에서의 암호 알고리즘 전환

- MySQL – 가장 널리 사용되는 관계형 데이터베이스 관리 시스템
  - 빠른 처리 속도와 안정성
  - 오픈소스
  - 플랫폼 독립성 – 다양한 운영체제에서 사용 가능
  - 보안 – 데이터 암호화 및 접근 제어 기능과 같은 보안 기능 제공

- MySQL에서 제공하는 암호화 함수
  - AES, SHA, MD5와 같은 기본적인 암호화 함수 제공

12.13 Encryption and Compression Functions

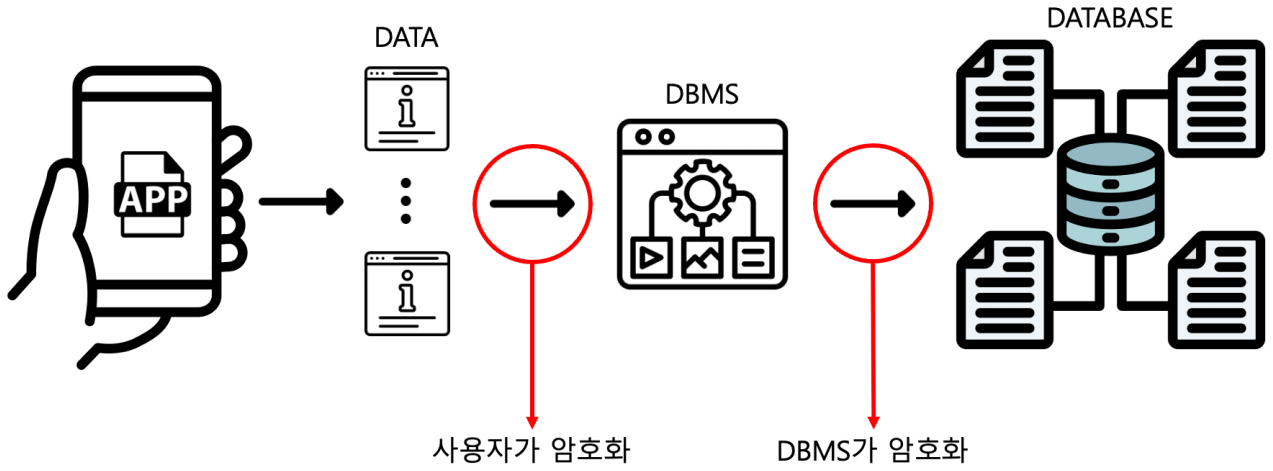
Table 12.18 Encryption Functions

Name	Description
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES
COMPRESS()	Return result as a binary string
MD5()	Calculate MD5 checksum
RANDOM_BYTES()	Return a random byte vector
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
STATEMENT_DIGEST()	Compute statement digest hash value
STATEMENT_DIGEST_TEXT()	Compute normalized statement digest
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

https://dev.mysql.com/doc/refman/8.0/en/encryption-functions.html#function_password

x. DB에서의 암호 알고리즘 전환

- DBMS에서 암호화
  - 사용자에게 받은 데이터를 암호화한 후에 데이터베이스에 입력
  - DBMS에서 제공하는 암호화 함수 사용
- 사용자가 암호화
  - 사용자가 암호화한 데이터를 DBMS에게 전달
  - DBMS에서는 암호화 과정 없이 데이터를 바로 데이터베이스에 입력
  - 따라서 외부 암호화 함수를 사용



x. 성능 측정을 위한 벤치마크 구현

- 데이터베이스 테이블 구성
  - 암호화된 데이터가 저장되는 test 테이블 / 사용된 키와 IV가 저장되는 key_manage 테이블

Test 테이블

```
mysql> DESC TEST;
+--------------+---------------+------+-----+---------+----------------+
| Field        | Type          | Null | Key | Default | Extra          |
+--------------+---------------+------+-----+---------+----------------+
| userid       | int           | NO   | PRI | NULL    | auto_increment |
| ciphertext_1 | varchar(256)  | YES  |     | NULL    |                |
| ciphertext_2 | varchar(512)  | YES  |     | NULL    |                |
| ciphertext_3 | varchar(3072) | YES  |     | NULL    |                |
+--------------+---------------+------+-----+---------+----------------+
```

userid	ciphertext_1	ciphertext_2	ciphertext_3
int	Varchar(256)	varchar(512)	Varchar(3072)
Index로 사용	16바이트 미만의 랜덤한 작은 데이터	16바이트의 고정 데이터	16~512바이트의 랜덤한 데이터

테스트에 필요한 데이터

Key_manage 테이블

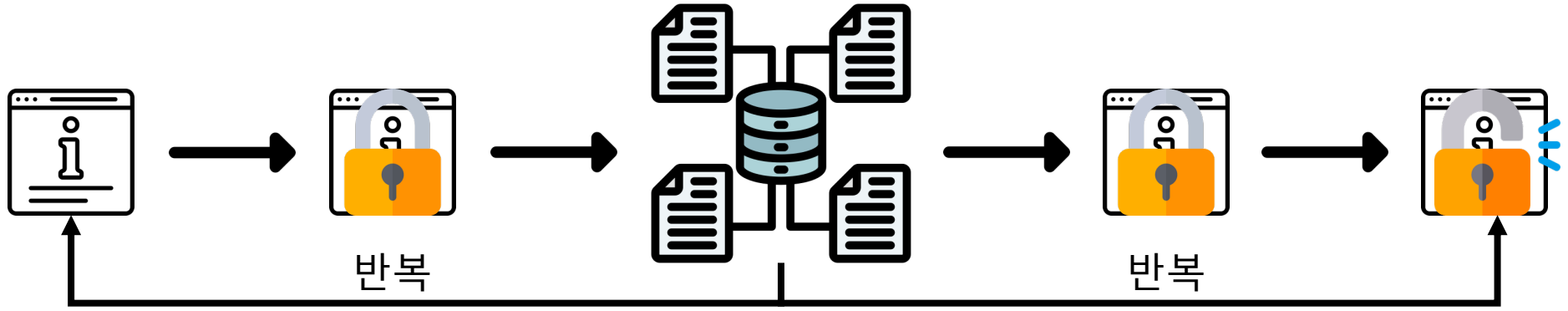
```
mysql> desc key_manage;
+-------------+---------------+------+-----+---------+----------------+
| Field       | Type          | Null | Key | Default | Extra          |
+-------------+---------------+------+-----+---------+----------------+
| userid      | int           | NO   | PRI | NULL    | auto_increment |
| temp_text_1 | varchar(256)  | YES  |     | NULL    |                |
| temp_text_2 | varchar(512)  | YES  |     | NULL    |                |
| temp_text_3 | varchar(3072) | YES  |     | NULL    |                |
| user_key    | varchar(64)   | YES  |     | NULL    |                |
| user_iv     | varchar(64)   | YES  |     | NULL    |                |
+-------------+---------------+------+-----+---------+----------------+
```

userid	Temp_text_1	Temp_text_2	Temp_text_3	User_key	User_iv
int	Varchar(256)	varchar(512)	Varchar(3072)	Varchar(64)	Varchar(64)
Index로 사용	암호화 이전의 평문	암호화 이전의 평문	암호화 이전의 평문	사용된 key	사용된 iv

값을 확인하기 위한 데이터

x. 성능 측정을 위한 벤치마크 구현

- 구현된 벤치마크 시나리오
  - DB에 암호화된 데이터 입력
    - 데이터베이스에 암호화된 데이터를 입력하고 시간을 측정
    - 랜덤한 text, key, iv를 성정하고 암호화된 text를 데이터베이스에 입력하는 과정을 반복
    - 암호화 과정과 데이터를 입력하는 쿼리만 시간을 측정
  - DB에 저장된 암호화된 데이터 읽기
    - 데이터베이스에 암호화되어 저장된 데이터를 가져와 복호화하여 데이터를 읽는 시간을 측정, 데이터를 읽는 방법을 두개로 나누어 구현
      - Index 순서대로 데이터 읽기
      - 랜덤한 Index의 데이터 읽기
    - 암호화된 데이터와 Key_manage 테이블에 저장된 key, iv를 가져오는 과정도 포함하여 시간 측정



x. 성능 측정을 위한 벤치마크 구현

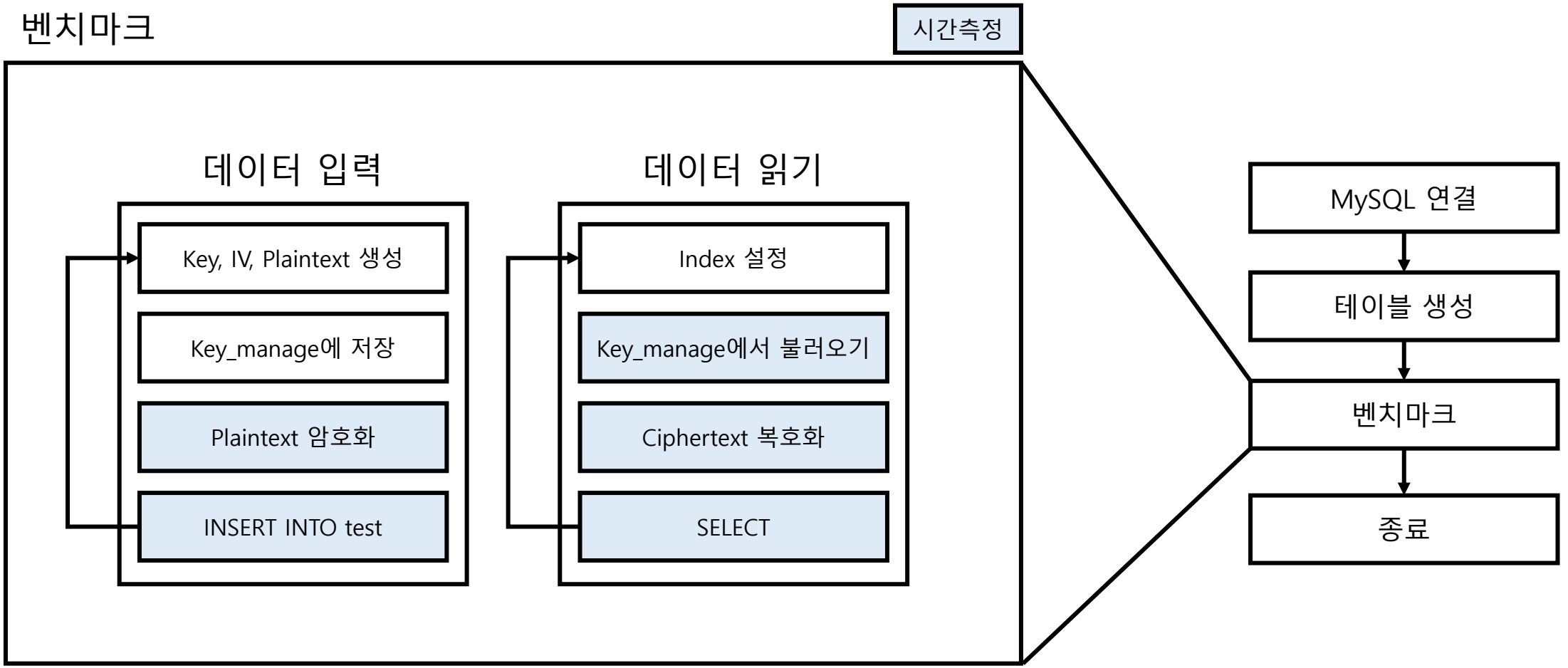
- 벤치마크 구현은 Python을 활용하여 구현을 진행
  - Pymysql을 사용하여 MySQL을 활용
  - Pycryptodome의 암호화 함수를 사용
    - AES, RSA, SHA256 등 다양한 암호화 및 해시 알고리즘을 사용할 수 있음

- MySQL에서의 암호화 함수 설정
  - SELECT @@block_encryption_mode; 를 통해서 확인 가능
    - 기본적으로 'aes_128_ecb' 로 되어 있음
  - SET block_encryption_mode='알고리즘';으로 변경
    - AES-128-CBC / AES-256-CBC 두 알고리즘을 비교 분석함

```
query = "SELECT @@block_encryption_mode;"
cur.execute(query)
```

```
query = "SET block_encryption_mode='aes-256-cbc'"
cur.execute(query)
```

x. 성능 측정을 위한 벤치마크 구현

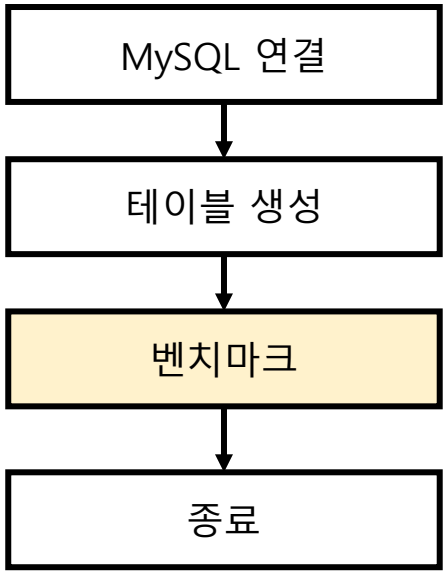


단순화 했다 설명

x. 성능 측정을 위한 벤치마크 구현

- 데이터 쓰기 테스트 – MySQL 암호화 함수 사용
  - 반복 수 만큼 데이터 생성 및 쓰기
  - SET @key 와 같이 세션 변수에 일시적으로 데이터를 저장해서 암호화에 사용
  - 데이터 다양성을 위해서 다양한 데이터 길이를 암호화

```
def mysql_write_only_test():
    for i in range(Iteration):
        try:
            mysql_set_randombyte(key_length)        랜덤 data, key, iv 생성
            mysql_insert_key_table()                키 관리 테이블에 key, iv 값 저장
            query = ("INSERT INTO test(ciphertext_1, ciphertext_2, ciphertext_3) VALUES("
                     "HEX(AES_ENCRYPT(@text_1, @key, @iv)),"
                     "HEX(AES_ENCRYPT(@text_2, @key, @iv)),"
                     "HEX(AES_ENCRYPT(@text_3, @key, @iv))"
                     ")")
            cur.execute(query)                      암호화 및 데이터 입력
        except Exception as ex:
            print(ex)
```



x. 성능 측정을 위한 벤치마크 구현

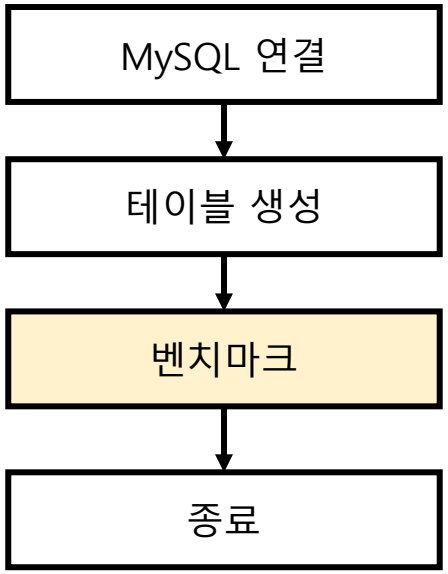
- 데이터 읽기 테스트 – MySQL 암호화 함수 사용
  - SELECT @key 와 같이 테이블에 저장된 Key, iv, ciphertext 값을 세션 변수에 일시적으로 저장하여 복호화에 사용

```
def mysql_read_only_test():
    # 순서대로 값을 읽을 때
    for i in range(1, Iteration + 1):
        mysql_set_key_iv_cipher(i)
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_1), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_2), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_3), @key, @iv)")
    # Random한 값을 읽을 때
    for i in range(1, Iteration + 1):
        mysql_set_key_iv_cipher(random.randint(1, Iteration))
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_1), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_2), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_3), @key, @iv)")
```

1~n 으로 순서대로 증가

랜덤한 index

복호화 및 데이터 읽기



x. 성능 측정을 위한 벤치마크 구현

- 데이터 쓰기 테스트 – Python Pycryptodome 라이브러리 암호화 함수 사용
  - MySQL 암호화 함수가 아닌 Python 암호화 함수를 사용하기 때문에 데이터베이스에 입력하기 전에 암호화 진행

```python
def mysql_write_only_test():
    for i in range(Iteration):
        try:
            mysql_set_randombyte(key_length)
            mysql_insert_key_table()
            query = ("INSERT INTO test(ciphertext_1, ciphertext_2, ciphertext_3) VALUES("
                     "HEX(AES_ENCRYPT(@text_1, @key, @iv)),"
                     "HEX(AES_ENCRYPT(@text_2, @key, @iv)),"
                     "HEX(AES_ENCRYPT(@text_3, @key, @iv))"
                     ")")
            cur.execute(query)
        except Exception as ex:
            print(ex)
```

```python
def python_mysql_write_only_test():
    for i in range(Iteration):
        key, iv, text_1, text_2, text_3 = python_mysql_set_randombyte(key_length)
        python_mysql_insert_key_table(key, iv, text_1, text_2, text_3)
        cipher = AES.new(key, mode, iv)키 확장 및 암호화 초기 설정
        text_1, text_2, text_3 = padding_data(text_1, text_2, text_3, pad)  입력값 패딩
        cipher_1 = cipher.encrypt(text_1)
        cipher_2 = cipher.encrypt(text_2)  암호화
        cipher_3 = cipher.encrypt(text_3)
        query = ("INSERT INTO test(ciphertext_1, ciphertext_2, ciphertext_3) VALUES ("
                 "'" + cipher_1.hex() + "',"
                 "'" + cipher_2.hex() + "',"
                 "'" + cipher_3.hex() + "'"          데이터베이스에 입력
                 ")")
        cur.execute(query)
```

x. 성능 측정을 위한 벤치마크 구현

- 데이터 읽기 테스트 – Python Pycryptodome 라이브러리 암호화 함수 사용
  - 키를 사용자가 가지고 있다고 가정하여, 데이터베이스에서 키와 IV 값을 읽어서 사용하지 않음.

```python
def mysql_read_only_test():
    # 순서대로 값을 읽을 때
    for i in range(1, Iteration + 1):
        mysql_set_key_iv_cipher(i)
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_1), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_2), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_3), @key, @iv)")
    # Random한 값을 읽을 때
    for i in range(1, Iteration + 1):
        mysql_set_key_iv_cipher(random.randint(1, Iteration))
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_1), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_2), @key, @iv)")
        cur.execute("SELECT AES_DECRYPT(UNHEX(@cipher_3), @key, @iv)")
```

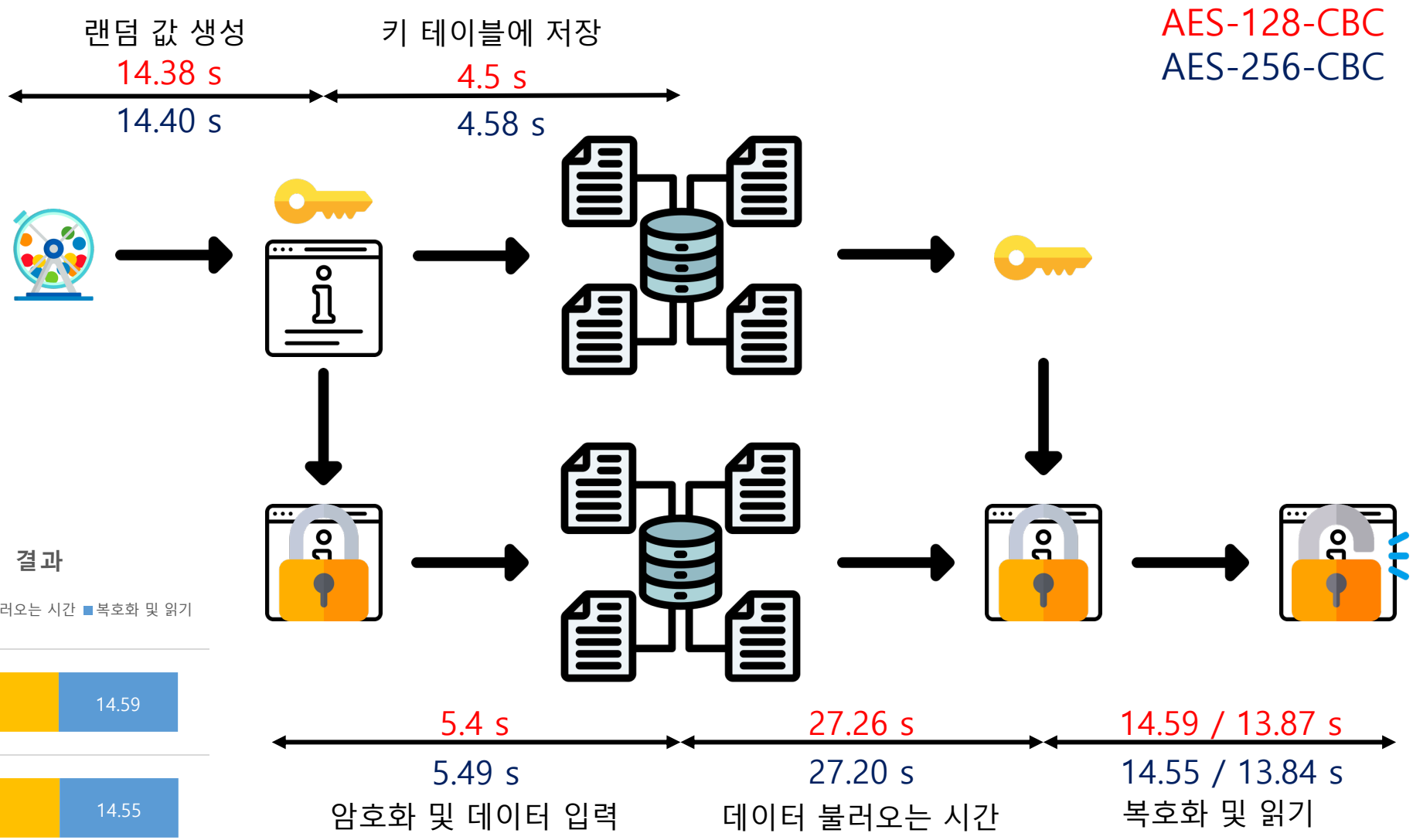
```python
def python_mysql_read_only_test():
    cipher_1 = None
    cipher_2 = None
    cipher_3 = None
    # 순서대로 값을 읽을 때
    for i in range(1, Iteration + 1):
        cur.execute("SELECT ciphertext_1, ciphertext_2, ciphertext_3 FROM test WHERE userid = " + str(i))
        for cur_str in cur:
            cipher_1 = bytes.fromhex(cur_str[0])
            cipher_2 = bytes.fromhex(cur_str[1])
            cipher_3 = bytes.fromhex(cur_str[2])
        key = key_array[i - 1]
        iv = iv_array[i - 1]
        cipher = AES.new(key, mode, iv)
        text_1 = cipher.decrypt(cipher_1)
        text_2 = cipher.decrypt(cipher_2)
        text_3 = cipher.decrypt(cipher_3)
        text_1, text_2, text_3 = padding_data(text_1, text_2, text_3, unpad)
    # Random한 값을 읽을 때
    for i in range(1, Iteration + 1):
        index = random.randint(1, Iteration)
        cur.execute("SELECT ciphertext_1, ciphertext_2, ciphertext_3 FROM test WHERE userid = " + str(index))
        for cur_str in cur:
            cipher_1 = bytes.fromhex(cur_str[0])
            cipher_2 = bytes.fromhex(cur_str[1])
            cipher_3 = bytes.fromhex(cur_str[2])
        key = key_array[index - 1]
        iv = iv_array[index - 1]
        cipher = AES.new(key, mode, iv)
        text_1 = cipher.decrypt(cipher_1)
        text_2 = cipher.decrypt(cipher_2)
        text_3 = cipher.decrypt(cipher_3)
        text_1, text_2, text_3 = padding_data(text_1, text_2, text_3, unpad)
```

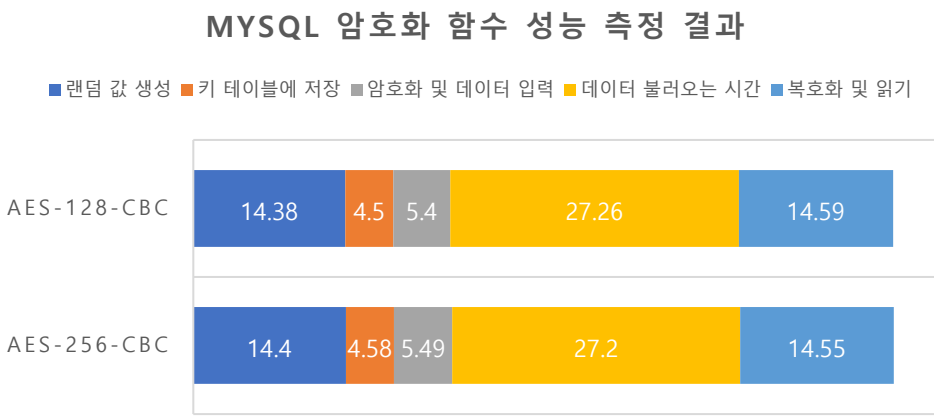
데이터베이스에서 암호문 불러와 변수에 저장

복호화 및 패딩된 값 제거

x. 성능 측정을 위한 벤치마크 구현

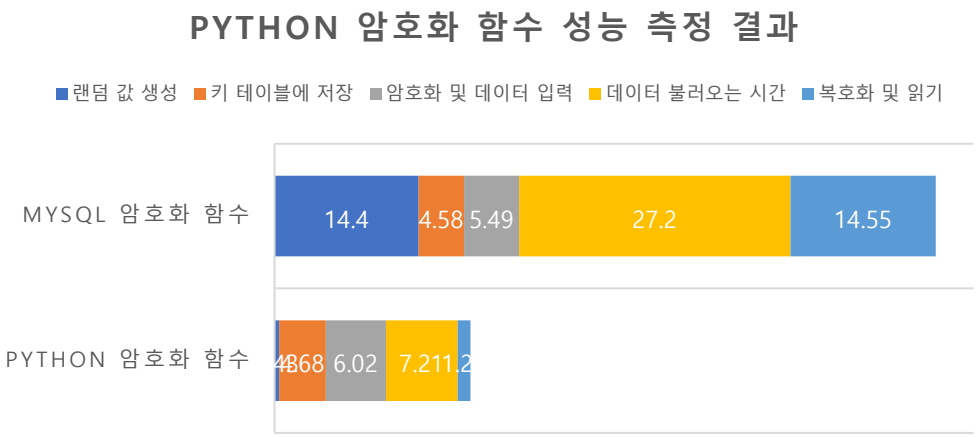
MySQL 암호화 함수 성능 측정 결과

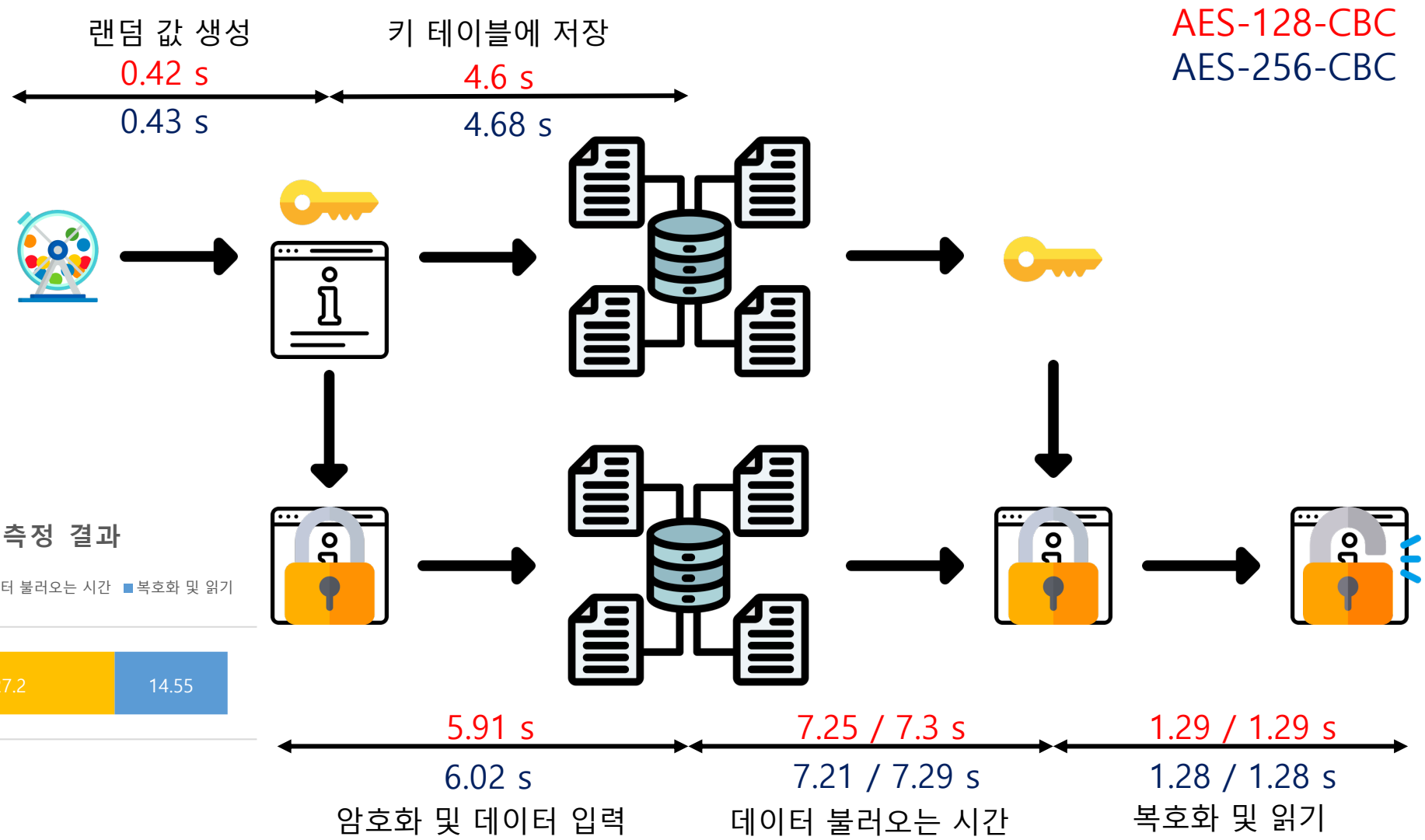




x. 성능 측정을 위한 벤치마크 구현

Python 암호화 함수 성능 측정 결과

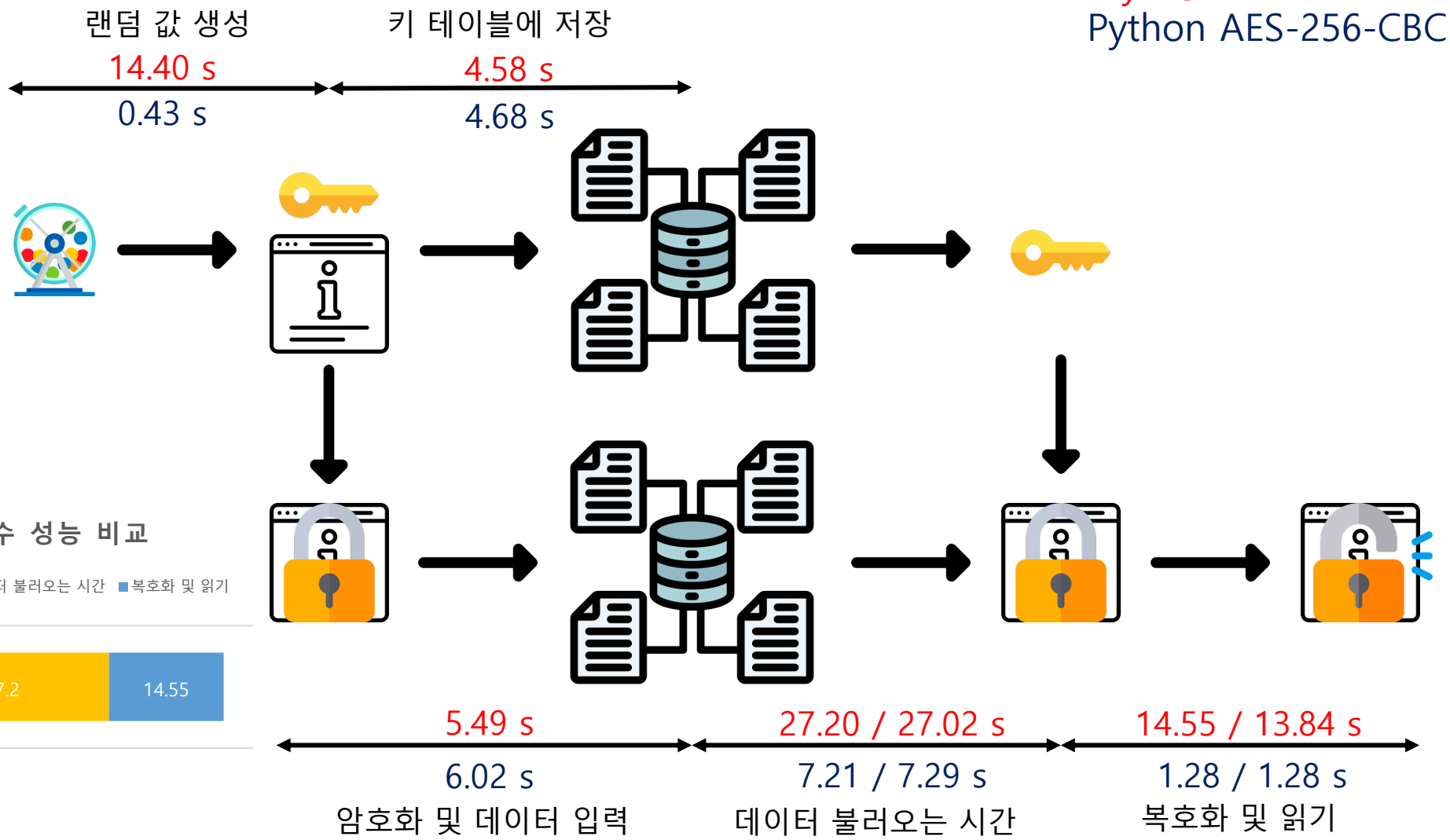


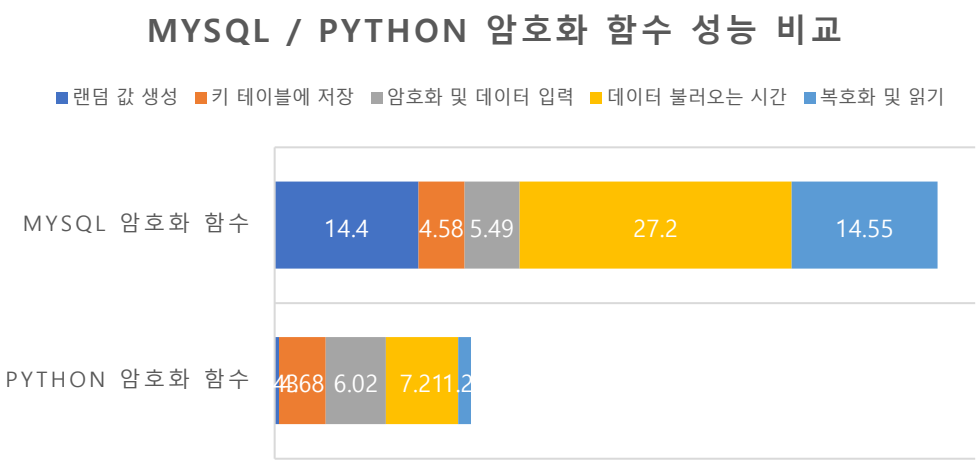


x. 성능 측정을 위한 벤치마크 구현

MySQL 암호화 함수
Python 암호화 함수
성능 비교

MySQL AES-256-CBC
Python AES-256-CBC

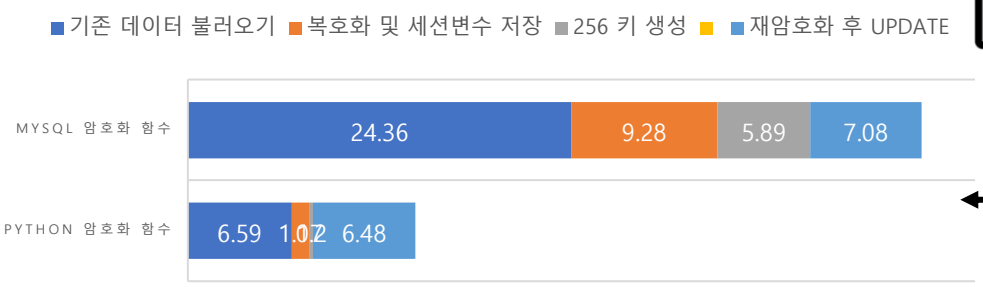


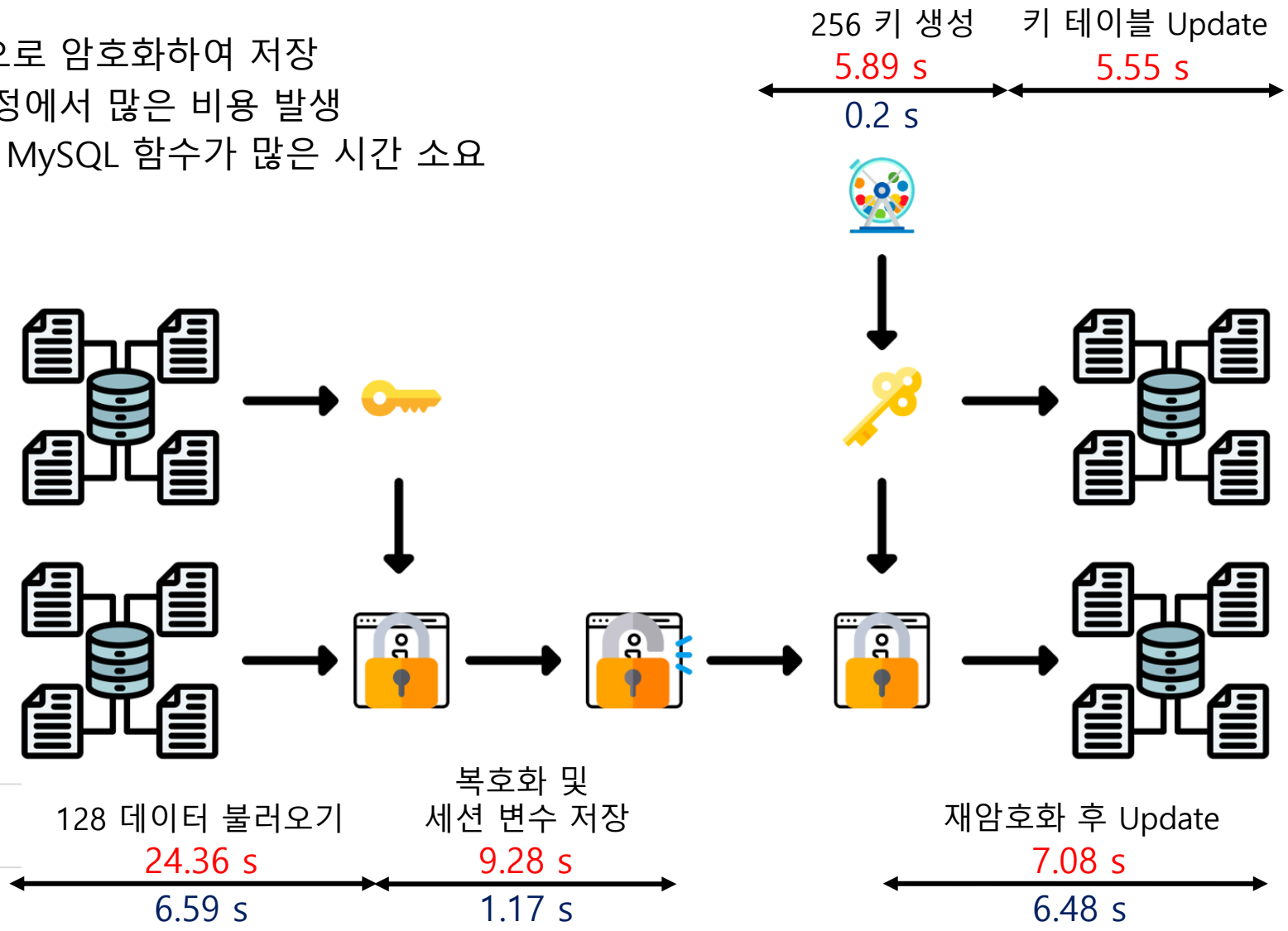


x. 성능 측정을 위한 벤치마크 구현

MySQL AES-256-CBC
Python AES-256-CBC

- 전체 재암호화 성능 측정
  - 128로 암호화된 데이터를 다시 256으로 암호화하여 저장
  - 데이터베이스에서 값을 불러오는 과정에서 많은 비용 발생
  - 랜덤 값 생성을 위한 Random_byte() MySQL 함수가 많은 시간 소요
    - 랜덤바이트 함수 비교 (10000 반복)
      - Mysql : 0.55 s
      - Python : 0.005 s





x. 성능 측정 및 평가

- 높은 보안 강도를 위해서 키 길이를 변경하였을 때, 암/복호화로 인한 비용 증가는 미미함
  - 다만 증가하는 키 길이로 인해서 늘어난 키를 저장하기 위한 더 많은 공간이 필요

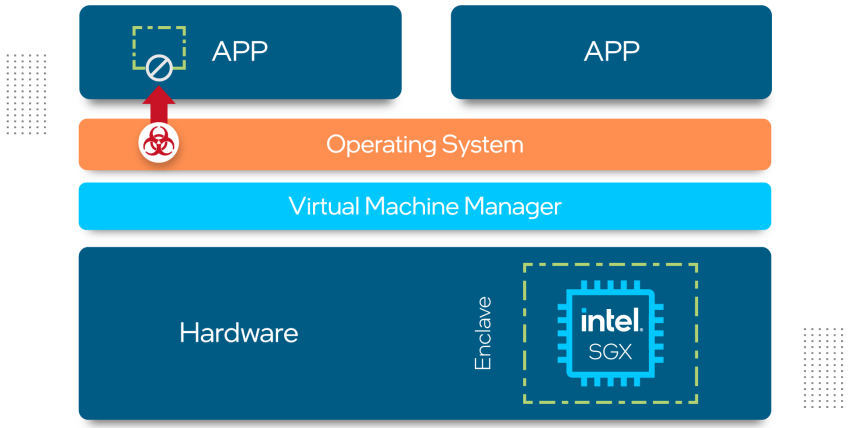
- 하지만 키 길이가 변경됨으로 인해 기존의 키 길이로 저장되어 있는(128-bit) 데이터를 변경된 키 길이(256-bit)로 다시 암호화하여 저장하는 작업은 많은 시간이 소요됨
  - MySQL 에서 제공하는 함수 의존성도 줄여야 함

- 또한 재암호화를 위해서 암호화된 데이터를 원본 데이터로 복원시켜야 하기 때문에 데이터가 노출되는 문제가 발생

재암호화를 하는 작업은 많은 시간이 소요되고 데이터가 노출되기 때문에
확실하고 안전하게 재암호화를 하는 방법에 대한 조사도 진행

x. 안전한 재암호화를 위한 방법 고찰

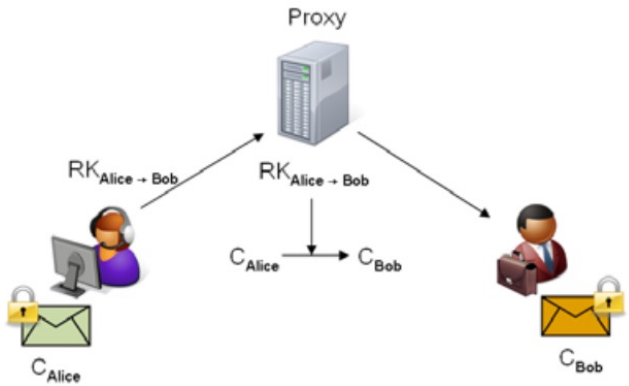
- SGX를 통해서 안전한 환경에서 암호화
  - SGX(Software Guard Extension)
    - 인텔 CPU에 적용된 기술로 사용자가 엔크레이브(enclave)라고 하는 보호 영역을 설정해 CPU가 메모리 일부를 암호화하고, 보호영역에 있는 데이터는 보호 영역 내에서 실행되는 프로그램 이외에는 접근할 수 없도록 하는 기술
    - 즉, 암호화된 비밀을 보호하고 민감한 코드 실행을 메모리에 격리해 신뢰실행환경(Trucsted Execution Environment, TEE)을 제공하는 것이 목적
    - ARM CPU에서도 TrustZone이라고 하는 하드웨어 기반 보안 확장 기능을 제공
  - TEE 환경에서 재암호화를 진행하여 외부로부터의 공격으로 부터 안전하게 재암호화를 진행할 수 있음.



출처 : https://melonicedlatte.com/2021/01/27/084700.html
출처 : https://www.intel.com/content/www/us/en/developer/articles/technical/sgx-device-plugin.html
출처 : https://learn.microsoft.com/ko-kr/azure/confidential-computing/confidential-computing-enclaves

x. 안전한 재암호화를 위한 방법 고찰

- Re-Encryption 기법을 활용하여 암호화
  - Re-Encryption
    - 일반적으로 암호화된 메시지를 복호화하지 않고 다른 암호화된 형태로 변환하는 과정을 의미
    - 주로 Proxy Re-Encryption으로 알려져 있음
      - Proxy Re-Encryption
        - Proxy는 중개자 대리인을 의미하는 단어로, 컴퓨터 쪽에서는 클라이언트와 서버 사이에 위치하여 데이터의 전송을 중개하는 역할을 수행
        - Alice의 공개키로 암호화된 암호문을 Bob의 비밀키로 복호할 수 있도록 암호문을 변환하는 방식
        - Proxy는 암호문을 변환하기 위한 키(re-encryption key)를 이용하여 기존의 암호문을 복호하지 않고 암호문을 변환할 수 있음.
  - 이러한 기법을 적용하게 되면, 128-bit 키를 사용하여 암호화된 데이터를 Re-encryption 기법을 적용하여 256-bit 키로 복호화할 수 있도록 재암호화를 하는 방법
  - 하지만 기존의 기법을 그대로 적용하게 될 경우 128-bit의 기존 키도 복호화를 할 수 있는 문제가 있음
    - Bob의 비밀키로 복호할 수 있지만 여전히 Alice도 복호화할 수 있는 문제가 해결되어야 함

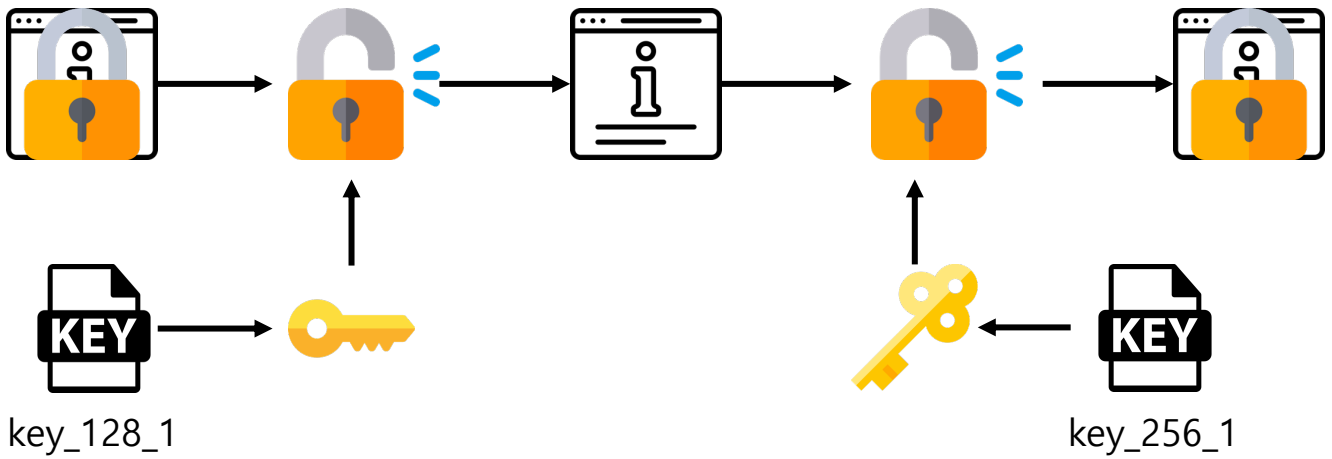


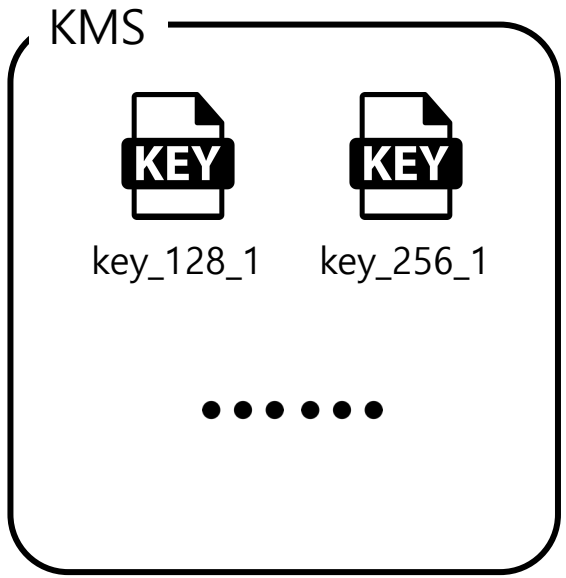
출처 : 송유진, 박광용. (2009). Proxy Re-encryption 기술. 정보보호학회지, 19(5), 95-104.

x. 안전한 재암호화를 위한 방법 고찰

- 키 버전 관리를 통한 재암호화 방법
  - Key Management System
    - 암호화 통신이나 데이터 보호를 위한 보안 시스템 중 하나
    - 데이터를 암호화할 때 사용되는 암호화 키를 안전하게 관리하는데 목적을 둔 시스템
    - 전용 장비를 사용하거나, 최근에는 AWS, Microsoft, Google 과 같은 클라우드 KMS 서비스가 있음
    - 키 생성, 저장, 분배, 순환과 같은 기능을 제공함

  - 한번에 모든 데이터를 재암호화를 하는 것은 많은 시간과 비용이 필요하기 때문에 데이터가 사용되고 다시 저장될 때 키 버전을 기록하여 새로운 키를 사용해 저장하는 방법





출처 : https://cloud.google.com/kms/docs/re-encrypt-data?hl=ko#re-encrypt

감 사 합 니 다