2020년도 1학기 졸업 프로젝트

Another Photo Editor

5조

201411257 컴퓨터공학과 강정모

201411307 컴퓨터공학과 이한강

201611284 컴퓨터공학과 이유진

KU KONKUK UNIVERSITY

목 차

1. 프로젝트 기획, 계획

1.1 프로젝트 기획

교수님께서 처음 프로젝트 주제를 선정할 때, 기존에 있는 프로그램을 Low Level부터 직접 만들어 보는 것도 좋다고 하셨기 때문에, 관심이 있었던 과목 중 하나였던 멀티미디어 쪽 주제를 정하고자 하였다. 온라인 동영상 컨버터나 영상에서 오디오 추출하기 등등의 주제를 고민하였으나 카카오톡으로 친한 친구들에게 이미지를 보내던 도중 이미지 편집기를 만들어보면 괜찮지 않을까 하는 생각을 하게 되었고, 그대로 주제를 이미지 편집기, 즉 포토 에디터로 정하게 되었다.

주제는 포토 에디터였으나 교수님께서 기존에 이미 있던 것을 직접 Low Level부터 구현해보는 것을 프로젝트 주제로 선정할 경우, 여러가지 요소에 대해서는 기존 프로젝트보다 떨어질 수 있지만 대신 특정 부분에 대해서는 우리 프로젝트가 더 좋다, 라고 할 만한 요소를 반드시 추가하라고 하셨고, 그 부분에 대해 자체적으로 DRM(Digital Right Management, 디지털 권리 관리) 기술을 추가하면 좋을 것이라고 생각하였다.

그렇게 그림판과 같이 대칭, 회전, 잘라내기, 해상도 조절, 파일 불러오기 및 저장, 이미지 포맷 간 변환 기능 등 다수의 기본적인 기능을 제공하는 포토 에디터에, 필요한 경우 사용할 수 있도록 자체적으로 DRM과 관련된 암호화 기술을 도입한 Another Photo Editor 프로젝트를 기획하게 되었다.

1.2 프로젝트 계획

Step 1. 프로젝트 제안서. (03.27 ~ 04.04)

처음 주제가 통과되고 나서 가장 기본적인 프로젝트 제안서를 쓰게 되었을 때, 우리는 교수님의 요구사항 중 image와 관련된 Library 및 API (ex, Pillow, OpenCV 등등)를 일절 쓰지 말 것이라는 사항과 관련하여 생각보다 프로젝트의 난이도가 올라갈 것이라고 판단하였다. 대칭, 회전, 잘라내기와 같은 이미지 편집 기능의 경우에는 image의 RGB 값을 저장하고 있는 numpy array를 처리하는 것으로 구현할 수 있을 것이라고 생각하였으나, 이미지를 binary data로부터 불러오는 과정을 Library 없이 구현해야한다는 것은 결국 image와 관련된 codec을 직접 만들어야야함을 의미하였다.

물론 codec 역시 직접 구현할 생각은 하고 있었고, 어느 특정 이미지 포맷 하나에 대해서라면 충분할 것이라고 생각하였으나, 교수님께서 요구하신 사항은 최소한 jpeg, png, bmp 이미지는 불러올 수 있어야 한다고 하셨기 때문에 처음에 생각했던 것 이상으로 프로젝트의 구현 난이도가 올라갔다고 판단하였다. 그래서 교수님이 요구하신 이미지 포맷 간 불러오기 및 인코딩 (변환) 기능에 초점을 맞추고, 기획 당시 있었던 기능의 일부를 제외하고 회전, 대칭 및 우리 프로젝트 만의 특징인 DRM 기술로 구현하고자 하는 목표를 압축하였다.

또한 GUI 코딩이 Low Level과 관련된 코딩은 아닐 것이라고 판단하여 GUI 코딩을 하는 대신 ffmpeg와 같은 terminal 창 기반에서 command로 실행되는 프로그램을 만들기로 하였다. DRM 기술의

경우 암호화를 고민하였으나 스테가노그래피 기법이라 하여, image의 low bit를 손실시키고 특정 정보를 넣어 숨기는 정보 은닉 기술을 컴퓨터 보안 시간에 배웠기 때문에 이를 실제로 구현해보고 프로젝트에 접목해보면 좋은 공부가 될 것이라 생각하여 스테가노그래피 기법을 최종적으로 선택하였다.

Step 2. 요구사항 분석. (04.06 ~ 04.11)
제안서가 별 다른 수정 사항 없이 통과되었다고 생각하여, 위의 4가지 기능에 초점을 맞춘 요구사항 분석을 진행하였다. 회전의 경우 90도 단위의 좌/우 방향 회전, 대칭의 경우 상하/좌우 대칭, DRM의 경우 스테가노그래피 기법을 워터마크와 접목하여 이미지 내부에 워터마크를 숨기고, 다시 복원하는 기능 위주로 요구사항을 작성하였다. 포맷 변환의 경우에도 교수님이 말씀하신 3가지 포맷간 변환에 초점을 맞추어 요구사항을 작성하였다.

component / deployment 다이어그램의 경우 명확한 내용을 몰라 class 다이어그램을 대신 넣었고, 교수님께서 말씀하신 대로 class 와 요구사항을 매핑하였다. test case의 경우 각 요구사항을 체크할 수 있는 test case 1개씩을 선정하여 제출하였다.

Step 3. 요구사항 분석(2). (04.13 ~ 04.18)
요구사항 분석이 1차적으로 있은 후 교수님께서는 더 해볼 것을 요구하셨다. 즉 기능을 좀 더 추가하고, GUI도 도입해보고 하면서 전체적인 프로젝트의 크기를 키우실 것을 바라셨다. 또 만약 GUI를 사용하지 않을 경우 terminal 창에서의 command를 parsing 할 때 YACC이나 LEX와 같은 parser tool을 사용하시길 바라셨는데 해당 tool들에 대해서는 접해본 적이 없었고, 또한 문법과 관련된 컴파일러 강의를 수강하지 않아 어려움이 따를 것으로 판단해 GUI 쪽을 구현하기로 결정하였다.

또 기능을 추가하고자 기존에 주제 선정 당시 썼던 기능을 모두 다시 반영하였다. 자유 각도 회전, 해상도 조절, 잘라내기 기능이 새로 추가되었으며 GUI가 도입되므로 확대 및 축소 기능과 함께 undo/redo 기능도 추가하게 되었다. DRM에 도입하기로 한 스테가노그래피 기술의 경우 정보 은닉 기법으로 삽입한 워터마크가 다양한 경로로 인해 손실될 수 있다는 의견이 팀 내에서 지속적으로 제기되어왔고, 그에 대한 효과적인 방법을 계속 모색하였으나 결국 짧은 기간 내에 효과적인 방법을 찾을 것이 어려울 것으로 사료되어 pixel noising 기법으로 방향을 전환하게 되었다. 특정 key를 입력받아 pixel들의 RGB 값을 흐트러트려 기존의 image를 사용할 수 없게 하는 원리로 구현하고자 하였다.

또한 component diagram 대신 architecture diagram을 그려 모듈 별로 기능 및 요구사항 mapping을 다시 하였다. GUI가 도입될 것이므로 prototype illustration 역시 새로 만들었다. test case의 경우 좀 더 구체적으로 요구사항을 테스팅 할 수 있는 것과 동시에 다양한 포맷으로 test case set을 만드실 것을 바라셔서 6장의 서로 다른 포맷 이미지로 test case set을 생성하였다.

Step 4. 디자인. (04.15 ~ 04.25)
요구사항(2) ~ 디자인 동안, 다시 한 번 DRM 기술에 대해 논의하였으나 결국 pixel noising 조차 완벽하지 않다는 의견이 제시되었다. 단적인 예로 pixel noising 된 image라 하더라도 다른 편집기로 불러오고 편집 하는 것이 가능하고, 이 경우 잘라내기 혹은 해상도 변환을 하게 될 경우 noising 하는 시작 지점이 변경되어 정상적인 key를 가지고도 다시 image를 복원하지 못하는 문제가 발생하게 된다.

그래서 추가로 더 고민한 끝에, 결국 최종적으로 처음 주제를 선정할 때 썼던 암호화 방향으로 노선을 결정하였다. 편집하기 때문에 이미지에 가한 여러 처리가 손실되게 된다면 아예 열 수 없게 만드는 것이 가장 좋은 방법이라고 생각하였고, 아예 열 수 없게 만들기 위해서는 file signature를 손상시키면 되지만 단순히 손상 시키고 복원하는 것을 떠나 좀 더 보안성을 강화하고 싶었다. 그래서 SHA 256 함수와 key의 길이가 일치하며 RSA와 같은 비대칭키 암호보다 속도가 더 빠른 AES 대칭키 암호 방식을 선택하게 되었다.

암호화를 사용하게 되었으므로 내부적으로 module 간의 연결 상태가 변화하여 architecture diagram을 새로 수정하였고, 요구사항에 따른 test case도 수정하였다. 이후 High level의 마지막 단계인 Sequence diagram을 그려야 했으나 약 2주 간의 기간 동안 계속하여 효과적인 DRM 기술에 대해 고민하였고 그 결과 architecture diagram에서 component들이 변경되면서 sequence가 달라지게 되었다. 해당 부분에 대한 검수 및 확인을 지속적으로 하느라 sequence diagram을 다 완성하지 못하였고, sequence diagram을 완성하지 못하자 필요한 sequence에 따른 module별 interface를 정하는데 어려움이 따르게 되었다. 이는 최종적으로 class diagram 및 traceability matrix의 미완성으로 이어지게 되었다.

해당 부분을 완성하지는 못했지만 일단 구현으로 들어가게 되었고, 또한 그 동안 팀원들 간에 많은 고민 및 의견 교환이 있었기 때문에 내부적으로 PM module 및 각 Module 들간의 interface는 구두로 합의된 상황이었기 때문에, 이후 구현 및 중간 발표에 이르기까지 다행히 큰 문제는 발생하지 않았다.

Step 5. 중간발표 1~2차. (05.04 ~ 05.18 / 05.18 ~ 05.29)
디자인 이후 Image format 별로 각자 codec을 전담하여 구현하기로 하였고, codec을 제외하고 그 외의 module 기능을 또 팀원들이 나누어 맡아 구현하기로 하였다. 처음 계획은 중간발표 1차까지 codec 및 편집 기능을 완성하고, 2차까지는 GUI를 완성하는 것이었으나 codec 별로 구현하는데 생각보다 난이도가 있어 계획이 상당 부분 지연되게 되었다. 따라서 다음과 같은 계획으로 진행되었다.

중간 발표 1차 (05.04 ~ 05.18)
- 좌/우 90도 회전, 자유각도 회전, 좌우 대칭, bmp decoder 구현 완료
- jpeg decoder, SHA-256 / AES-256, png decoder, cropping 구현 중

중간 발표 2차 (05.18 ~ 05.29)

- 해상도 조절, jpeg decoder, SHA-256, png decoder, cropping, 상하 대칭, bmp encoder, png encoder, processing management module 연관 기능 구현 완료
- jpeg encoder, AES-256, GUI 구현 중

Step 6. 최종 발표 (05.29 ~ 06.17)

구현 중이던 jpeg encoder, AES-256, GUI 구현을 완료하였다.

2. SRS

2.1 Functional Requirements

(1) 해상도 조절

1.1 자유 선택

사용자가 직접 가로 해상도와 세로 해상도를 지정하여 image의 해상도를 변경하고, ~~변경된 image와 변경 전 image를 함께~~ **변경된** image**를** 화면에 출력한다.

1.2 비율 유지

1.2.1 가로 해상도 지정

사용자가 직접 가로 해상도를 지정하면 자동으로 image의 가로 세로 해상도 비에 따라 세로 해상도가 정해지고 image의 해상도가 변경된다. 그리고 ~~변경된 image와 변경 전 image를 함께~~ **변경된** image**를** 화면에 출력한다.

1.2.2 세로 해상도 지정

사용자가 직접 세로 해상도를 지정하면 자동으로 image의 가로 세로 해상도 비에 따라 가로 해상도가 정해지고 image의 해상도가 변경된다. 그리고 ~~변경된 image와 변경 전 image를 함께~~ **변경된** image**를** 화면에 출력한다.

(2) 회전

2.1 왼쪽으로 회전

2.1.1 90도 단위로 반복

입력을 줄 때마다 왼쪽으로 90도 → 180도 → 270도 → 360도(처음) 순으로 반복하여 image를 회전시키고 ~~회전된 image를 변경 전 image와 함께~~ **회전된** image**를** 화면에 출력한다.

2.2 오른쪽으로 회전

2.2.1 90도 단위로 반복

입력을 줄 때마다 오른쪽으로 90도 → 180도 → 270도 → 360도(처음) 순으로 반복하여 image를 회전시키고 ~~회전된 image를 변경 전 image와 함께~~ 회전된 image를 화면에 출력한다.

2.3 원하는 방향으로 0° ~45° 각도 회전

사용자가 왼쪽과 오른쪽 중 원하는 방향을 선택 후 0° ~45° 내에서 원하는 각도를 입력하여 image를 회전시키고, ~~회전된 image를 변경 전 image와 함께~~ 회전된 image를 화면에 출력한다.

(3) 대칭

3.1 상하 대칭

image를 세로 해상도의 중간값을 기준으로 대칭시키고 ~~대칭된 image와 변경 전 image를 함께~~ 대칭된 image를 화면에 출력한다.

3.2 좌우 대칭

image를 가로 해상도의 중간값을 기준으로 대칭시키고 ~~대칭된 image와 변경 전 image를 함께~~ 대칭된 image를 화면에 출력한다.

※ (1) ~ (3)의 SRS가 변경된 이유

⇒ 중간 발표 2차에서 GUI구현 계획이 하나의 출력 창만 가지는 것으로 변경되었기 때문에 이를 반영한 것이다.

(4)DRM

4.1 AES256 암호화

사용자로부터 ~~최대 256Byte의 string data를 입력 받아~~ 최대 (2^64)-1 Byte의 string data를 입력 받아 SHA-256으로 key를 생성하고, 생성한 key를 가지고 AES-256 암호화 알고리즘을 사용하여 현재 image를 암호화하여 저장한다.

4.2 AES256 복호화

사용자로부터 ~~최대 256Bytes의 string data를 입력 받아~~ 최대 (2^64)-1 Byte의 string data를 입력 받아 SHA-256으로 key를 생성하고, 생성한 key를 가지고 암호화된 image를 복호화하여 저장한다

※ (4)의 SRS가 변경된 이유

⇒ 보안 강도를 높이고자 중간 발표 2차에서 변경되었음.

(5) 변환

5.1 jpg to bmp

jpg format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ 해상도를 유지하며 bmp format으로 변환한다.

5.2 jpg to png

jpg format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ **해상도를 유지하며** png format으로 변환한다.

5.3 png to bmp

png format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ **해상도를 유지하며** bmp format으로 변환한다.

5.4 png to jpg

png format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ **해상도를 유지하며** jpg format으로 변환한다.

5.5 bmp to png

bmp format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ **해상도를 유지하며** png format으로 변환한다.

5.6 bmp to jpg

bmp format의 image를 ~~해상도와 컬러/흑백 속성을 유지하며~~ **해상도를 유지하며** jpg format으로 변환한다.

※ (5)의 SRS이 변경된 이유

⇒ 최근의 image들은 모두 gray-scale을 별도로 사용하지 않고 모두 RGB true color를 사용해 흑백을 표현한다. 따라서 해당 경향을 반영하여 우리는 흑백 속성을 사용하지 않고 컬러 속성만 사용하기로 결정하였고, 이는 test case set에도 반영되어 중간발표 1차와 중간발표 2차에 test case set이 변경되는 결과로 이어졌다. 해당 SRS의 변경은 그러한 내용을 반영한 것이다.

(6) 잘라내기

6.1 원하는 영역 자르기

left-top pixel과 right-bottom pixel을 주어 image 내의 원하는 image 영역만 잘라내고, 잘라낸 image 영역과 기존의 image를 함께 화면에 출력한다.

(7) 확대 및 축소

7.1 확대

image를 단계적으로 ~~150%~~ → 200% → 300% → 400% → 500% 까지 확대하여 화면에 출력한다.

7.1.1 화면 이동

image를 확대한 경우 출력 창 크기를 초과하면 화면을 이동하며 확대된 image를 살펴볼 수 있게 해준다.

7.1.2 축소

image를 단계적으로 확대한 경우 다시 단계적으로 축소하여 image를 화면에 출력한다.

※ (7)의 확대 SRS이 변경된 이유

⇒ GUI 상에서 확대 축소를 지원하지 않아 해상도 조절 기능으로 확대 축소를 구현하였고, 그 과정에서 150%를 도입하는데 설계 상의 문제가 있어 결국 중간발표 2차 ~ 최종 발표 사이에 부득이하게 변경할 수 밖에 없게 되었다.

(8) 저장 및 불러오기, 실행 취소 및 재실행

8.1 저장

사용자가 원하는 파일을 원하는 file name 및 file format으로 원하는 위치에 저장한다.

8.2 불러오기

사용자가 원하는 파일을 지정된 위치로부터 불러와 화면에 출력한다.

8.3 실행 취소

사용자가 실행한 동작을 취소시키고 한 단계 전으로 돌아가고, 해당 단계의 image를 화면에 출력한다.

8.4 재실행

사용자가 실행 취소한 동작을 재실행 하고, 해당 단계의 image를 화면에 출력한다.

2.2 Non-functional requirements

(1) Performance

⇒ 사용자가 원하는 기능을 입력한 후 **최대 5초 안에** 결과를 출력한다.

※ python을 사용하여 속도가 느린 점으로 인해 5초 안에 결과를 출력할 수 없는 경우가 존재한다.

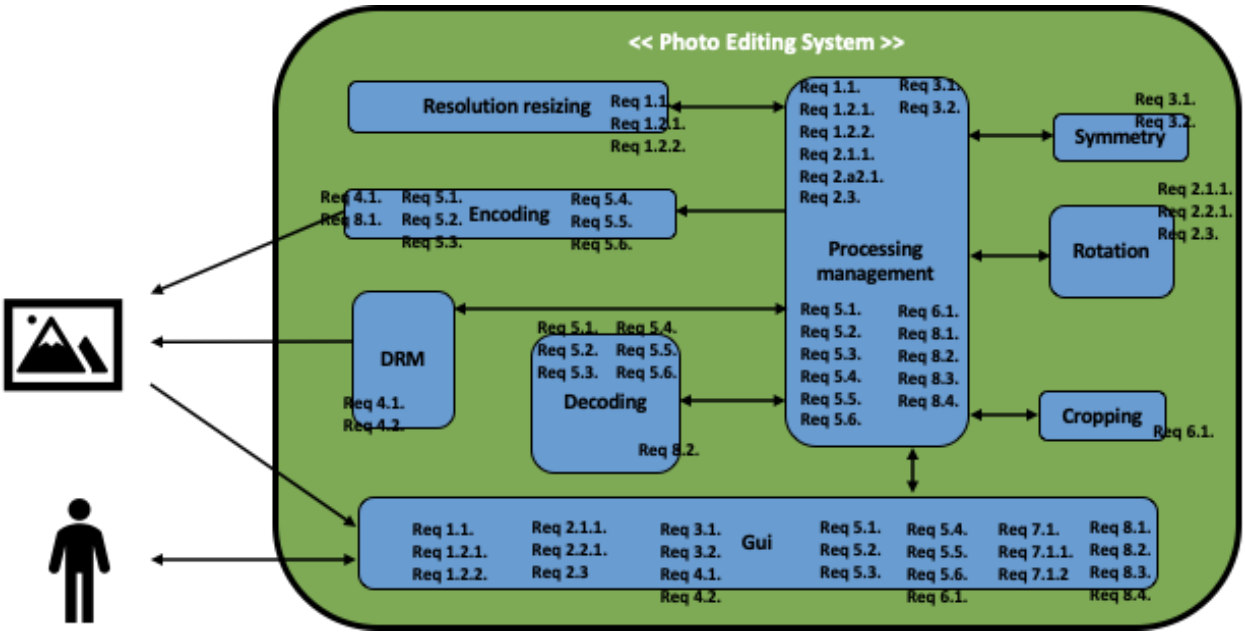
- JPEG decoding / 자유 각도 회전 및 해상도 확대의 일부 경우.

(2) Usability

⇒ 그림판과 같은 단축 아이콘을 사용한다. 또 해당 아이콘에 마우스를 올리면 제공하는 기능에 관한 메시지를 띄워 단축 아이콘 및 기능에 대한 사용자의 이해를 돕고 편의성을 제공한다.

3. SDS

3.1 Architecture Diagram



본 프로젝트는 여러 module들이 모여 하나의 독립된 프로그램을 구성하고, server - client와 같은 network 상에 기반을 둔 구조가 아니므로 프로젝트의 구조를 가장 잘 표현할 수 있는 디자인으로 Architecture Diagram을 제시하였다.

먼저 클라이언트가 Gui의 아이콘을 누르면 Processing Management로 연결되고 pm에서 해당 아이콘의 기능을 하는 클래스와 연결한다. 기능이 적용된 픽셀 데이터(RGB numpy array)를 pm으로 전달하고 pm은 다시 Gui에 전달하여 사용자가 원하는 이미지를 띄운다. 모든 기능의 수행은 Gui ⇒ pm ⇒ 기능 클래스 순서로 진행된다.

Gui에서 파일 불러오기 아이콘을 누르면 Processing Management 클래스가 생성되고 pm에서 이미지 디코딩을 위해 Decoding 클래스로 연결한다. 디코딩이 완료되면 다시 pm으로 픽셀 데이터(numpy)가 전달된다. 이후 pm에서는 기능에 따라 계속 현재 픽셀 데이터를 업데이트한다. 기능이 수행될 때마다 해당 기능의 클래스에 numpy를 인자로 전달하고 기능 완료 후에 다시 numpy를 return 받는다. 즉, Processing Management가 controller 역할을 한다.

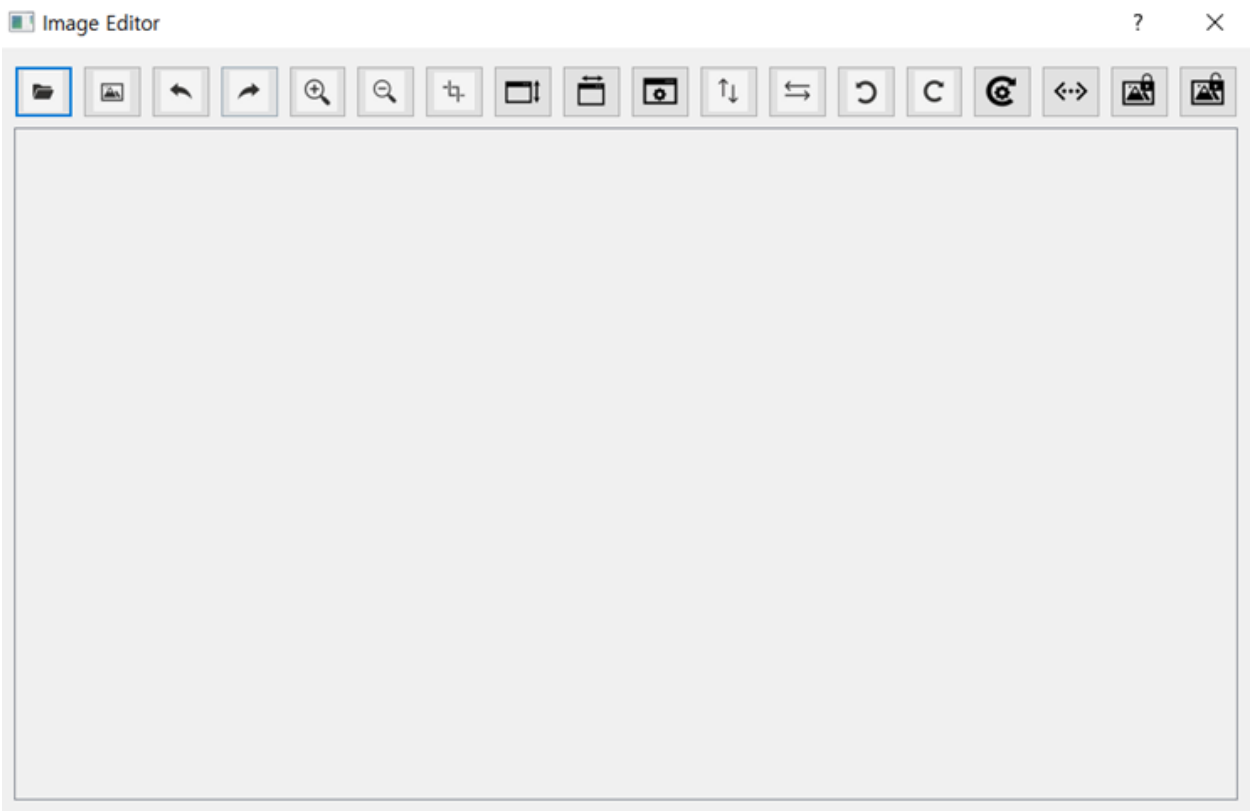
다이어그램에서 보이는 7가지의 기능 클래스 중 gui를 거치지 않고 결과를 도출하는 클래스가 2가지 있다. 저장하기 아이콘을 누르면 pm의 픽셀 데이터가 Encoding 클래스로 전달된다. 인코딩

결과는 다시 pm로 전달되지만 gui에 이미지를 띄우는 과정은 거치지 않는다. 대신 입력한 저장 경로에 이미지가 저장된 걸 확인할 수 있다. 파일 포맷 변환의 경우도 마찬가지다. pm에서 Encoding 클래스에 numpy와 변환할 파일 포맷을 인자로 전달한 후 완료 시 저장 경로에 이미지의 포맷이 바뀌어 저장된 것을 볼 수 있다.

Encoding 클래스와 마찬가지로 DRM 클래스도 결과가 gui를 통해 도출되지 않는다. 암호화는 이미 불러온 이미지에 대해서만 가능한 기능이다. gui를 통해 key를 입력하면 pm에서 처음에 불러온 이미지와 key를 DRM 클래스에 전달한다. DRM에서는 key값을 바탕으로 처음에 불러온 이미지를 암호화 후 원래 이미지의 경로에 저장한다. 복호화는 gui에서 입력한 파일 경로와 key를 DRM 모듈로 전달한다. DRM 클래스에서 key를 바탕으로 암호화된 이미지의 픽셀 데이터를 다시 복호화한 후 저장한다. 암호화, 복호화는 이미 저장된 이미지에 대해서만 가능하므로 gui 상에서만 변환된 이미지는 저장 후 그 이미지를 불러와 진행하면 된다.

나머지 회전, 자르기, 해상도 조절, 대칭 기능은 gui ⇒ pm ⇒ 기능 클래스 ⇒ pm ⇒ gui의 순서로 진행된다.

3.2 Prototype



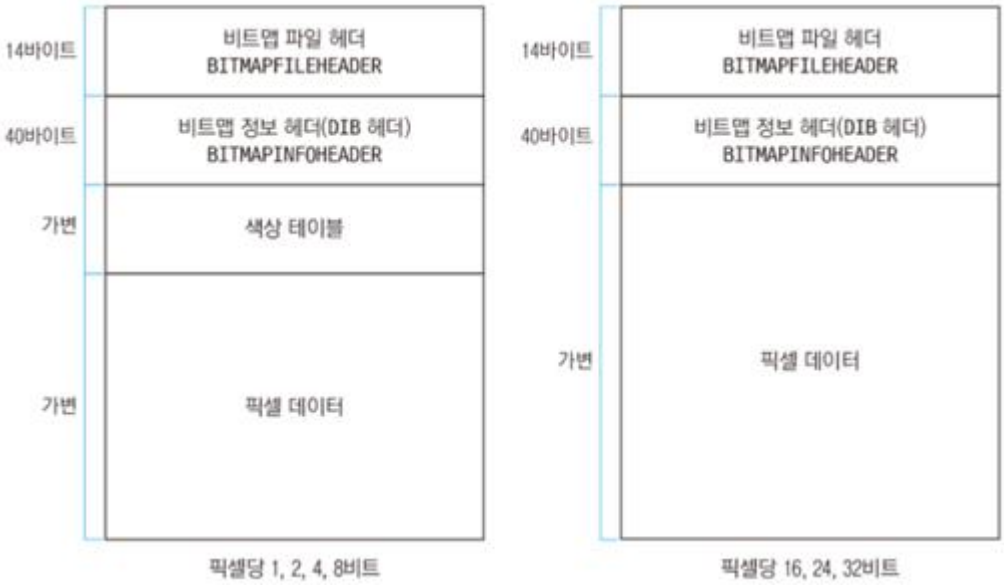
4. 구현 결과물 중 4.4 GUI에서 별도로 설명하였다.

4. 구현 결과물

4.1 포맷 변환

4.1.1 BMP

(1) BMP 구조



BMP는 비트맵 파일 헤더와 비트맵 정보 헤더(DIB 헤더), 색상 테이블, 픽셀 데이터로 구성된다. 픽셀당 1, 2, 4, 8비트의 경우 헤더 뒤에 색상 테이블에 따로 색상 정보를 저장하고 픽셀당 16, 24, 32 비트의 경우 색상 테이블을 갖지 않는다. 본 프로젝트에서 bmp 이미지는 픽셀당 24(RGB)비트 또는 32(RGBA)비트를 가지므로 색상 테이블은 처리하지 않는다 .

1.1 BMP 파일 헤더

멤버	크기(byte)	목적
bfType	2	BMP 파일을 식별하는 데 쓰이는 매직 넘버: 0x42 0x4D
bfSize	4	BMP 파일 크기 (바이트 단위)
bfReserved1	2	현재는 사용하지 않는 미래를 위한 예약 공간

bfReserved2	2	현재는 사용하지 않는 미래를 위한 예약 공간
bfOffBits	4	비트맵 데이터의 시작 위치 (바이트 단위)

이미지의 데이터를 읽어올 때 제일 먼저 확인해야하는 멤버가 bfType이다. bfType(2 bytes)이 B와 M에 대한 ASCII 코드인 0x42와 0x4D를 가져야 해당 이미지가 BMP임을 식별할 수 있다. bfSize는 비트맵 파일 헤더(14 bytes)와 DIB 헤더(40 bytes), 색상테이블, 픽셀 데이터의 크기를 모두 더한 값이다. bfReserved1과 bfReserved2는 예약공간이며 현재는 사용되지 않아서 항상 0을 가진다. bfOffBits는 비트맵 데이터(픽셀 데이터)의 시작 위치를 가리킨다.

1.2 DIB 헤더(비트맵 정보 헤더)

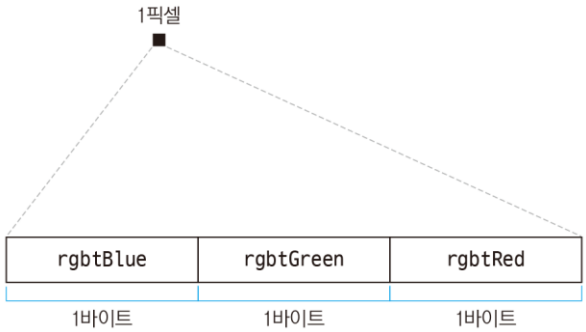
멤버	크기(byte)	목적
biSize	4	이 헤더의 크기
biWidth	4	비트맵 이미지의 가로 크기
biHeight	4	비트맵 이미지의 세로 크기
biPlanes	2	사용하는 색상판(color plane)의 수
biBitForPixel	2	픽셀 하나를 표현하는 비트 수
biCompression	4	압축 방식
biSizeImage	4	그림 크기. 압축되지 않은 비트맵 데이터의 크기
biXPelsPerMeter	4	그림의 가로 해상도
biYPelsPerMeter	4	그림의 세로 해상도
biClrUsed	4	색상 테이블에서 실제 사용되는 색상 수
biClrImportant	4	비트맵을 표현하기 위해 필요한 색상 인덱스 수

biSize는 BMP 헤더의 크기이며 보통 54바이트를 갖는데 헤더와 픽셀 데이터 사이에 빈 공간이 있는 이미지의 경우 138바이트를 갖는다. biWidth와 biHeight는 각각 이미지의 가로 픽셀 수, 세로 픽셀 수를 의미한다. 색상판의 수(biPlanes)는 항상 1로 설정되어있다. 비트맵은 압축을 하지 않으므로 biCompression은 0을 갖는다. biXPelsPerMeter과 biYPelsPerMeter는 각각 이미지의 가로 미터당

픽셀, 세로 미터당 픽셀 값을 갖는다. 본 프로젝트의 BMP 이미지는 색상 테이블을 처리하지 않으므로 biClrUsed와 biClrImportant는 0이다.

1.3 픽셀 구조체

멤버	크기(byte)	설명
rgbBlue	1	파랑
rgbGreen	1	초록
rgbRed	1	빨강
rgbaAlpha	1	알파



픽셀 구조체는 RGBA와 RGB로 나뉘는데 왼쪽이 RGBA 구조체이고 오른쪽 그림이 RGB 구조체이다. B, G, R, A 모두 한 바이트의 크기를 갖는다. RGB의 경우 BGR 순서로 저장되고 RGBA의 경우 BGRA 순서로 저장된다.

(2) BMP decoder

2.1 bmp_decoder 클래스

bmp_decoder 클래스 생성 시 filepath를 인자로 받아 BMP 파일 헤더와 DIB 헤더를 바이트 단위로 읽어와서 헤더 멤버를 각각 변수에 저장한다.

2.2 주요 함수

1) check_format

check_format 함수에서 BMP 이미지의 매직 넘버인 'BM' (ASCII 0x42, 0x4D)가 맞는지 파일의 포맷을 확인한다.

2) cal_width

BMP 이미지의 경우 width가 반드시 4의 배수이다. 실제 width가 4의 배수가 아니더라도 4의 배수가 되도록 각 열의 끝을 0(byte)로 채우기 때문이다. 예를 들어 width가 29 bytes라면 각 열의 끝에 000(3 bytes)가 추가되어 마치 width가 32 bytes인 것처럼 저장된다. cal_width 함수에서 width가 4의 배수가 아닌 경우 뒤에 0(byte)가 얼마나 추가되었는지를 계산하여 저장하고 color_numpy 함수에서 색상 정보를 읽어올 때 사용한다.

3) jump

픽셀 데이터의 시작 위치로 jump하는 함수이다. 픽셀 데이터는 54 바이트 또는 138 바이트부터 시작하는데 후자의 바이트의 경우 54 바이트부터 137바이트까지 있는 빈 공간을 읽어서 다음 읽을 위치가 픽셀 데이터의 시작이 되도록 한다.

4) color_numpy

BMP 이미지는 RGB와 RGBA의 경우로 나누어져 색상 정보를 저장한다. 헤더로부터 읽어온 멤버 중 bit for pixel의 값이 24라면 RGB이고 32이면 RGBA임을 알 수있다.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

BMP 이미지는 리틀 엔디안 방식으로 픽셀 데이터를 저장하기 때문에 마지막 행부터 첫 행 순서로 읽고 픽셀도 BGR, BGRA로 읽게된다. 즉, 위의 4x4 픽셀 데이터는 13번부터 읽을 수 있고 13~16번을 읽은 후 9~12번, 5~8번, 1~4번으로 이어진다.

RGB의 경우 모든 픽셀 데이터를 리틀 엔디안 순서로 읽어와 픽셀마다 B, G, R 값을 저장한 후 이를 모두 순서에 맞게 numpy 배열에 저장한다. 만약, width가 4의 배수가 아니라면 모든 열의 끝에 0이 저장되어 있으므로 이를 읽고 무시하여 read 포인트가 다른 열의 시작점으로 가도록 만든다.

RGBA의 경우 모든 픽셀 데이터를 리틀 엔디안 순서로 읽어와 픽셀마다 B, G, R, A 값을 저장한 후 이를 모두 순서에 맞게 numpy 배열에 저장한다. RGBA는 width가 항상 4의 배수이므로 모든 값을 numpy에 저장하면 된다.

이렇게 저장한 numpy 값이 BMP 이미지를 decoding한 결과이다.

(3) BMP encoder

3.1 bmp_encoder 클래스

멤버	값
bfType	BM
bfSize	계산 필요

bfReserved1	0
bfReserved2	0
bfOffBits	54

멤버	값
biSize	40
biWidth	계산
biHeight	계산
biPlanes	1
biBitForPixel	24 or 32
biCompression	0
biSizeImage	계산
biXPelsPerMeter	0
biYPelsPerMeter	0
biClrUsed	0
biClrImportant	0

헤더 멤버 중 일부를 고정한다. 먼저, 파일 format을 ‘BM’으로 세팅한다. 예약 공간 reserved1, 2는
사용하지 않으므로 0으로 설정한다. 픽셀 데이터 시작 주소인 offset을 54로, DIB header size를 40으로 고정하여 헤더와 픽셀 데이터 사이에 빈 공간이 없는 파일로 전제한다. 본 프로젝트에서 BMP는 color palette를 사용하지 않으므로 color plane의 값을 1로하고 그에따라 ColorUsed, ColorImportant 값도 0으로 정한다. BMP는 압축하지 않으므로 압축방식 compression을 0으로 고정한다. 이미지 encoding에 영향을 미치지 않는 미터당 가로 픽셀 수 hor과 미터당 세로 픽셀 수 ver는 0으로 한다.

bmp_encoder는 클래스 생성 시 이미지의 색상정보를 갖는 numpy와 color_format(RGB, RGBA)을 인자로 받는다. numpy의 shape를 이용해 이미지의 width와 height를 저장한다.

3.2 주요 함수

1) CalHeader

정해지지 않은 헤더 멤버의 값을 넣어준다. color_format이 RGB인지 RGBA인지에 따라 bit for pixel이 24이거나 32로 정해진다. raw size는 width*height*bfp + (채워진 0의 수)로 계산하고 file size는 raw size+offset으로 계산한다.

2) PutHeader

저장할 파일에 헤더 멤버의 값을 차례로 쓴다.

3) PutNumpy

3차원 배열 numpy의 모든 행, 열의 RGB 또는 RGBA의 값을 리틀 엔디안 순서로 파일에 쓴다. RGB의 경우 width가 4의 배수가 아니라면 모든 열의 끝에 0(byte)를 넣어 width가 4의 배수가 되도록 padding한다. RGBA의 경우 A의 값을 0xFF로 고정하여 넣는다.

4) write_file

filepath를 인자로 받아 해당 경로에 새로운 파일을 만들고 BMP의 매직넘버 'BM'과 헤더,

픽셀 데이터 값을 차례로 파일에 쓰는 작업을 통해 BMP 이미지를 생성한다.

4.1.2 PNG

(1)png 구조

1.1 png 시그니처

89 50 4E 47 0D 0A 1A -- png 이미지 시작 부분에 저장되어 있는 데이터입니다. Png 이미지 란 것을 알려줍니다.

1.2 png 기본 구조

Chunk

Length	4 bytes
Chunk type	4 bytes
Chunk data	(Length) byte
CRC	4 byte

Png는 모든 정보가 chunk 단위로 저장되어 있습니다. 그리고 chunk의 기본구조는 위 표와 같습니다. Png를 구성하는 chunk는 여러 개가 존재하지만 필수 적인 chunk type은 IHDR, IDAT, IEND입니다.

1.2.1 IHDR

<table>
<tr><td>Length</td><td>00 00 00 0D (13)</td></tr>
<tr><td>Chunk type</td><td>IHDR</td></tr>
<tr><td>Chunk data
<table>
<tr><td>Width</td><td>4 byte</td></tr>
<tr><td>Height</td><td>4 byte</td></tr>
<tr><td>Bit depth</td><td>1 byte</td></tr>
<tr><td>Color type</td><td>1 byte</td></tr>
<tr><td>Compression</td><td>1 byte</td></tr>
<tr><td>Filter</td><td>1 byte</td></tr>
<tr><td>Interlace</td><td>1 byte</td></tr>
</table></td><td>Length -- 13 byte</td></tr>
<tr><td>CRC</td><td>4 byte</td></tr>
</table>

IHDR의 chunk data는 13byte이며 그 안에는 이미지의 폭과 높이를 저장하는 width와 height, 이미지의 색상정보를 가지는 color type, 색상정보에 사용한 비트를 알려주는 bit depth가 있습니다

Image type	Color type	Bit depth	
Grayscale	0	1,2,4,8,16	그레이 스케일 값
True color	2	8,16	RGB 값
Indexed color	3	1,2,4,8	팔레트 값
Grayscale with alpha	4	8,16	그레이 스케일 + alpha 값
True color with alpha	6	8,16	RGBA 값

Color type 과 bit depth은 다음과 같이 종류와 값이 정해져 있으며 true color with alpha, true color을 이용한 png을 구현 했습니다. 압축 방법을 저장하는 compression, 필터 종류를 알려주는 filter, interlace 종류를 알려주는 interlace가 저장되어 있습니다. 그 중에서 compression, filter는 방법이 한가지이며 0을 저장합니다. compression의 경우 deflate 압축을 합니다. Interlace 또한 방법이 한가지 만 존재하여 사용 시에는 1, 사용을 하지 않을 시에는 0을 저장합니다.

1.2.2 IDAT

Length	
Chunk type	IDAT
Chunk data	Length ()
CRC	4 byte

IDAT chunk는 이미지 데이터 값이 저장되어 있습니다. 보통 65536 바이트가 저장되지만 유동적입니다. Png는 이미지를 저장할 때 filter 연산을 한 후에 compression을 합니다. 따라서 ID

AT chunk에 있는 데이터는 rgb 나 rgba 값이 아닌 알 수 없는 값이며, 데이터의 일부분으로 이미지의 한 부분을 보여줄 수 없습니다.

1.2.2.1 filter

Png의 필터링의 경우 각 scanline(행) 마다 합니다. 총 5가지의 필터 타입이 있습니다. 어떤 필터 타입을 적용하는 지는 휴리스틱 적으로 결정합니다. 사용한 필터 타입에 대한 정보는 각 scanline 맨 앞에 1byte를 이용하여 저장합니다.

c	b
a	x

필터링 하는 기준은 바이트이며 위 표에서 필터링 할 바이트는 x 라고 하겠습니다.

Type	Name	Function
0	None	F(x) = O(x)
1	Sub	F(x) =O(x)-O(a)
2	Up	F(x)=O(x)-O(b)
3	Average	F(x)=O(x)-floor(O(a)+O(b))/2
4	Paeth	F(x)=O(x)-predictor(O(a),O(b),O(c))

F(x) 는 필터링 한 x 이며 O() 는 원래 바이트 값을 뜻합니다. 다음 표와 같은 필터 타입 5가지를 이용하여 필터링을 합니다.

1.2.3 IEND

Length	00 00 00 00 (0 byte)
Chunk type	IEND

Chunk data	0 byte
CRC	4 byte

IEND chunk 는 데이터는 없는 chunk 로 png 이미지의 끝을 알려주기 존재하는 chunk 입니다.

(2)png decoder

1.1 pngdecoder 자료구조

pngchunks	Chunks	IHDR	width
			Height
			bitdepth
			colortype
			compression
			Filter
			interlace
		IDAT	data
		IEND	

1.2 주요 함수

1) Read_chunks(png image)

→ 이미지 파일에서 png 시그니처 '\x89PNG\r\n\x1a\n' 와 동일 하면 디코딩을 시작합니다.

→ Chunk 별로 데이터를 저장합니다.

*IHDR 의 경우 width, height, bitdepth, colortype, compression, filter, interlace를 나눠 저장합니다.

* IDAT 의 경우 data 부분에 이미지 데이터를 저장합니다.

2) decompress (pngchunks)

→ IDAT의 data 를 zilb 라이브러리를 이용하여 압축을 해제합니다.

→ 압축 해제한 데이터를 de-filter 를 합니다.

→ 복원된 이미지 데이터를 numpy 배열에 저장합니다.

(3)png encoder

1.1자료 구조

png class

{

self.w = width

self.h = height

self.bit_depth = 8

self.compression_level = 7

self.filter_type = 0

self._interlace = 0

self._compression = 0

}

→ Numpy 배열을 받아 png 파일로 인코딩 할 때 width 와 heigh 값만 input을 받고 나머지 IHDR 에 들어갈 정보들은 고정 시켰습니다.

1.2 주요 함수

1) get_chunks(data)

→ Numpy 배열을 list 로 바꾼 데이터를 입력 받습니다.

→ Png 시그니처를 write 하고 IHDR, IDAT, IEND 를 저장합니다.

2) fil_compress(data)

→ Data를 필터링을 하고 zilb 라이브러리를 이용한 압축을 하는 함수입니다.

→ IDAT chunk를 만들기전에 먼저 실행 후, 필터링과 압축이 된 data를 IDAT chunk에 저장합니다.

4.1.3 JPG

1. JPG 구조

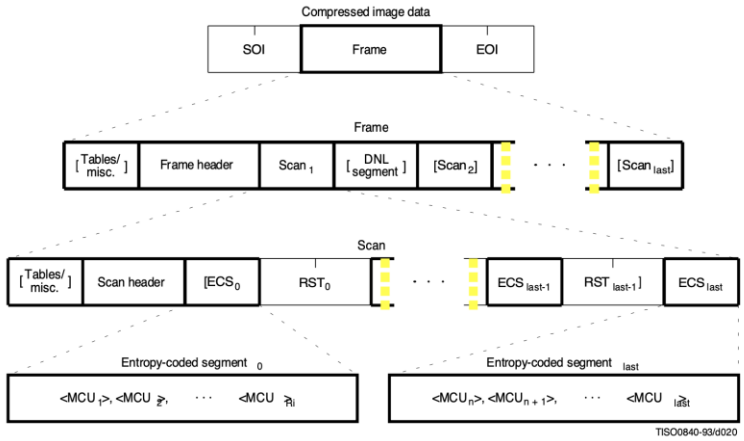


Figure B.2 – Syntax for sequential DCT-based, progressive DCT-based, and lossless modes of operation

위의 그림은 전체적인 JPEG format의 image 파일에 대한 구조이다.
이후 JPEG의 구조를 알기 위해서는 미리 사전에 정의된 표현을 알아야만 한다. (아래는 예시)

- 정사각형 : 1Byte
- 따라서 정사각형 2개가 모인 것은 2 Byte (SOFn), 가운데 점선인 것은 4Bit씩을 의미한다 (H1, V1)

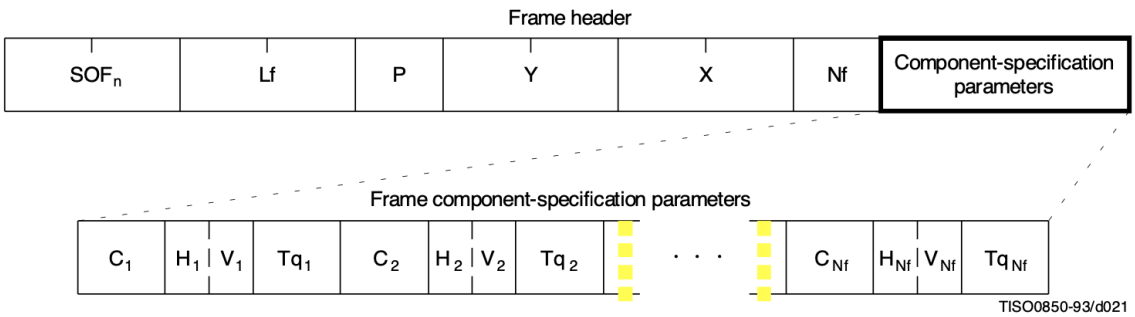


Figure B.3 – Frame header syntax

본 프로젝트에서는 JPEG의 종류 중 baseline sequential huffman coding / interleaving 방식에 대한 decoding 및 encoding을 다루었고, 세부적인 다른 종류에 대해서는 2. JPG 종류에서 언급하도록 할 것이다.
해당 방식에서 JPEG decoder / encoder logic을 설명하기 위해서 반드시 알아야만 하는 header 및 marker 는 다음과 같다.

- SOF (Start Of Frame)
- JFIF (JPEG File Interchange Format)
- DHT (Define Huffman Table)
- DQT (Define Quantum Table)
- SOS (Start Of Scan)
- RST (Restart)
- DRI (Define Restart Interval)
- EOI (End Of Image)
- SOI (Start Of Image)

1) SOF marker

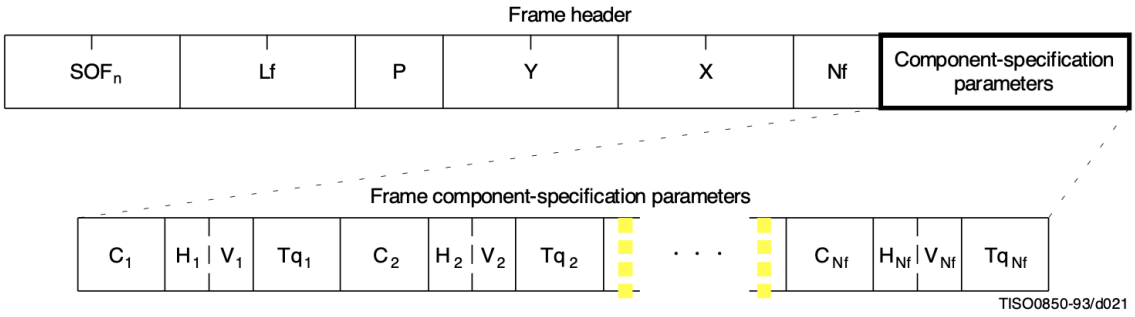


Figure B.3 – Frame header syntax

- SOFn : 2Byte.
  ⇒ JPEG의 인코딩 및 디코딩 방식에 따라 marker 값이 달라진다. 대표적으로 다음과 같은 marker 값이 있다.
  - SOF0 : Baseline DCT
  - SOF1 : Extended sequential DCT - huffman coding
  - SOF2 : Progressive DCT - huffman coding
  - SOF3 : Lossless sequential - huffman coding
  - SOF9 : Extended sequential DCT - arithmetic coding
  - SOF10 : Progressive DCT - arithmetic coding
  - SOF11 : Lossless sequential - arithmetic coding

  이 중 사용하는 것은 SOF0, BaseLine DCT이고 해당되는 marker 값은 0xFFC0이다.

- Lf : 2Byte
  ⇒ 모든 marker 다음에는 2Byte의 Length Field가 붙는다. 이 Length Field는 자신의 길이(2 Byte) 역시 포함한다.

- P : 1Byte

⇒ Data Precision으로, 한 color 당 몇 bit로 표현될지를 결정한다. 대부분의 경우 8bit 이므로 8이라는 값이 들어온다.

- Y : 2 Byte
  ⇒ jpeg image의 height pixel 해상도를 나타낸다.

- X : 2 Byte
  ⇒ jpeg image의 width pixel 해상도를 나타낸다.

- Nf : 1 Byte
  ⇒ Number of Component를 의미하며, 8bit 이므로 이론상 255개의 component가 가능하지만
  JFIF header에서 3개만 사용하도록 정의하고 있다. Y component, Cb component, Cr component이다.

- Component Specification (보통 3개의 component에 대한 값이 들어온다.)
  - C : 1 Byte
    ⇒ Component ID. 1부터 시작하며 보통 Y = 1, Cb = 2, Cr = 3이다.
  - H : 상위 4Bit
    ⇒ Horizontal sampling. 보통 1, 2의 값을 가지며 sampling 단계에서 사용된다.
  - V : 하위 4Bit
    ⇒ Vertical sampling. 보통 1, 2의 값을 가지며 sampling 단계에서 사용된다.
  - Tq : 1 Byte
    ⇒ Component 별로 사용하는 Quantum table의 ID를 정의한다.

2) JFIF marker

JFIF APP0 marker segment		
Field	**Size (bytes)**	**Description**
APP0 marker	2	`FF E0`
Length	2	Length of segment excluding APP0 marker
Identifier	5	`4A 46 49 46 00` = "JFIF" in ASCII, terminated by a null byte
JFIF version	2	First byte for major version, second byte for minor version (`01 02` for 1.02)
Density units	1	Units for the following pixel density fields • `00` : No units; width:height pixel aspect ratio = Ydensity:Xdensity • `01` : Pixels per inch (2.54 cm) • `02` : Pixels per centimeter
Xdensity	2	Horizontal pixel density. Must not be zero
Ydensity	2	Vertical pixel density. Must not be zero
Xthumbnail	1	Horizontal pixel count of the following embedded RGB thumbnail. May be zero
Ythumbnail	1	Vertical pixel count of the following embedded RGB thumbnail. May be zero
Thumbnail data	3 × *n*	Uncompressed 24 bit RGB (8 bits per color channel) raster thumbnail data in the order R0, G0, B0, ... Rn-1, Gn-1, Bn-1; with *n* = Xthumbnail × Ythumbnail

별도의 format이므로 JPEG ITU-81 표준 문서에는 제시되어 있지 않다. 영문 위키피디아에서 field의 값을 가지고 왔다.

- APP0 marker : 2 Byte
  ⇒ JFIF, Exif와 같이 JPEG과 관련된 추가적인 application data를 담고 있는 marker이다. 보통 SOI(Start Of Image, 2 Byte, 0xFFD8) 뒤에 바로 온다.

- Length : 2 Byte
  ⇒ 항상 marker 뒤에는 자기 자신 (Length Field)을 포함한 marker 전체의 길이를 나타내는 Length Field가 온다.

- Identifier : 5 Byte
  ⇒ JFIF임을 나타내는 0x 4A 46 49 46 00이 온다. 이는 ASCII 문자로 JFIF\0을 의미한다.

- JFIF version : 1 Byte + 1 Byte
  ⇒ JFIF extended의 경우 01.02로 표현되지만 일반적인 경우 01.01로 표현된다.

- Density Units : 1 Byte
  ⇒ 보통 00 혹은 01이 사용된다. pixel의 밀도를 나타내는 단위를 설정할 때 사용한다.

- X density : 2 Byte
- Y density : 2 Byte
  ⇒ 둘 다 pixel ratio를 정할 때 사용되며 보통 72를 나타내는 hexadecimal 값이 온다.

- X thumbnail : 1 Byte
- Y thumbnail : 1 Byte
- Thumbnail data : 3 * X thumbnail * Y thumbnail Bytes.
  ⇒ thumbnail을 담는 field이나 대부분은 이 field에 thumbnail을 담지 않아 0x00으로 표현된다.

3) DHT marker

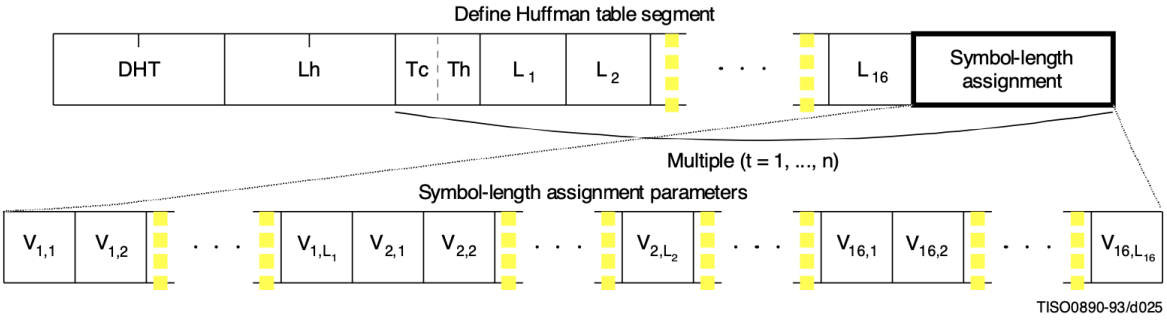


Figure B.7 – Huffman table syntax

- DHT : 2 Byte
  ⇒ Huffman table임을 나타내는 marker 값으로 보통 0xFFC4가 온다. huffman table 경우 일반적으로 4개가 사용된다. AC chroma & Lumi, DC chroma & Lumi coefficient에 각각 하나씩.

- Lh : 2 Byte
  ⇒ DHT marker의 길이를 나타내는 field

- Tc : 4 Bit
  ⇒ A′ C′ component에 사용하는 Huffman table인지, D′ C′ component에 사용하는 Huffman table인지 알려준다. 보통 0이면 DC, 1이면 AC에 사용된다.
- Th : 4 Bit
  ⇒ huffman table의 number를 알려준다. 보통 0 ~ 3의 값이 사용되며, Tc + Th를 합쳐 1 Byte를 huffman table의 Identifier로 본다.

- L1 ~ L16 : 1 Byte
  ⇒ 해당하는 bit Length를 갖는 Symbol들의 개수를 나타낸다. huffman의 경우 AC huffman이 최대로 16 bit 길이의 Symbol을 사용하므로 최대 L1 ~ L16까지 총 16개의 field가 존재한다.

- Symbol-length assignment : n Bytes
  ⇒ 위에서 L1 ~ L16에 해당하는 Symbol들이 실제로 들어온다. AC의 경우 보통 162개, DC의 경우 12개 정도가 온다.

4) DQT marker

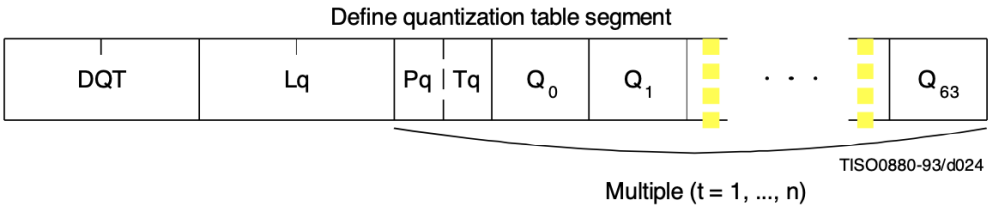


Figure B.6 – Quantization table syntax

- DQT : 2 Byte
  ⇒ DQT marker 임을 나타내는 값. 보통 0xFFDB가 온다. Quantum table의 경우 Y component와 Cb & Cr의 chroma component를 위해 2개가 사용된다.

- Lq : 2 Byte
  ⇒ DQT marker의 length를 알려주는 field이다.

- Pq : 4 Bit
  ⇒ Quantum table의 한 요소에 필요한 bit수를 결정한다. 0이면 8 Bit, 1이면 16 Bit이지만 보통 0으로 1 Byte 씩 사용한다.

- Tq : 4 Bit
  ⇒ Quantum table의 number를 의미한다. 해당 number를 가지고 SOF marker에서 명시된 component들이 사용하는 quantum table이 결정되는 것이다.

- Q0 ~ Q63 : 1 Byte
  ⇒ Quantum table안의 값. 중요한 것은 table을 채우는 순서이다. 순서는 아래와 같다.

0	2	5	9	14	20	27	35
1	4	8	13	19	26	34	42
3	7	12	18	25	33	41	48
6	11	17	24	32	40	47	53
10	16	23	31	39	46	52	57
15	22	30	38	45	51	56	60
21	29	37	44	50	55	59	62
28	36	43	49	54	58	61	63

5) SOS marker

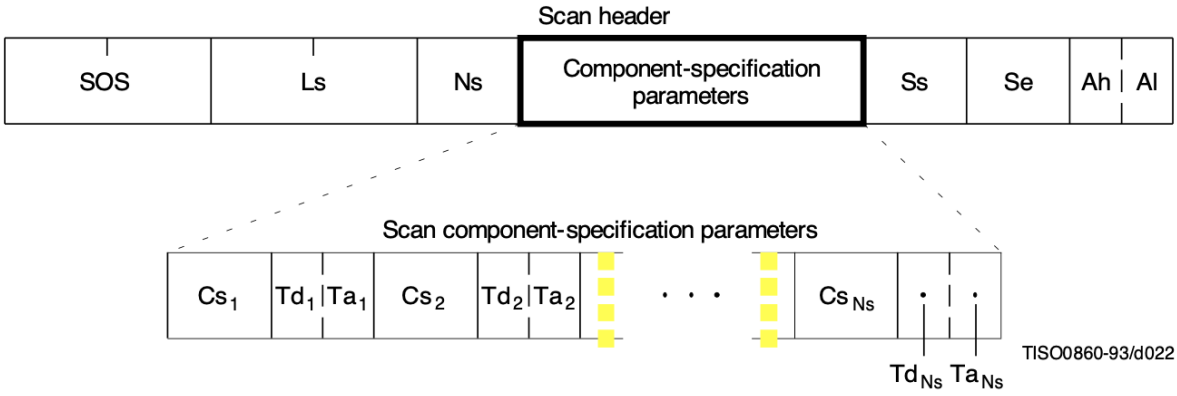


Figure B.4 – Scan header syntax

- SOS : 2 Byte
  ⇒ Scan 이 시작됨을 알려주는 marker로, baseline sequential interleaving 방식의 경우 하나만 존재하지만 interleaving 방식의 경우 component 별로 scan을 하므로 3개가 존재하게 된다. 또한 progressive 및 hierarchy 방식 또한 다수의 Start Of Scan marker가 존재할 수 있다.
  해당 marker를 나타내는 값은 0xFFDA 이다.

- Ls : 2 Byte
  ⇒ 해당 Start Of Scan field의 길이를 나타내는 field이다.

- Ns : 1 Byte
  ⇒ 이번에 scan 되는 component의 개수를 의미한다. 1이면 scan 되는 component가 1개이므로 non-interleaving 방식이라 하고, 2 이상이면 다수의 component가 scan 되는 것이므로 interleaving 방식이라고 한다. gray-scale의 경우 이 값이 1로 설정되지만 요즘은 대부분 RGB True Color로 표현하기 때문에 대부분 2 이상의 값을 갖는다.

- Ss : 1 Byte
  ⇒ Spectral Selection start 라 하여, progressive 방식의 encoding 및 decoding 에서 사용되는 field이다. 보통 baseline sequential의 경우 0 ~ 63개의 coefficient 단위로 인코딩을 진행하지만 progressive의 경우에는 0, 1 ~ 10, 11 ~ 63 이런 식으로 끊어서 encoding 및 decoding을 진행한다. 이 경우 이번 scan의 시작 지점을 몇 번째 coefficient로 할지 나타내는 field이다.

- Se : 1 Byte
  ⇒ 마찬가지로 위에서 말한 Spectral Selection end 지점을 결정하는 field이다.

- Ah : 4 Bit

⇒ Successive approximation에 사용되는 bit이다. progressive scan의 경우 multiple scan 이 되므로, coefficient의 상위 몇 bit만 먼저 encoding 및 decoding 하여 중요한 정보를 우선적으로 전달할 수 있다. 이 때 상위 몇 bit를 먼저 encoding 및 decoding 할 지 결정하는 field이다.

- Al : 4 Bit
  ⇒ Successive approximation의 나머지 하위 bit를 scan하도록 정하는 field이다.

- Component specification : N byte
  - Cs : 1 Byte
    ⇒ 이번에 scan 되는 component의 ID가 들어온다.
  - Td : 4 Bit
    ⇒ 이번에 scan 되는 component가 사용할 DC Huffman table의 number가 들어온다.
  - Ta : 4 Bit
    ⇒ 이번에 scan 되는 component가 사용할 AC Huffman table의 number가 들어온다.

(6) DRI & RST marker

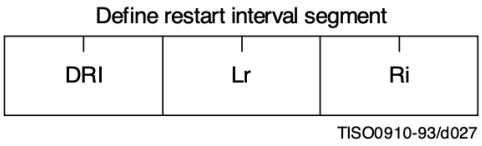


Figure B.9 – Restart interval definition syntax

- DRI : 2 Byte
  ⇒ JPEG의 경우 non-interleaving 방식, 혹은 progressive와 같은 Multiple scan의 경우, scan을 할 때 MCU(Minimum Coded Unit) segment 단위로 진행되지만 가끔씩 이 segment가 error가 생기는 경우가 있다. 이 때 재 scan 하도록 interval을 지정하거나, 혹은 multi thread 를 사용하여 compression & decompression을 지원하기 위해 사용한다. 해당 marker의 값은 0xFFDD이다.

- Lr : 2 Byte
  ⇒ 마찬가지로 2 Byte 길이의 DRI marker 의 길이를 나타내는 field이다. 보통은 4이다.

- Ri : 2 Byte

⇒ Interval을 정하는 field다. 보통은 Width를 16으로 나눈 값이 들어온다. 이는 encoding & decoding의 sampling 및 run length encoding의 특성 때문이다.

- RST : 2 Byte

  ⇒ MCU segment 사이에 들어가는, 0xFFD0 ~ 0xFFD7의 값을 갖는 2 Byte marker이다. modulo 8 연산을 하여 계속해서 0xFFD0 ~ 0xFFD7의 marker가 순환하여 사용된다.

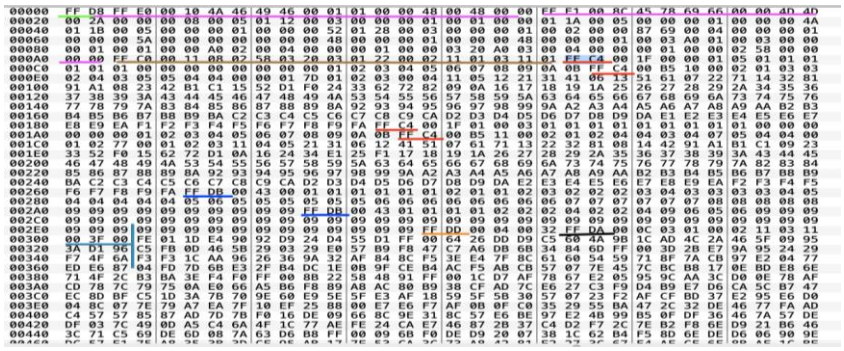
(7) SOI & EOI marker

- SOI : 2 Byte

  ⇒ Start Of Image. 사실상 JPEG의 file signature에 해당하며 0xFFD8 값을 갖는다.

- EOI : 2 Byte

  ⇒ End Of image. JPEG image의 마지막에 오며, RST marker를 사용한다면 마지막에는 RST marker 대신에 사용되기도 한다.



[실제 marker 들이 들어가 있는 JPEG file의 header.]

2. JPG 종류

(1) JPG의 종류는 아래와 같이 크게 4가지로 분류할 수 있다. 각 종류별로 실행되는 compression & decompression 옵션이 대개 정해져 있으며세부적으로 또 나누어지게 된다.

Baseline process (required for all DCT-based decoders)
• DCT-based process • Source image: 8-bit samples within each component • Sequential • Huffman coding: 2 AC and 2 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Extended DCT-based processes
• DCT-based process • Source image: 8-bit or 12-bit samples • Sequential or progressive • Huffman or arithmetic coding: 4 AC and 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Lossless processes
• Predictive process (not DCT-based) • Source image: P-bit samples ($2 \leq P \leq 16$) • Sequential • Huffman or arithmetic coding: 4 DC tables • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

Hierarchical processes
• Multiple frames (non-differential and differential) • Uses extended DCT-based or lossless processes • Decoders shall process scans with 1, 2, 3, and 4 components • Interleaved and non-interleaved scans

(2) 세부적인 구조

- baseline sequential process

1) baseline sequential;
2) extended sequential, Huffman coding, 8-bit sample precision;
3) extended sequential, arithmetic coding, 8-bit sample precision;
4) extended sequential, Huffman coding, 12-bit sample precision;
5) extended sequential, arithmetic coding, 12-bit sample precision.

- progressive process

1) spectral selection only, Huffman coding, 8-bit sample precision;
2) spectral selection only, arithmetic coding, 8-bit sample precision;
3) full progression, Huffman coding, 8-bit sample precision;
4) full progression, arithmetic coding, 8-bit sample precision;
5) spectral selection only, Huffman coding, 12-bit sample precision;
6) spectral selection only, arithmetic coding, 12-bit sample precision;
7) full progression, Huffman coding, 12-bit sample precision;
8) full progression, arithmetic coding, 12-bit sample precision.

- Lossless process

1) lossless processes with Huffman coding;
2) lossless processes with arithmetic coding.

- Hierarchical process

Table J.1 – Coding processes for hierarchical mode

Process	Non-differential frame specification	
1	Extended sequential DCT, Huffman, 8-bit	Annex F, process 2
2	Extended sequential DCT, arithmetic, 8-bit	Annex F, process 3
3	Extended sequential DCT, Huffman, 12-bit	Annex F, process 4
4	Extended sequential DCT, arithmetic, 12-bit	Annex F, process 5
5	Spectral selection only, Huffman, 8-bit	Annex G, process 1
6	Spectral selection only, arithmetic, 8-bit	Annex G, process 2
7	Full progression, Huffman, 8-bit	Annex G, process 3
8	Full progression, arithmetic, 8-bit	Annex G, process 4
9	Spectral selection only, Huffman, 12-bit	Annex G, process 5
10	Spectral selection only, arithmetic, 12-bit	Annex G, process 6
11	Full progression, Huffman, 12-bit	Annex G, process 7
12	Full progression, arithmetic, 12-bit	Annex G, process 8
13	Lossless, Huffman, 2 through 16 bits	Annex H, process 1
14	Lossless, arithmetic, 2 through 16 bits	Annex H, process 2

이와 같이 약 30여개 가까운 option이 있으며, 가장 많이 사용되는 것은 huffman coding 방식을 사용한 baseline sequential process이다. progressive process의 경우 web에서 전송될 때 사용되며, arithmetic coding의 경우 사용할 때마다 YBM에 사용료를 지불해야 하므로 huffman coding 방식을 많이 사용한다. 일반적으로 JPEG image를 볼 때 도중에 image가 끊겨 회색으로 나머지가 표시된다면 baseline sequential 방식을 사용하는 것이다.

3. JPG decoder의 작동 방식
   (1) 먼저 header를 parsing 하여, 필요한 Huffman table 및 Quantum table을 추출하여 Dictionary data type으로 저장한다.

(2) SOF header를 parsing 하여 component 별로 해상도와 V samping, H sampling 값, 그리고 사용하는 Quantun table ID를 확인한다.

(3) SOS header를 parsing 하여 component 별로 사용하는 DC huffman table 값과 AC huffman table 값을 확인한다.

(4) 각 component 별로 preprocession 처리를 한다.

- JPEG의 경우 RGB Color space를 YCbCr Color space로 바꾸어서 인코딩을 하므로, 디코딩 역시 당연히 Y, Cb, Cr component 별로 진행된다. 한편 디코딩은 8 * 8 크기의 pixel block 단위로 진행이 되는데, 이는 JPEG의 image 해상도 자체가 8의 배수값을 가지고 있어야 함을 의미한다. 따라서 가로 세로 해상도가 8의 배수가 되지 않으면 padding 과정을 추가로 진행해야 한다.

- 하지만 위의 설명대로 실제 8의 배수로 padding을 진행하면 문제가 발생하게 된다. JPEG의 경우에는 인코딩할 때 Luminance를 나타내는 Y component 대신, Chroma를 나타내는 Cb와 Cr component의 가로 세로 해상도 pixel 수를 각각 절반으로 줄여버리는 chroma subsampling 이라는 과정을 진행한다. 즉, Y component의 8 * 8 pixel block의 수와, Cb Cr component의 8 * 8 pixel block의 수의 비가 4:1이 되는 것이다.

- 이 문제를 해결하기 위해, Y component의 8 * 8 block을 실제로는 4개를 묶어 한 번에 디코딩하므로, 실제로는 가로 세로 해상도가 8의 배수가 아니라 16의 배수가 되도록 padding을 해주어야 한다. 이 때 그리고 각 component 들이 얼마나 sub sampling 되었는지 확인하기 위해 SOF marker 내부의 Hn과 Vn을 사용한다.

cf) sampling 측정 방식

1) Hn과 Vn 중 가장 큰 값을 각각 Hmax, Vmax로 지정한다.
2) 가로 해상도 * Hn/Hmax (올림) 하여 해상도를 비교한다.
   예를 들어 Y component의 경우 보통 Hn = 2, Cb & Cr component의 경우 Hn = 1이 들어오므로, Y component의 Hn = 2가 Hmax가 되고, 따라서 Y component의 가로 해상도는 가로 해상도 * 2/2 = 원본과 동일 = sub sampling 되지 않음.

   으로 판별된다. 반면에 Cb & Cr component의 경우 Hn = 1이므로 다음과 같이 가로 해상도가 결정된다.

   가로 해상도 * 1/2 = 원본의 절반에 해당 = chroma sub sampling 파악.

   이 내용을 나타낸 수식 및 예시는 다음과 같다.

$$x_i = \left\lceil X \times \frac{H_i}{H_{max}} \right\rceil \text{ and } y_i \left\lceil Y \times \frac{V_i}{V_{max}} \right\rceil,$$

Component 0	$H_0 = 4, \quad V_0 = 1$
Component 1	$H_1 = 2, \quad V_1 = 2$
Component 2	$H_2 = 1, \quad V_2 = 1$

Then $X = 512$, $Y = 512$, $H_{max} = 4$, $V_{max} = 2$, and x_i and y_i for each component are

Component 0	$x_0 = 512, \quad y_0 = 256$
Component 1	$x_1 = 256, \quad y_1 = 512$
Component 2	$x_2 = 128, \quad y_2 = 256$

(5) 위와 같이 padding 및 subsampling 파악 여부가 끝났으면, SOS marker를 읽었을 때

- 각 component의 DC Huffman table의 ID와 AC Huffman table의 ID
- non-interleaving / interleaving 방식
- Minimum Coded Block의 구성 방식

이 결정된다. 만약 scan 되는 component가 1개라면 non-interleaving, 2개 이상이면 interleaving 방식이고, subsampling이 되었다면 어떻게 sub sampling 되었는지에 따라 다음과 같은 순서로 MCU가 결정된다.

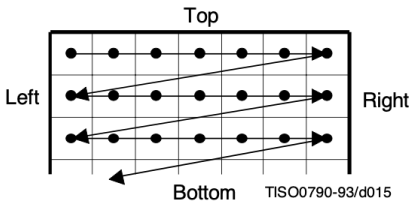
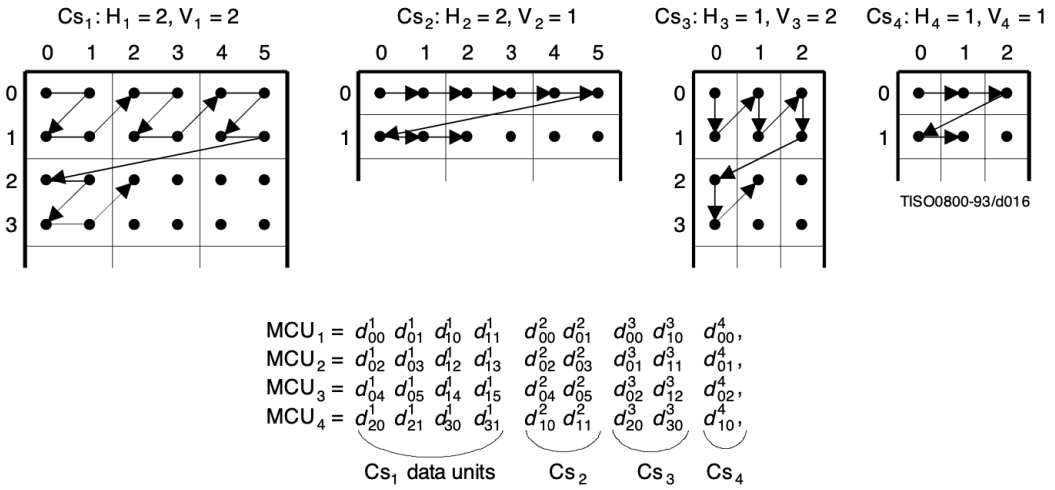


Figure A.2 – Non-interleaved data ordering



대부분은 4:2:0으로 subsampling 되므로, Cs1 1개와 Cs4 2개의 조합, 총 6개의 8*8 block이 한개의 MCU로 묶이게 된다.

(6) 위와 같이 결정되면 이제 MCU segment의 값을 읽어오는데, 이 Segment는 Run-Length & huffman encoding 두 가지 방식을 혼용하여 인코딩 되어 있다.

cf) Run - Length

JPEG의 경우 양자화 table과, DCT(이산 코사인 변환)을 거쳐 대부분의 8 * 8 pixel block의 내부 pixel 값을 0으로 만들어버린다. 하지만 (0,0) index의 값은 더 큰 값을 가지는데, 이 값을 보통 DC coefficient라 하고, 나머지 63개의 값을 AC coefficient라 한다. DC는 DC 끼리 이전 DC coefficient와 현재 DC coefficient의 차이만 인코딩 하는 차분 부호화를 하고, AC는 0이 아닌 coefficient만 인코딩 하되, 자신의 앞에 0이 몇 개 있는지를 같이 나타내어 인코딩 한다. 이렇게 자신의 앞에 0이 몇개 있는지를 같이 인코딩하는 방식을 Run - Length encoding 이라 한다. decoding 시에는 이러한 특성을 반영하여 decoding 한다.

cf2) ZigZag decoding

8 * 8 block의 경우 왼쪽 상단에서부터 점차 0이 많아지는 형태로 pixel 값이 나오므로, 단순히 행순으로 인코딩 하는 것은 효율적이지 못하다. 따라서 JPEG은 zig-zag 방식으로 인코딩을 하고, 디코딩시에도 이 순서를 유지하며 decoding 해야 한다.

cf3) Huffman table

각 Coefficient의 경우 제일 처음 Coefficiet가 몇 bit의 Symbol로 encoding 되어 있는지를 의미하는 huffman symbol이 나타난다. 그 후 뒤이어 해당 값을 나타내는 bit열이 뒤이어 나온다. 즉,

DC coefficient : symbol length + symbol value

AC coefficient : Run length / Symbol length + symbol value

의 형태로 인코딩 되어 있으며, 따라서 디코딩 시에도 이를 유념하여 decoding 해야 한다. 보통 Huffman table의 경우 ITU-81에 제시된 표준 Huffman table을 참조한다.

위의 cf 내용 3가지 개념에 대해 알았다면, 실제로 다음과 같이 decoding process가 진행된다.

먼저 앞의 Bit 열을 쭉 읽는다. 읽으면서 DC huffman table에 해당되는 값이 있는지 확인한다.

Table K.3 – Table for luminance DC coefficient differences

Category	Code length	Code word
0	2	00
1	3	010
2	3	011
3	3	100
4	3	101
5	3	110
6	4	1110
7	5	11110
8	6	111110
9	7	1111110
10	8	11111110
11	9	111111110

표준 DC Huffman table의 경우에는 위와 같으므로, Category 10에 해당하는 Code word 임을 확인할 수 있다. 그럼 다음에는 10개의 bit 열을 읽어들인다. 그리고 Symbol value를 계산한다.

SSSS	DIFF values
0	0
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023
11	–2 047..–1 024,1 024..2 047

JPEG에서는 Symbol value를 계산하는 방법이 별도로 있기 때문에, 0000 0001 00 이라는 bit 열은 단순히 3이 아니라 -1019에 해당한다. 이후 AC coefficient에 대해 마찬가지로 동일한 방법으로, 다른 Huffman table을 사용해 진행한다.

다음에 나오는 값 중 01이라는 bit 열은 아래의 0/2에 해당한다.

Table K.5 – Table for luminance AC coefficients (sheet 1 of 4)

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	5	11011
1/3	7	1111001

즉 앞에는 0개의 0이 있고, 해당 위치에 오는 값은 2 Bit로 표현되는 값이라는 의미이므로, 다시 뒤의 2 Bit를 더 읽어들인다. 그럼 11이라는 Bit열을 읽어오게 된다. 이것을 AC 전용의 Symbol value table을 참조하여 3으로 바꿀 수 있다.

SSSS	AC coefficients
1	–1,1
2	–3,–2,2,3
3	–7..–4,4..7
4	–15..–8,8..15
5	–31..–16,16..31
6	–63..–32,32..63
7	–127..–64,64..127
8	–255..–128,128..255
9	–511..–256,256..511
10	–1 023..–512,512..1 023

참고로, AC와 DC 모두 일반적인 규칙은 모든 bit가 0일 때 가장 작은 값부터 시작하고, 모든 bit가 1일 때 가장 큰 값을 갖게 된다. 즉 11이라면 2 Bit 이므로, -3 (00), -2 (01), 2 (10), 3 (11) 중 3에 해당함을 알 수 있다.
이와 같이 Symbol value를 계산할 때에는 아래와 같은 방법을 사용한다.

- 음수 : -1을 해준 후 아래 부터 n bit를 읽어온다.
- 양수 : 아래부터 n bit를 읽어온다.

예를 들어, 4를 표현하기 위해서는 0000 0100 이라고 할 수 있다. 4는 SSSS가 3에 해당하므로, 하위 3 bit만 읽어오면 된다. 즉 100이 4를 나타내는 값이 되는 것이다. 반대로 -5라면 1111 1011 이 된다. 여기에 -1을 취해주면 -6이므로 1111 1010이 되고, 이 상태에서 하위 3 bit을 읽어오면 010이 된다. 000이 -7이고, 001이 -6, 010이 -5를 나타내므로 일치함을 알 수 있다.

위와 같이 AC component를 (0, 0)(EOB, End of Block)을 나타내는 code를 읽을 때까지 계속 읽거나, 혹은 63번째 AC component의 값이 0이 아닌 값으로 끝날 경우 해당 값을 읽어옴으로써 한 개의 8 * 8 block을 다 디코딩 하게 된다.

(7) 한 개의 8 * 8 block을 디코딩 했다면 양자화 table을 element 별로 곱해준 후, IDCT(Inverse Discrete Cosine Transforming) 후 +128하여 level shift 해준다. 이 때 중요한 것은, 양자화 table을 곱해주었을 때에는 정수, IDCT 한 것은 실수, 그리고 다시 +128 해주기 전에 반드시 -128 ~ 127 사이의 값이 되도록 boundary를 맞추어 주어야 한다. 그렇지 않으면 pixel 값이 overflow 되어 군데군데 점이 생기게 된다.

(8) 위와 같이 1개의 8 * 8 block을 interleaving 방식의 경우 Y component 4개, Cb & Cr component 1개, 총 6개 decoding 하게 되면 하나의 MCU를 decoding 한 셈이 된다. 이 MCU를 width를 16으로 나눈 값만큼 디코딩 하게 되면 하나의 MCU segment를 decoding 한게 된다. 이후 RST marker를 확인 후 다시 새로운 MCU segment를 decoding 한다. 이 과정을 모든 MCU segment를 다 decoding 할 때까지 진행한다.

(9) decoding 된 component는 결국 Y, Cb, Cr component이고, Cb, Cr의 경우에는 가로 세로 해상도를 절반으로 줄인 chroma subsampling을 했으므로 다시 원래의 해상도로 복원시켜주어야 한다. 그냥 한 pixel을 가로 세로 2배로 늘려버릴 수도 있지만, 정확히 하기 위해서 n 번째 pixel row와, n+1번째 pixel row의 평균 값으로 가운데 행을 새로 추가하고, 다시 m번째 pixel column과 m+1번째 pixel column의 평균 값으로 가운데 열을 추가한다. 위와 같은 방식으로 가로 세로 해상도를 2배로 만든 후에 아래 식에 집어 넣어 RGB numpy array를 만든다. 그리고 padding 되어 늘어난 가로 세로 해상도를 다시 잘라낸다.

```
#1. RGB -> YCbCr
# [M, N, 3] : [M, N, 0] = R / [M, N, 1] = G / [M, N, 2] = B
#
#  RGB to YCbCr
#  Y  =   0 + (0.299 * R) + (0.587 * G) + (0.114 * B)
#  Cb = 128 - (0.168736 * R) - (0.331264 * G) + (0.5 * B)
#  Cr = 128 + (0.5 * R) - (0.418688 * G) - (0.081312 * B)
#
#  bias(1, 1, 3) + RGB[M, N, 3].dot(3, 3)
#
#  YCbCr to RGB
#  R  =  Y                          + 1.402 * (Cr - 128)
#  G  =  Y - 0.344136 * (Cb - 128) - 0.714136 * (Cr - 128)
#  B  =  Y + 1.772 * (Cb - 128)
#
#  (YCbCr[M, N, 3] - bias(1, 1, 3)).dot(3, 3)
#
```

[JPG encoder를 구현하며 작성한 color space 변환 주석]

4. JPG encoder
   (1) 먼저 RGB numpy Array를 위의 식을 이용하여, Y Cb Cr로 바꾼다.
   (2) 가로 세로 해상도를 체크하여, 만약 16의 배수가 아니면 16의 배수 해상도가 되도록 padding 처리를 해준다.
   (3) 16의 배수 값을 갖는 해상도로 만들어준 뒤에, Cb Cr component의 경우 4:2:0 chroma sub sampling을 진행한다. 즉 가로 세로 양 옆의 pixel을 모두 합쳐 4로 나눈 평균으로 하나의 pixel 값을 결정한다.
   (4) binary file 하나를 생성 후에 필요한 DQT, DHT marker 값을 집어 넣는다. JFIF와 SOF0 marker 역시 집어 넣고, DRI marker 역시 삽입한다. 이 때 interval은 Y component의 가로 해상도를 16으로 나눈 값을 넣는다. Quantum table의 경우 ITU-81 표준에서 통계적으로 약

50% 정도의 압축 효율을 보이는 table을 제시하고 있으며 huffman table 역시 제시하고 있어 이를 사용하였다.

(5) Y component에서 8 * 8 block을 떼어내어 -128을 element 별로 계산해 level shift를 해주고, DCT를 진행한 후 Qauntum table로 나눈다. 디코딩 및 인코딩 시에 DCT를 및 Quantum table로 나누는 연산을 할 때에는 기본적으로 8 bit data를 16 bit data까지 확장하여 값의 범위를 늘려중어야 한다.

(6) 이후 (0, 0) index인 DC coefficient를 초기 값을 0으로 하고 차분 부호화 한 후, 나머지 63개의 AC coefficient를 Zig-Zag scanning & Run length - Huffman encoding 한다. 간략히 표현하면 다음과 같다.

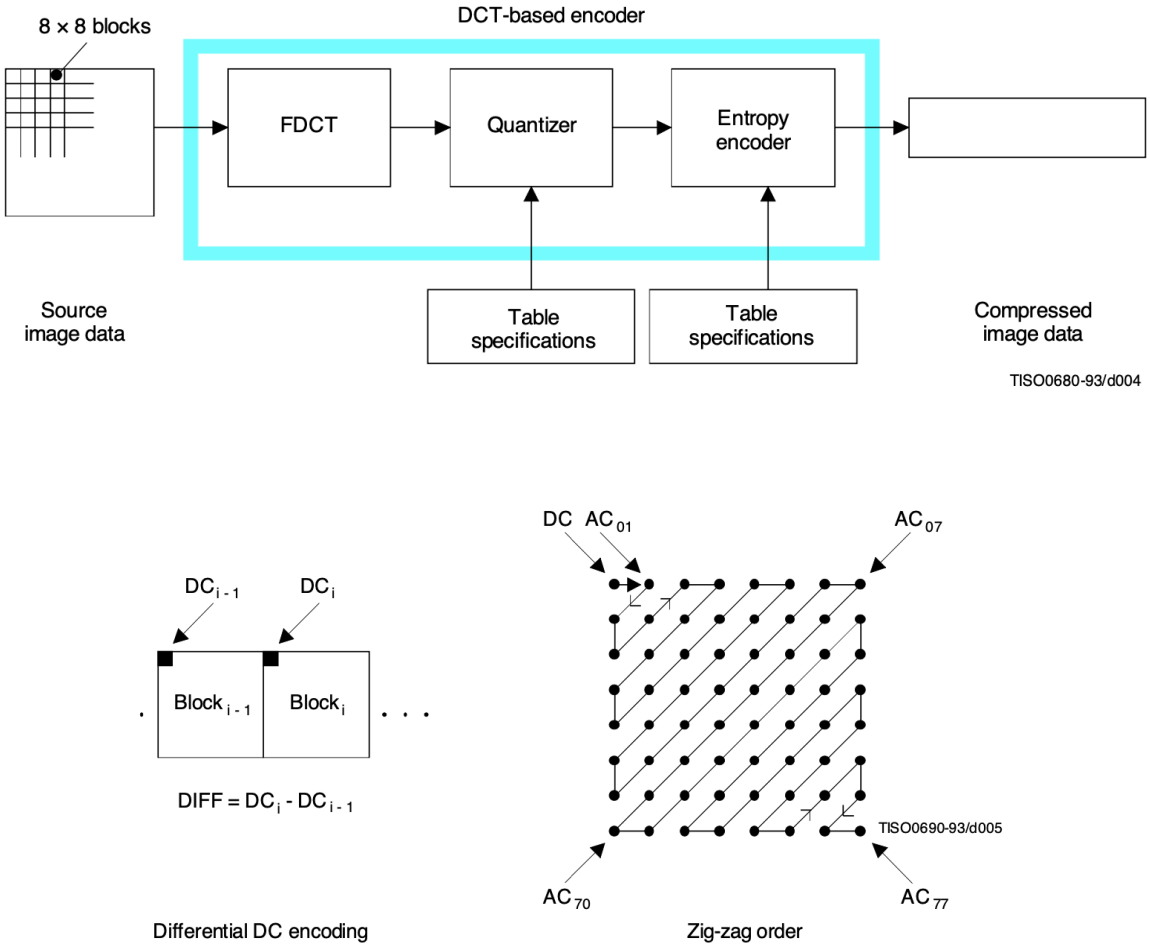


Figure 5 – Preparation of quantized coefficients for entropy encoding

위의 그림은 8 * 8 pixel block의 DC와 AC coefficient 및 Zig Zag scanning 절차를 나타내는 것이다.

(7) 위와 같은 과정을 Y component에 대해서 4번, Cb & Cr component에 대해 1번 수행하면 1개의 MCU에 대한 Encoding이 완료되고, 이런 MCU를 Width / 16만큼 encoding 하고 나면 1번의 MCU segment를 생성할 수 있다. 참고로, MCU segment 안에는 FF가 등장할 경우 이

값이 maker가 아님을 표현하기 위해 뒤에 0x00을 붙이므로, decoding 시에는 이 0x00을 없애고, encoding 시에는 필요하면 0x00을 넣어주는 post processing을 꼭 해야한다.

(8) 각 MCU segment는 반드시 Byte 단위여야 하므로, 만약 MCU segment가 Byte로 끝나지 않으면 bit '1'을 붙여 Byte 단위로 만들어준다. 이후 RST marker 혹은 file의 끝이라면 EOI marker를 붙이면 끝난다.

4.2 DRM

1. SHA-256

단방향 함수로, 한 번 Hash 하고 나면 원본 문자열로 되돌릴 수 없는 함수이다. 보통 고유한 Key 값을 생성하거나, 고유한 Index를 사용하기 위한 Hash table 에서 사용되는 함수다. SHA-256을 선택한 이유는 다음과 같았다.

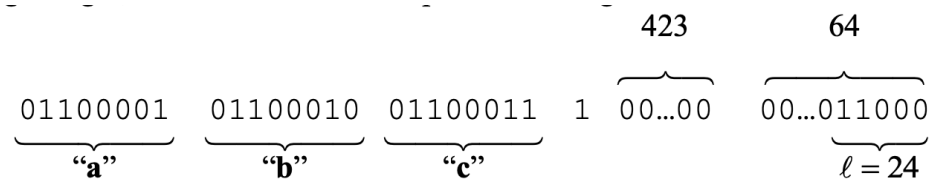
- SHA-1 계열의 hash 함수는 brute force 공격에 이미 취약함이 밝혀졌다.
- SHA-2 계열의 hash 함수에는 4개가 존재하며 그 중 SHA-256이 가장 Key가 길다. Key가 길수록 길이에 비례하여 brute force 해야 하는 수가 2의 지수승으로 커지므로, SHA-2 계열에서 가장 Key가 긴 SHA-256을 선택하였다.
- Key의 길이가 256 Bit로 나오기 때문에, 256 Bit의 Key를 요구하는 대칭키 암호인 AES-256과의 호환성이 좋다.

위과 같은 이유로 SHA-256을 Key 생성 hash 함수로 선택하였다. hash 함수가 256 Bit key를 생성하는

메커니즘은 다음과 같다.

(1) input string의 512 Bit Block화

- input data가 있을 경우 512 Bit 단위의 Block으로 나눈다. 또한 전체적인 Bit 수가, modulo 512 연산을 취했을 때 448 이 나오도록 만든다. 즉 아래의 수식을 만족하도록 한다.



(message의 Bit 길이) + 1(message의 끝을 알림) + 00000…(padding) ≡ 448 mod 512

- 이는 message의 가장 마지막 block에 64 Bit 길이의 전체 message Length를 나타내는 field를 추가했을 때, 512 Bit 길이의 Block을 만들기 위함이다.

(2) 초기 hash 값 설정하기.

$$
\begin{aligned}
H_0^{(0)} &= \texttt{6a09e667} \\
H_1^{(0)} &= \texttt{bb67ae85} \\
H_2^{(0)} &= \texttt{3c6ef372} \\
H_3^{(0)} &= \texttt{a54ff53a} \\
H_4^{(0)} &= \texttt{510e527f} \\
H_5^{(0)} &= \texttt{9b05688c} \\
H_6^{(0)} &= \texttt{1f83d9ab} \\
H_7^{(0)} &= \texttt{5be0cd19}
\end{aligned}
$$

- SHA-256의 경우 32 Bit 단위의 Word 8개가 모여 256 Bit를 구성하고, 또한 이 Word 단위로 hash 연산이 수행된다. 따라서 초기 hash값으로 사용할 32Bit Word 8개의 값을 준비한다.

SHA-224 and SHA-256 use the same sequence of sixty-four constant 32-bit words, $K_0^{\{256\}}, K_1^{\{256\}}, \ldots, K_{63}^{\{256\}}$. These words represent the first thirty-two bits of the fractional parts of the cube roots of the first sixty-four prime numbers. In hex, these constant words are (from left to right)

```
428a2f98 71374491 b5c0fbcf e9b5dba5 3956c25b 59f111f1 923f82a4 ab1c5ed5
d807aa98 12835b01 243185be 550c7dc3 72be5d74 80deb1fe 9bdc06a7 c19bf174
e49b69c1 efbe4786 0fc19dc6 240ca1cc 2de92c6f 4a7484aa 5cb0a9dc 76f988da
983e5152 a831c66d b00327c8 bf597fc7 c6e00bf3 d5a79147 06ca6351 14292967
27b70a85 2e1b2138 4d2c6dfc 53380d13 650a7354 766a0abb 81c2c92e 92722c85
a2bfe8a1 a81a664b c24b8b70 c76c51a3 d192e819 d6990624 f40e3585 106aa070
19a4c116 1e376c08 2748774c 34b0bcb5 391c0cb3 4ed8aa4a 5b9cca4f 682e6ff3
748f82ee 78a5636f 84c87814 8cc70208 90befffa a4506ceb bef9a3f7 c67178f2
```

- 또한 64개의 상수 32Bit Word를 사용하므로 이를 준비한다.

(3) hash 연산 설정하기.

- hash의 경우 주의해야하는 것은, + 연산이 단순한 Bit 끼리의 합이 아니라, modulo 연산으로 수행된다는 점이다. 따라서 이 점을 미리 정의해야 한다.

```python
def add(self, x, y):
    S = (int(str(x), 16) + int(str(y), 16)) % (2**32)
    #print("S : ", S)
    temp = BitArray(uint=S, length=32)
    #print(temp)
    result = temp#[0:32]
    return result
```

[추가로 code 상에서 직접 구현했을 때 정의한 Add 연산. 연산자 overloading을 썼으면 더 좋았을 것이다.]

- 이외에도 Shift와 Rotation, exclusive-or 연산이 있으나 기본적으로 제공되는 bit operation들이므로 생략하고, 이 연산들을 이용해 정의한 hash 함수의 부가적인 연산이 있다.

$$
\begin{aligned}
Ch(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\
Maj(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)
\end{aligned}
$$

$$
\begin{aligned}
\sum_0^{\{256\}}(x) &= ROTR^{2}(x) \oplus ROTR^{13}(x) \oplus ROTR^{22}(x) \\
\sum_1^{\{256\}}(x) &= ROTR^{6}(x) \oplus ROTR^{11}(x) \oplus ROTR^{25}(x) \\
\sigma_0^{\{256\}}(x) &= ROTR^{7}(x) \oplus ROTR^{18}(x) \oplus SHR^{3}(x) \\
\sigma_1^{\{256\}}(x) &= ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)
\end{aligned}
$$

- 위의 연산은 word를 뒤섞거나 늘릴 때, 그리고 사용자가 입력한 input message를 hash에 넣을 때 사용된다.

(4) hash 함수 반복하기.

For i=1 to N:
{

1. Prepare the message schedule, $\{W_t\}$:

$$
W_t = \begin{cases} M_t^{(i)} & 0 \le t \le 15 \\ \sigma_1^{\{256\}}(W_{t-2}) + W_{t-7} + \sigma_0^{\{256\}}(W_{t-15}) + W_{t-16} & 16 \le t \le 63 \end{cases}
$$

- 먼저 하나의 512 Bit message Block을 32Bit word로 나누어 0 ~ 15까지의 16개 word를 채운다. 이후 위에서 정의한 연산을 사용하여 나머지 48개의 Word를 생성한다.
- a ~ h 까지 32 Bit word에, 기존의 256 Bit hash를 쪼개 할당 후 64개의 word에 대해 아래의 연산을 수행하여 a ~ h의 32 Bit word를 뒤섞는다. T1, T2는 temp 값을 말한다.

$$
\begin{aligned}
a &= H_0^{(i-1)} \\
b &= H_1^{(i-1)} \\
c &= H_2^{(i-1)} \\
d &= H_3^{(i-1)} \\
e &= H_4^{(i-1)} \\
f &= H_5^{(i-1)} \\
g &= H_6^{(i-1)} \\
h &= H_7^{(i-1)}
\end{aligned}
$$

3. For t=0 to 63:
{

$$
\begin{aligned}
T_1 &= h + \sum_1^{\{256\}}(e) + Ch(e,f,g) + K_t^{\{256\}} + W_t \\
T_2 &= \sum_0^{\{256\}}(a) + Maj(a,b,c) \\
h &= g \\
g &= f \\
f &= e \\
e &= d + T_1 \\
d &= c \\
c &= b \\
b &= a \\
a &= T_1 + T_2
\end{aligned}
$$

}

- 기존의 hash에 뒤섞은 a ~ h 변수를 더한 후 합쳐서 256 Bit hash를 만든다.
  이 과정을 모든 message block에 대해 수행한다.

4. Compute the i^{th} intermediate hash value $H^{(i)}$:

$$H_0^{(i)} = a + H_0^{(i-1)}$$
$$H_1^{(i)} = b + H_1^{(i-1)}$$
$$H_2^{(i)} = c + H_2^{(i-1)}$$
$$H_3^{(i)} = d + H_3^{(i-1)}$$
$$H_4^{(i)} = e + H_4^{(i-1)}$$
$$H_5^{(i)} = f + H_5^{(i-1)}$$
$$H_6^{(i)} = g + H_6^{(i-1)}$$
$$H_7^{(i)} = h + H_7^{(i-1)}$$

}

After repeating steps one through four a total of N times (i.e., after processing $M^{(N)}$), the resulting 256-bit message digest of the message, M, is

$$H_0^{(N)} \| H_1^{(N)} \| H_2^{(N)} \| H_3^{(N)} \| H_4^{(N)} \| H_5^{(N)} \| H_6^{(N)} \| H_7^{(N)}$$

2. AES-256
   - GF(가우스 필드)를 사용하는 대칭키 암호화 알고리즘이다.
   - 암호화 알고리즘이지만 미국 FIPS의 문서에 표준화 되어 공개되어 있는 암호화 알고리즘이다.
   - DES의 문제점을 해결하기 위해 제시된 새로운 암호화 알고리즘이다.

RSA와 같은 비대칭키 암호화 알고리즘도 있으나 public key 관리 문제 및 소인수분해 연산과 같은 부분이 구현하기 복잡하고 속도가 상대적으로 느리며 관리하기 어렵다고 판단되어, 주로 Bit operation을 사용하여 속도가 빠르고 소인수분해 연산을 구현할 필요가 없는 AES-256 암호화 알고리즘을 채택하였다.

(1) 가우스 필드를 활용하여 bit열을 다항식으로 이해하기

$$b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0 = \sum_{i=0}^{7} b_i x^i \ . \qquad (3.1)$$

For example, {01100011} identifies the specific finite field element $x^6 + x^5 + x + 1$.

위와 같이 8 Bit를 7차 다항식으로 이해할 수 있다. 이 경우 8차 이상의 다항식은 다음의 식으로 나누어 8 Bit에 다시 넣는다.

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 \ \text{modulo} \ (x^8 + x^4 + x^3 + x + 1)$$
$$= \quad x^7 + x^6 + 1 .$$

(2) State Array 준비하기

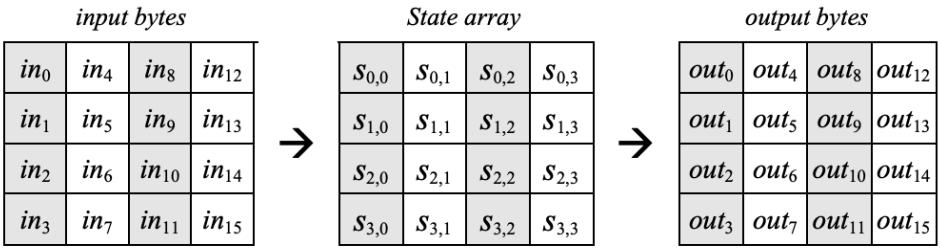


Figure 3. State array input and output.

AES-256의 경우 16 Byte 단위로 암호화를 하므로, 위와 같이 4 * 4 Byte array에 각각 1 Byte 씩 넣어두어 State Array 라는 것을 만든다. 이후 모든 연산은 이 state array 단위로 이루어지게 된다.

(3) Sub Byte & inverse Sub Byte 연산

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure 7. S-box: substitution values for the byte xy (in hexadecimal format).

		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
	1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
	2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
	3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
	4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
	5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
	6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
	7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
	8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
	9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
	a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
	b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
	c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
	d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
	e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
	f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figure 14. Inverse S-box: substitution values for the byte xy (in hexadecimal format).

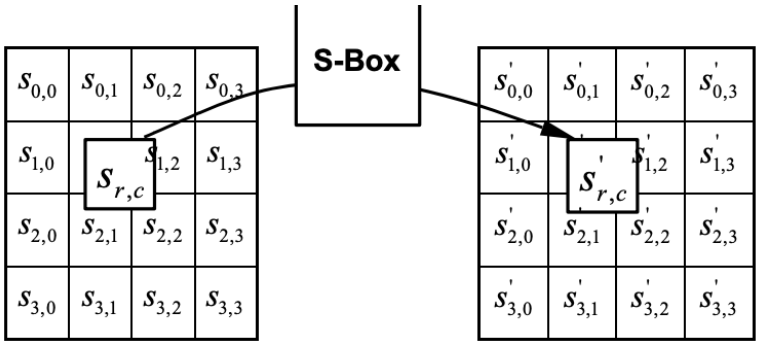


Figure 6. `SubBytes()` applies the S-box to each byte of the State.

위와 같이 S-Box 안의 값을 참고하여 State array의 값을 다 바꾸어준다. Inverse S-Box의 경우 복호화에 사용되는 S-Box이다.

(4) Shift Rows & Inverse Shift Rows

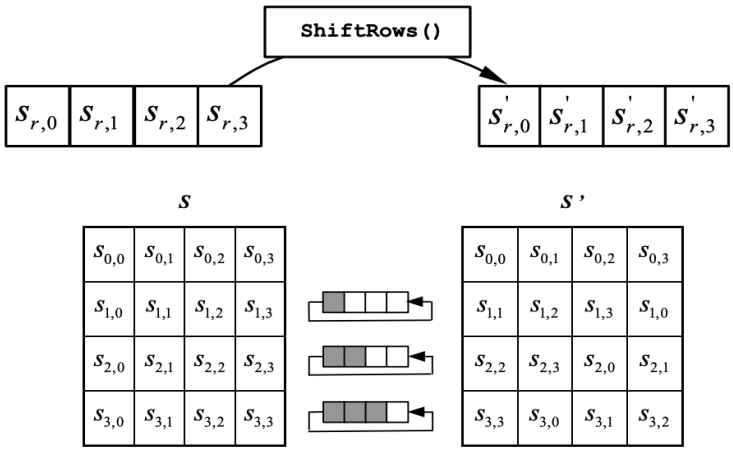


Figure 8. `ShiftRows()` cyclically shifts the last three rows in the State.

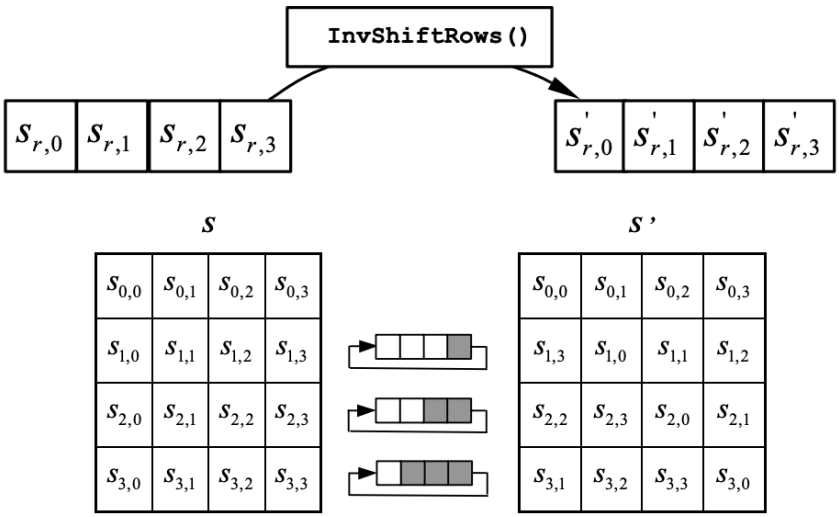


Figure 13. `InvShiftRows()` cyclically shifts the last three rows in the State.

위와 같이 S-Box 를 사용하여 state array의 byte를 뒤섞어준 후, shiftrows를 사용하여 row 단위로 state array를 한번씩 또 shift 해준다.

(5) mixColumn & Inverse mix Column

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb. \qquad (5.6)$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{02\} \bullet s_{0,c}) \oplus (\{03\} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (\{02\} \bullet s_{1,c}) \oplus (\{03\} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (\{02\} \bullet s_{2,c}) \oplus (\{03\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{03\} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (\{02\} \bullet s_{3,c}).$$

Figure 9 illustrates the **MixColumns()** transformation.

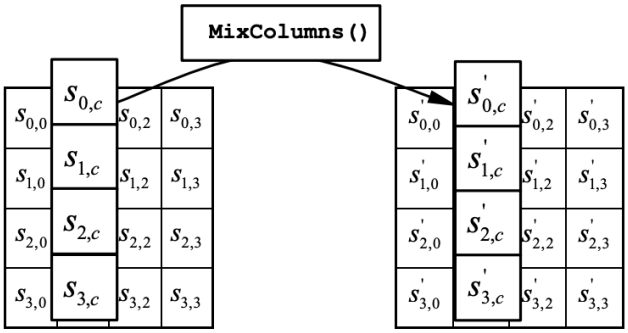


Figure 9. MixColumns() operates on the State column-by-column.

위의 다항식 matrix 곱을 이용하여, 각 Column의 값을 바꾸어준다. 마찬가지로 inverse mix column 연산이 있다. inverse mix column 연산에서 사용하는 저 값은 GF 상에서의 위의 matrix의 역원에 해당한다.

$s'(x) = a^{-1}(x) \otimes s(x)$:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \quad \text{for } 0 \le c < Nb. \qquad (5.10)$$

As a result of this multiplication, the four bytes in a column are replaced by the following:

$$s'_{0,c} = (\{0e\} \bullet s_{0,c}) \oplus (\{0b\} \bullet s_{1,c}) \oplus (\{0d\} \bullet s_{2,c}) \oplus (\{09\} \bullet s_{3,c})$$

$$s'_{1,c} = (\{09\} \bullet s_{0,c}) \oplus (\{0e\} \bullet s_{1,c}) \oplus (\{0b\} \bullet s_{2,c}) \oplus (\{0d\} \bullet s_{3,c})$$

$$s'_{2,c} = (\{0d\} \bullet s_{0,c}) \oplus (\{09\} \bullet s_{1,c}) \oplus (\{0e\} \bullet s_{2,c}) \oplus (\{0b\} \bullet s_{3,c})$$

$$s'_{3,c} = (\{0b\} \bullet s_{0,c}) \oplus (\{0d\} \bullet s_{1,c}) \oplus (\{09\} \bullet s_{2,c}) \oplus (\{0e\} \bullet s_{3,c})$$

(6) add round key & key expansion function

- add round key의 경우 inverse 과정이 동일함. 단, key expansion function을 통해 만들어낸 key를 역순으로 더해준다는 것이 차이점이다.
- 시작 전에 각 state array column 별로 4번, 또 AES-256의 경우 14 round가 진행되므로 round 별로 4번, 총 4 + 14*4 = 4 + 56 = 60개의 round key가 필요하므로, 256 Bit key를 확장시키는 key expansion function이 꼭 필요하다.

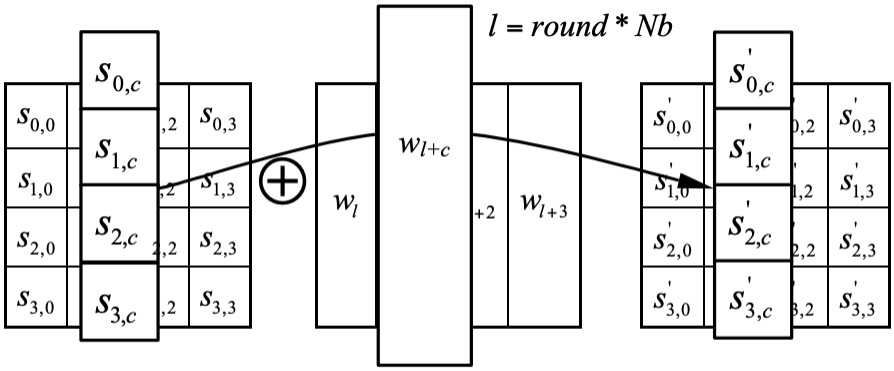


Figure 10. `AddRoundKey()` XORs each column of the State with a word from the key schedule.

```
KeyExpansion(byte key[4*Nk], word w[Nb*(Nr+1)], Nk)
begin
   word  temp

   i = 0

   while (i < Nk)
      w[i] = word(key[4*i], key[4*i+1], key[4*i+2], key[4*i+3])
      i = i+1
   end while

   i = Nk

   while (i < Nb * (Nr+1)]
      temp = w[i-1]
      if (i mod Nk = 0)
         temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
      else if (Nk > 6 and i mod Nk = 4)
         temp = SubWord(temp)
      end if
      w[i] = w[i-Nk] xor temp
      i = i + 1
   end while
end
```

- 위는 key expasion 알고리즘이다. 256 Bit key를 Byte 단위로 나누어 32개의 key를 만든 후, 다시 4개씩 묶어 8개의 word를 만든다. 이후 52개의 key를 확장으로 통해 만들어낸다. 여기서 SubWord는 SubByte와 마찬가지로 S-Box를 사용해 Byte를 치환하는 것이고, RotWord는 단순히 [a0, a1, a2, a3] 인 4 Byte word를 [a1, a2, a3, a0]으로 rotation 하는 것이다.

- Rcon의 경우 [2^(i/Nk) (AES-256의 경우 Nk = 8)인 bit열을 갖는 1 Byte + 0x00인 3 Byte]로 이루어진 word이다. 예를 들어 i = 32라면 [0x10, 0x00, 0x00, 0x00]인 4 Byte word가 된다.

- 위와 같은 Key expansion으로 60개의 key를 만들어낸 후 암호화는 0 ~ 59 순서로, 복호화는 59 ~ 0 순서로 add round key() 과정을 처리하면 된다.

(7) 전체적인 암/복호화 알고리즘

- AES-256의 경우 Nk = 8, Nb = 4, Nr = 14임을 참고하면 된다.

	Key Length *(Nk words)*	**Block Size** *(Nb words)*	**Number of Rounds** *(Nr)*
AES-128	4	4	10
AES-192	6	4	12
AES-256	8	4	14

```
Cipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[0, Nb-1])                  // See Sec. 5.1.4

   for round = 1 step 1 to Nr-1
      SubBytes(state)                              // See Sec. 5.1.1
      ShiftRows(state)                             // See Sec. 5.1.2
      MixColumns(state)                            // See Sec. 5.1.3
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
   end for

   SubBytes(state)
   ShiftRows(state)
   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])

   out = state
end
```

Figure 5. Pseudo Code for the Cipher.[1]

```
InvCipher(byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
   byte  state[4,Nb]

   state = in

   AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1]) // See Sec. 5.1.4

   for round = Nr-1 step -1 downto 1
      InvShiftRows(state)                    // See Sec. 5.3.1
      InvSubBytes(state)                     // See Sec. 5.3.2
      AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
      InvMixColumns(state)                   // See Sec. 5.3.3
   end for

   InvShiftRows(state)
   InvSubBytes(state)
   AddRoundKey(state, w[0, Nb-1])

   out = state
end
```

Figure 12. Pseudo Code for the Inverse Cipher.[3]

4.2.1 암호화

- 사용자로부터 문자열을 입력 받아 SHA-256 함수를 통해 AES-256 암호화 알고리즘에 사용할 256 Bit key를 만들어낸다.
- 현재 불러온 파일을 AES-256 암호화 알고리즘을 이용해 binary file 자체를 암호화 후, 뒤에 .enc라는 확장자를 추가하여 암호화 된 파일임을 명시한다. 해당 file은 실제로 file signature가 바뀌어 버리므로 일반적인 image 파일로 인식되지 않고, 설령 file signature를 다른 사용자가 임의로 변경하여 image 파일로 인식 되게 만든다 하더라도 나머지 부분이 암호화 되었기 때문에 정상적인 디코딩을 할 수 없다.

4.2.2 복호화

- 사용자로부터 문자열을 입력 받아 SHA-256 함수를 통해 AES-256 복호화 알고리즘에 사용할 256 Bit key를 만들어낸다.
- 복호화 할 파일의 경로를 입력 받으면 해당 파일을 불러온 후, AES-256 복호화 알고리즘을 이용해 binary file 자체를 복호화한 후, 뒤에 .dec 라는 확장자를 추가하여 복호화 된 파일임을 명시한다. 뒤의 .enc.dec 파일 확장자를 지울 경우 다시 정상적인 image 파일로 인식된다.
- 만약 key를 잘못 입력하게 될 경우, 복호화가 진행되면 이는 또 다른 암호화가 진행되는 것과 동일하므로, key를 잘못 입력해도 여전히 image로 인식되는 파일이 나올 수 없다.

4.3 이외 기능

4.3.1 회전

(1) 90도 회전

현재 색상 정보 numpy와 방향(오른쪽, 왼쪽)을 인자로 받는다. width와 height를 바꿔 회전 후의 색상 공간을 저장할 buffer를 만든다. 회전에 따라 배열의 index를 이용해 버퍼에 numpy 값을 복사한다.

(2) 자유 각도 회전

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

현재 색상 정보 numpy와 방향(오른쪽, 왼쪽), 각도(1~45도)를 인자로 받는다. 먼저 현재 이미지 사이즈의 버퍼를 만들고 픽셀 값을 0으로 세팅한다. 위의 공식을 이용하여 회전한 인덱스 버퍼에 기존 이미지의 픽셀 값을 복사한다. 이때 회전한 이미지 내에 중간중간 기존 이미지로부터 픽셀값을 받지 못한 부분은 검은색으로 표시된다. 이 hole을 보완하기 위해 보간법을 이용한다. 선형 보간법을 이용하여 hole의 왼쪽 값과 오른쪽 값의 평균으로 hole을 매꾼다. 이후 회전한 이미지를 확대한 후 원래 해상도만큼 자른다.

4.3.2 대칭

현재 색상 정보 numpy와 방향(상하, 좌우)를 인자로 받는다. 인덱스를 이용하여 상하는 열을 뒤집고 좌우는 행을 뒤집어서 대칭된 색상 정보 배열을 만든다.

4.3.3 해상도 조절

(1) 비율 유지

현재 색상 정보 numpy와 가로 해상도, 세로 해상도를 인자로 받는다. 단, 가로 해상도를 입력한 경우 세로 해상도는 0으로 전달하고 세로 해상도 조절 시에는 가로 해상도를 0으로 전달하여 구분한다. numpy를 통해 원래 이미지의 width:height 비율을 저장하고 입력받은 가로 또는 세로 해상도에 이 비율을 곱해 세로 또는 가로 해상도를 구한다. 조절된 해상도에 맞게 버퍼를 생성하고 모든 픽셀 값을 0으로 세팅한다. 버퍼의 인덱스 중 numpy의 인덱스에 변화 비율을 곱해서 나온 인덱스에 해당 numpy의 값을 복사한다.

이때 버퍼 중간중간 기존 numpy로부터 픽셀 값을 받지 못한 부분은 검은색으로 표시된다. 이 hole을 보완하기 위해 보간법을 이용한다. 두 가지의 선형 보간법을 이용하는데, 먼저 hole의 왼쪽 값과 오른쪽 값의 평균으로 hole을 매꾼다. 다음 hole의 위쪽 값과 아래쪽 값의 평균으로 hole에 누적하여 픽셀 값을 계산한다. 이렇게 수직, 수평 보간법을 이용하여 빈 픽셀 값을 보완한다.

(2) 자유 비율

현재 색상 정보 numpy와 가로 해상도, 세로 해상도를 인자로 받는다. 가로 해상도와 세로 해상도를 모두 입력받아 전달한다. 입력받은 해상도에 맞게 버퍼를 생성하고 모든 픽셀 값을 0으로 세팅한다. 버퍼의 인덱스 중 numpy의 인덱스에 변화 비율을 곱해서 나온 인덱스에 해당 numpy의 값을 복사한다.

이때 버퍼 중간중간 기존 numpy로부터 픽셀 값을 받지 못한 부분은 검은색으로 표시된다. 이 hole을 보완하기 위해 보간법을 이용한다. 두 가지의 선형 보간법을 이용하는데, 먼저 hole의 왼쪽 값과

오른쪽 값의 평균으로 hole을 매꾼다. 다음 hole의 위쪽 값과 아래쪽 값의 평균으로 hole에 누적하여 픽셀 값을 계산한다. 이렇게 수직, 수평 보간법을 이용하여 빈 픽셀 값을 보완한다.

4.3.4 잘라내기

현재 색상 정보 numpy와 자를 부분의 좌표 값 두 개를 인자로 받는다. 위의 그림에서 보면 박스 친 부분이 자를 부분일 때 왼쪽 위의 좌표와 오른쪽 아래 좌표를 전달하면 된다. 이때 이미지의 가장 왼쪽 상단 좌표를 (0,0)으로 하고 오른쪽, 아래로 갈수록 좌표의 x, y 값이 커진다. 받은 좌표의 x좌표끼리, y좌표끼리 뺄셈하여 자를 부분의 width와 height를 계산하고 이에 따라 버퍼를 생성한다. 이 버퍼에다 numpy에서 자르는 부분에 해당하는 값을 복사한다.

4.3.5 뒤로가기 / 앞으로 가기

pm 클래스에 history 리스트를 두어 뒤로가기에 필요한 명령어 또는 기능 전, 후의 numpy를 저장한다. 90도 회전과 대칭의 경우 역 명령어를 저장한다. 자유 회전과 해상도 조절, 자르기의 경우는 명령어 수행 전과 후의 numpy를 저장한다.

(1) 뒤로가기

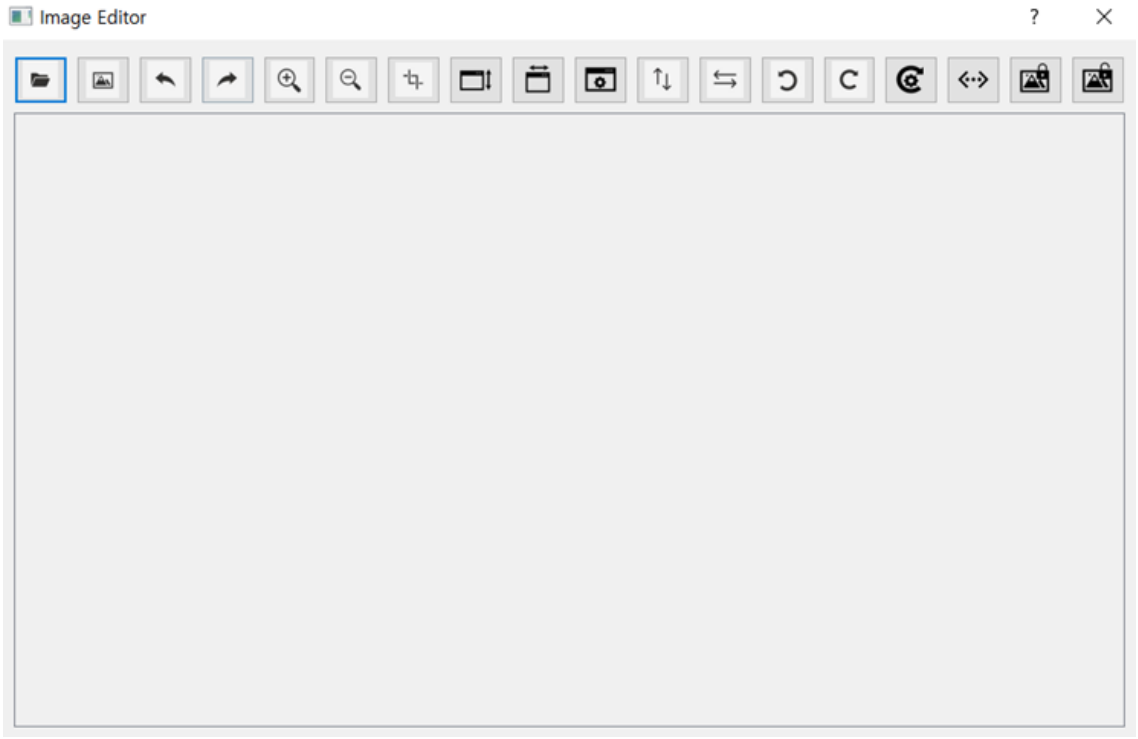
뒤로가기 수행 시 history pointer를 인덱스 값으로 하여 현재 이미지에 history에 저장된 명령어를 수행하거나 현재 numpy값을 명령어 수행 전 numpy으로 교체한다. 뒤로가기 도중 새로운 기능을 수행하면 history pointer의 인덱스 부터 history 끝까지(뒤로가기가 진행된 리스트)는 삭제되고 새로운 기능에 대한 역명령어가 다시 history에 저장된다.

(2) 앞으로가기

뒤로가기와 마찬가지로 history와 history pointer를 이용하여 뒤로가기의 역기능을 한다. 90도 회전과 대칭의 경우 history에 저장된 명령어의 역을 수행하도록 한다. 해상도 조절, 잘라내기, 자유각도 회전은 history에 저장된 기능 후의 numpy값을 받아온다.

4.4 GUI

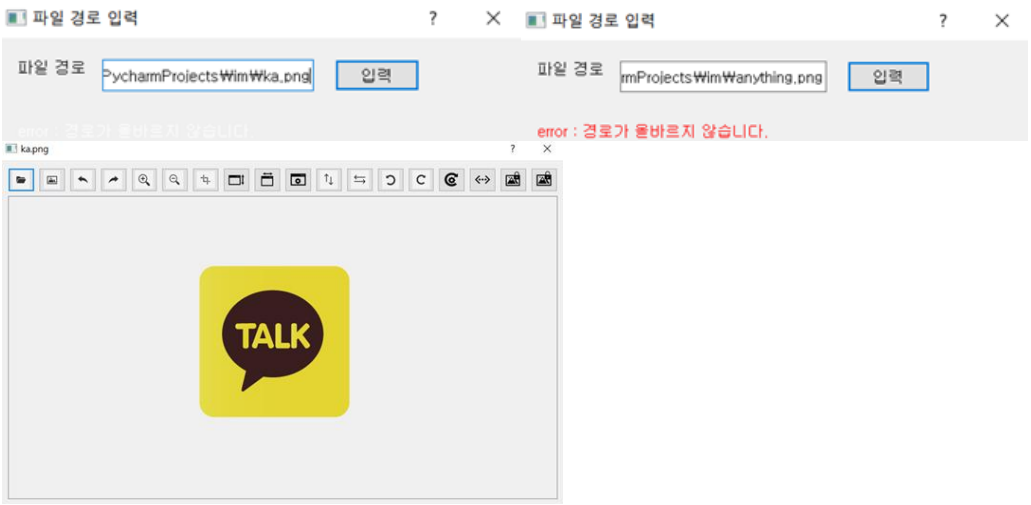
(1) 기본 화면



gui 첫 화면입니다. 18개의 아이콘이 있으며, 각 버튼마다 기능이 연결되어 있습니다.

	파일 열기		잘라내기		왼쪽 90도 회전
	파일 저장		상하 해상도 조절		오른쪽 90도 회전
	실행 취소		좌우 해상도 조절		자유 각도 회전 (0~45 도)
	실행 회복 (취소를 취소 하는 기능)		자유 해상도 조절		이미지 포맷 변환
	이미지 확대		상하 대칭		암호화
	이미지 축소		좌우 대칭		복호화

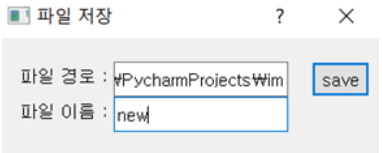
(2) 파일 열기



→ 이미지 파일의 경로를 입력하면 이미지 파일의 포맷에 따라 디코딩하여 출력합니다.

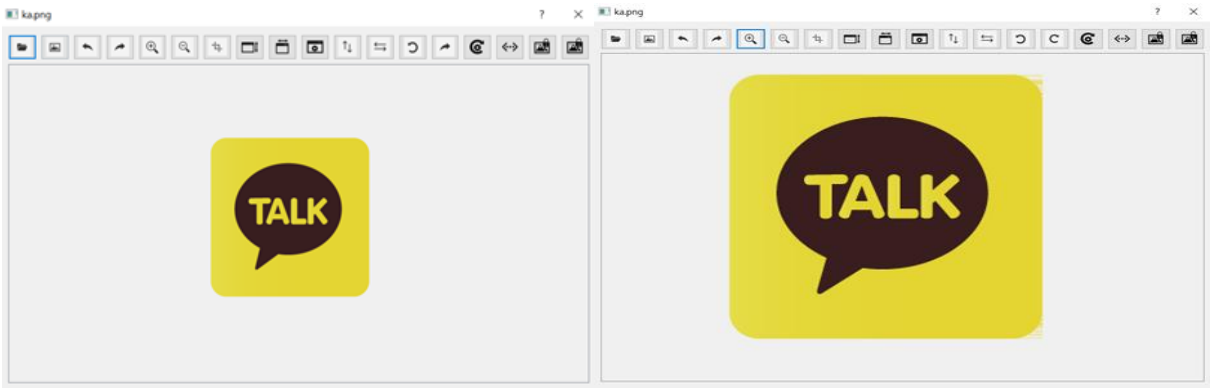
→ 존재하지 않는 이미지 혹은 경로 입력이 잘못 되어 있으면 error 메세지가 출력됩니다.

(3) 파일 저장

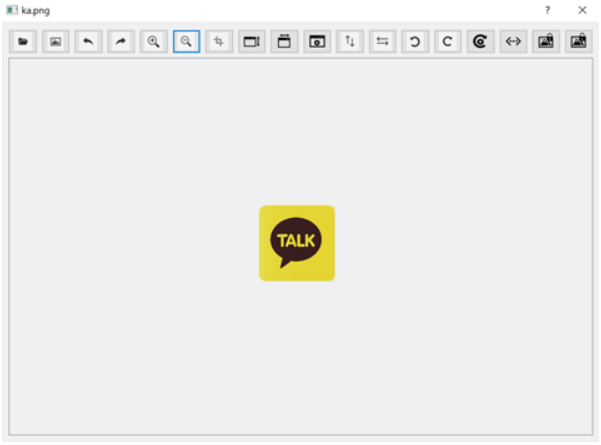


→ 저장하고자 하는 경로를 파일 경로에 입력 후, 파일 이름에 저장하고자 하는 이름을 입력하면 저장됩니다.

(4) 이미지 확대, 축소



→ 원본 이미지 → 확대 이미지

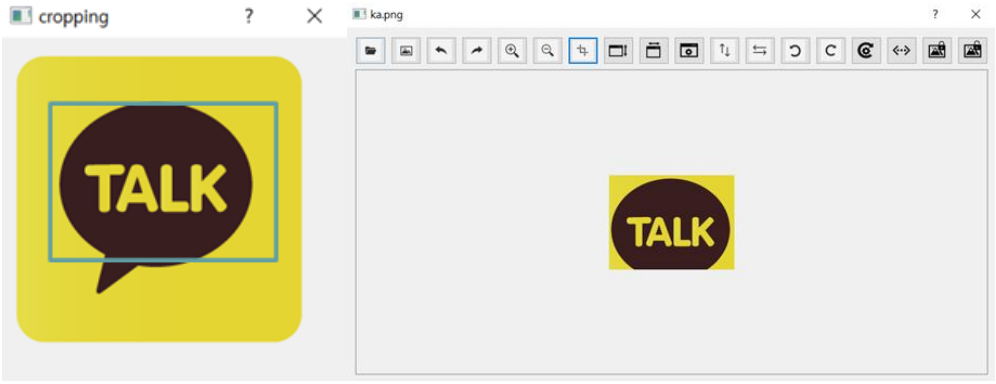


→ 축소 이미지

→ 확대 버튼을 누르면 원래 이미지의 해상도를 상하 좌우를 2배씩 한 이미지를 출력합니다

→ 축소 버튼을 누르면 원래 이미지의 해상도를 상하 좌우를 0.5배씩 한 이미지를 출력합니다.

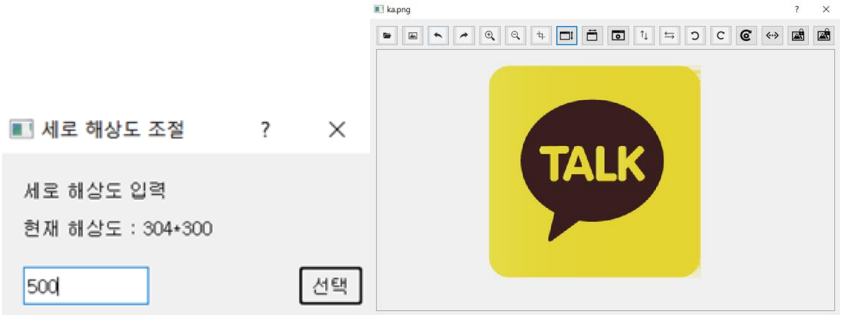
(5) 잘라내기



→ 잘라내기 버튼을 누르면 이미지를 출력하는 작은 화면이 출력됩니다.

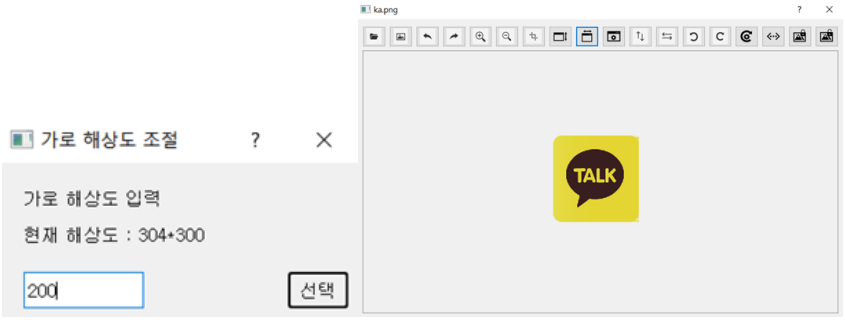
→ 원하는 영역만큼 마우스를 좌클릭 후 드래그를 하면 우측 이미지처럼 잘린 영역의 이미지를 출력합니다.

(6) 해상도 조절

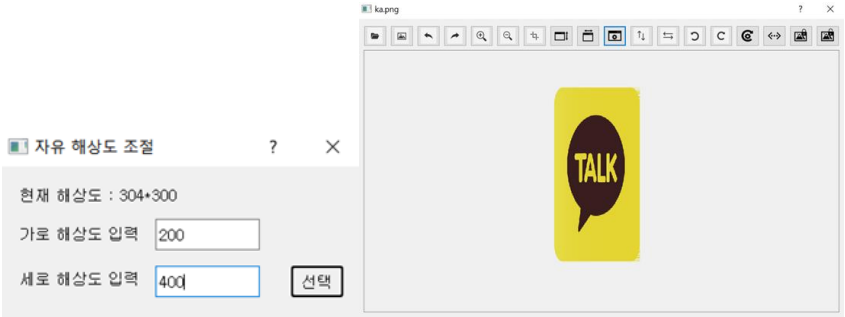


→ 세로 해상도 조절 버튼을 누른 후, 원하는 해상도를 입력합니다.

→ 입력한 세로 해상도에 따라 원본 이미지의 비율에 맞춰 가로 해상도를 자동으로 조절합니다.



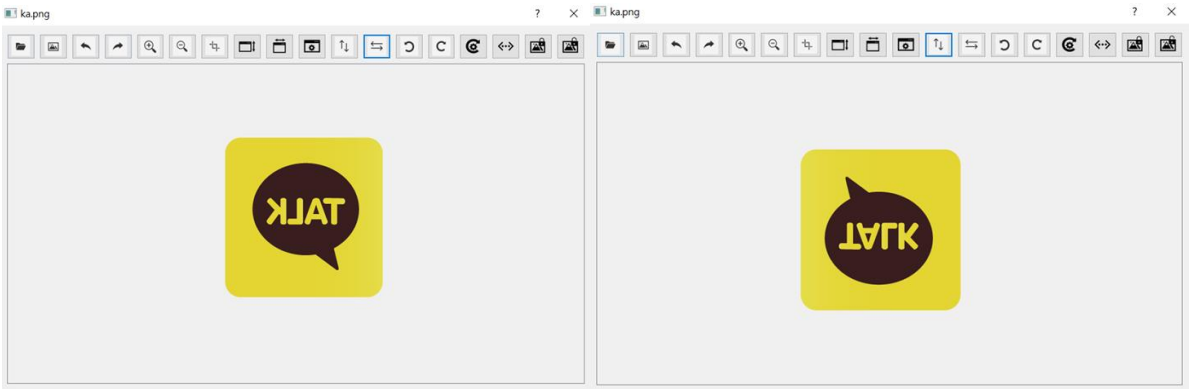
→ 가로 해상도 또한 세로 해상도 조절과 동일합니다.



→ 자유 해상도 조절은 가로 해상도와 세로 해상도를 둘 다 입력합니다.

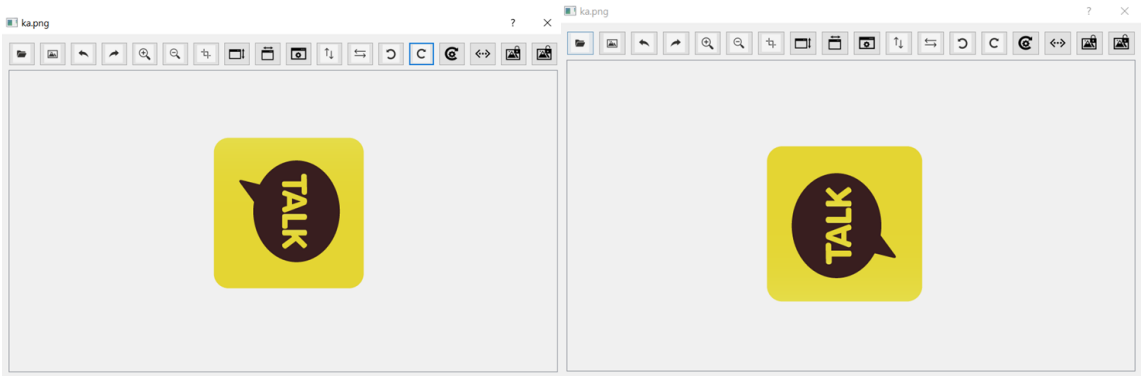
→ 위 두 기능과 달리 이미지의 비율은 고려 하지 않습니다.

(7) 대칭



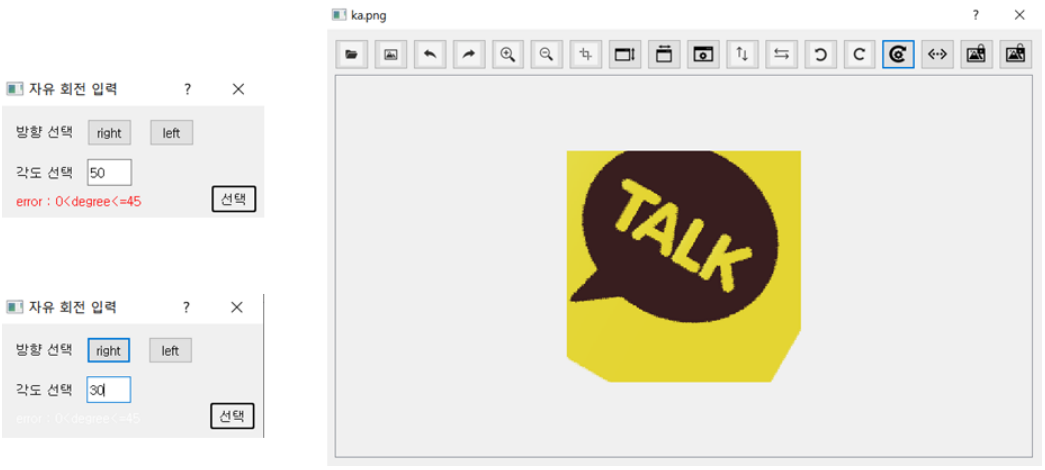
→ 좌측 이미지는 좌우 대칭버튼을 누른 결과 이며, 우측 이미지는 상하 대칭버튼을 누른 결과입니다.

(8) 회전



→ 왼쪽 이미지는 오른쪽 90도 회전 버튼을 누른 결과입니다.

→ 오른쪽 이미지는 왼쪽 90도 회전 버튼을 누른 결과입니다.

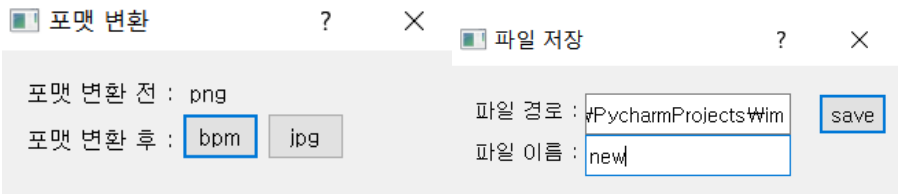


→ 자유 회전 버튼을 누르면 왼쪽 이미지 처럼 새로운 창이 생깁니다.

→ 방향 선택을 하고 각도를 1~45 도 내에서 입력하면 오른쪽 이미지와 같이 결과가 출력됩니다.

→ 1~45도 가 아닌 값을 입력하면 error 메세지가 나옵니다.

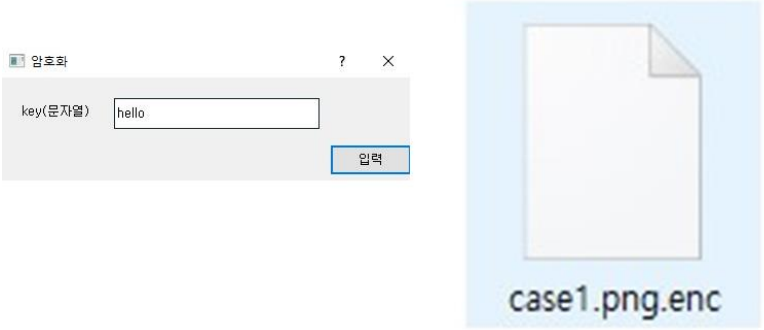
(9) 포맷 변환



→ 포맷 변환을 누르면 왼쪽 이미지 처럼 원하는 포맷으로 변환 할 수 있습니다.

→ 변환한 이미지는 파일 저장 버튼을 눌러서 저장을 해야합니다.

(10) 암호화



→ 암호화 버튼을 누르면 암호화를 하기 위한 key 입력 창이 나옵니다.

→ 입력 하면 우측 이미지와 같이 .enc 포맷의 암호 파일이 생성됩니다.

(11) 복호화

→ 복호화 버튼을 누르면 복호화를 하기 위한 **key** 입력과 파일 경로를 입력 받는 창이 나옵니다.

→ 암호화 시 입력 했던 **key** 값을 입력해야 복호화가 정상적으로 됩니다.

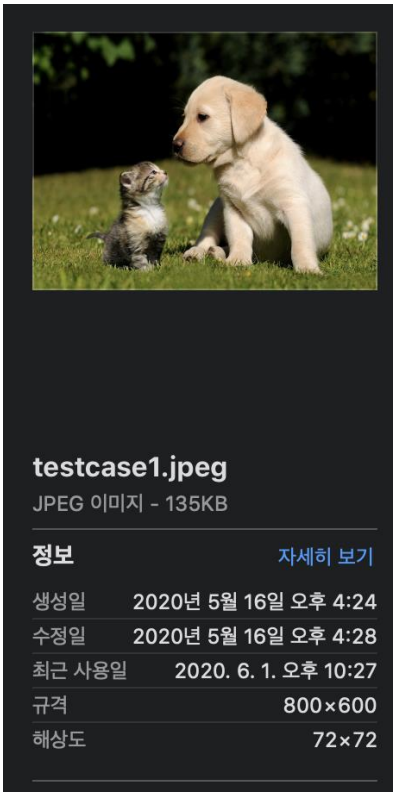
→ 복호화 완료시 우측 이미지와 같이 **.dec** 포맷의 복호화 파일이 생성됩니다.

→ 이미지 파일로 사용을 위해서는 **.enc.dec** 이름을 삭제하면 가능 합니다.

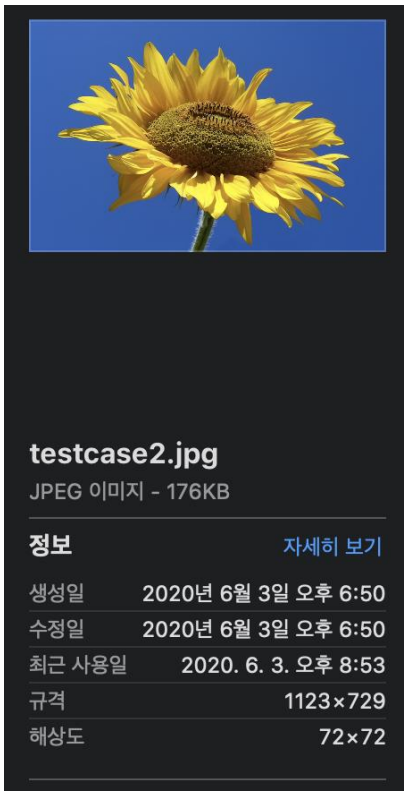
5. 결과

5.1 System Test Cases - 여기서는 해상도가 **pixel per inch**를 의미하고 있음에 유의.

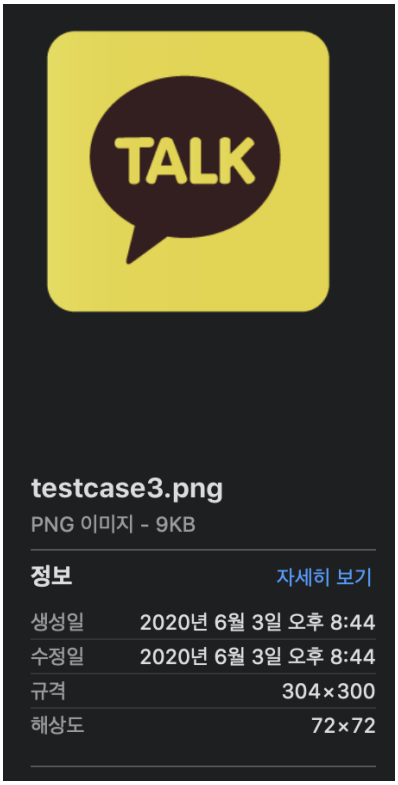
(1) 800 * 600 jpg color image



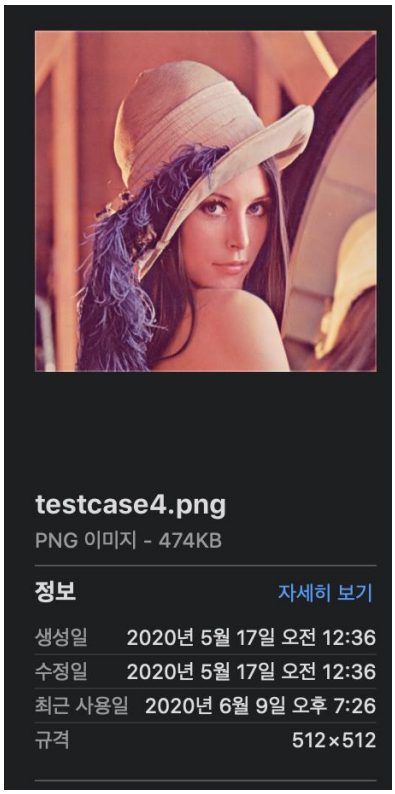
(2) 1123 * 729 jpg color image



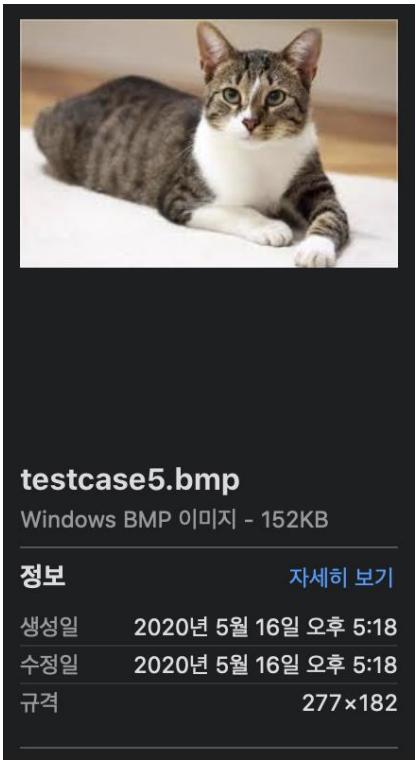
(3) 304 * 300 png color image



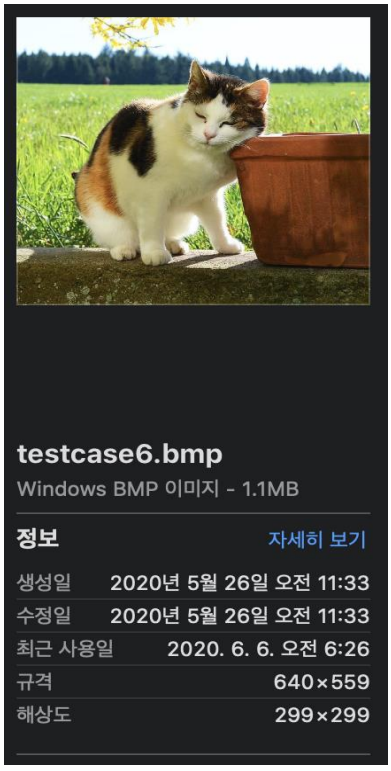
(4) 512 * 512 png color image



(5) 277 * 182 bmp color image



(6) 640 * 559 bmp color image



5.2 System Test Pass & Fail

Spec	input	Expected Output	Pass / Fail	comment
1.1	(3), 가로 520, 세로 230 해상도	화면에 520 * 230 해상도를 가진 png color image 출력	P	구현 완료
1.2.1.	(2), 가로 1600 해상도	화면에 1600 * 1039 해상도를 가진 jpg color image 출력	P	구현 완료
1.2.2.	(4), 세로 539 해상도	화면에 539 * 539 해상도를 가진 png color image 출력	P	구현 완료
2.1.	(6), 왼쪽 회전 1번	화면에 왼쪽으로 90° 회전된 559 * 640 bmp color image 출력	P	구현 완료
2.2.	(5), 오른쪽 회전 3번	화면에 오른쪽으로 270° 회전된 182 * 227 bmp color image 출력	P	구현 완료
2.3.	(4), 왼쪽 회전 27°	화면에 왼쪽으로 27° 회전된 381 * 381 png color image 출력	P*	구현 완료* (513 * 513 png color image 출력)
3.1.	(1), 상하 대칭	화면에 상하 대칭된 800 * 600 jpg color image 출력	P	구현 완료
3.2.	(5), 좌우 대칭	화면에 좌우 대칭 된 277 * 182 bmp color image 출력	P	구현 완료
4.1.	(2), hello, world! 입력	다른 image editor에서는 열리지 않는 암호화된 (2) image file	P	구현 완료
4.2.	4.1.의 expected output, hello, world! 입력	다른 image editor에서도 열리는 복호화된 (2) image file	P	구현 완료
5.1.	(1), jpg → bmp 선택	800 * 600 해상도의 bmp color image	P	구현 완료
5.2.	(2), jpg → png 선택	1123 * 729 해상도의 png color image	P	구현 완료

Spec	input	Expected Output	Pass / Fail	comment
5.3.	(3), png → bmp 선택	304 * 300 해상도의 bmp color image	P	구현 완료
5.4.	(4), png → jpg 선택	512 * 512 해상도의 jpg color image	P	구현 완료
5.5.	(5), bmp → png 선택	277 * 182 해상도의 png color image	P	구현 완료
5.6.	(6), bmp → jpg 선택	640 * 559 해상도의 jpg color image	P	구현 완료

6.1.	(1), (1) 내부를 마우스 커서로 drag 한 영역	화면에 Drag한 영역 만큼의 해상도를 가지는 (1) 내부의 jpg color image 출력	P	구현 완료
7.1.	(3)을 띄운 출력 창 선택 3번 확대	화면에 (3)이 300% (3배) 확대된 imag e 출력	P	구현 완료
7.1. 1.	키보드 → 화살표	확대된 image가 출력 창 크기를 초과하는 경우 확대된 (3)의 오른쪽 부분으로 이동하여 출력	P	구현 완료
7.1. 2.	(3)을 3번 확대한 출력 창 선택 1번 축소	화면에 (3)이 200% (2배) 확대된 Image 출력 (확대 단계 1단계 감소)	P	구현 완료
8.1.	4.2. expected output, 원하는 directory, file nam e, file extension	4.2.의 expected output을 지정한 dir ectory에 지정한 file extenstion format과 file name으로 저장	P	구현 완료
8.2.	(6), (6) file이 있는 directory 경로	(6) Image file을 창에 출력	P	구현 완료
8.3.	6.1. expected output	(1)출력 (expected output 잘못 기입)	P	구현 완료
8.4.	8.3. expected output	화면에 6.1. expected output	P	구현 완료

전체 test case 24개. 총 통과 test case 24개. Pass & Fail ratio = 24/24 = 100%.
다만 2.3.의 경우 완벽하게 통과는 아님에 유의해야 한다.
381 * 381 계산 수식이 어려워 자유 회전 27° 에 초점을 맞추었을 때의 기준임.
해당 부분을 △ 로 판단 시 23.5/24 = 97.9 %.

5.3 Success Criteria 대비 달성률

(1) 초창기의 Success Criteria

- 완성도에 대한 기준

핵심 기능 : 포맷 변환, Steganography 디지털 워터마크 DRM 기술 구현
(전체 개발 기간의 80% 예측 — 80% 구현의 기준)
부가 기능 : 대칭, 회전 기능 구현
(전체 개발 기간의 20% 예측 — 20% 구현의 기준)
⇒ 초창기에 작성했던 Success Criteria 이므로 현재 project에 작성하기에는 무리가 있음.
(GUI 추가, 각종 기능의 추가, DRM 기술의 구현 및 적용 방법 변경으로 인해 재사용하기 어려움)

- 각 기능의 구현 성공에 대한 기준

포맷 변환 : 파일 속성 output message로 확인 ⇒ 확인 가능함.

대칭, 회전 : 변환 후 이미지를 별도의 창으로 띄워 실제로 확인. ⇒ 확인 가능함.

Steganography 디지털 워터마크 DRM

⇒ Steganography의 경우 정보 은닉이 목적이므로존재 여부는 따로 알려주지 않음.

※ 확인 절차

1) Steganography 기법으로 삽입할 image와 워터마크가 삽입될 image를 준비함

2) Steganography 기법으로 디지털 워터마크를 삽입

3) 다시 추출 했을 때 육안으로 동일하다 식별 가능한 디지털 워터마크 image가 나와야 함

(2) 현재에 맞게 수정되어야 할 Success Criteria

- 완성도에 대한 기준

핵심 기능 : 포맷 변환, AES-256 및 SHA-256 기반의 DRM 암호화 기술

⇒ 포맷 변환 구현 완료 및 test case에 대한 test 완료.

⇒ DRM 암호화 기술 구현 완료 및 test case에 대한 test 완료

부가 기능 : 대칭, 회전, 잘라내기, 해상도 조절

⇒ 대칭 구현 완료 및 test case에 대한 test 완료.

⇒ 회전 구현 완료,

하지만 자유 각도 회전의 경우 해상도 조절에는 미흡한 부분이 존재.

그 외에는 test case에 대한 test 완료.

⇒ 잘라내기 구현 완료. 실제로 마우스로 드래그하여 입력받는 것까지 구현 완료

test case에 대한 test 완료.

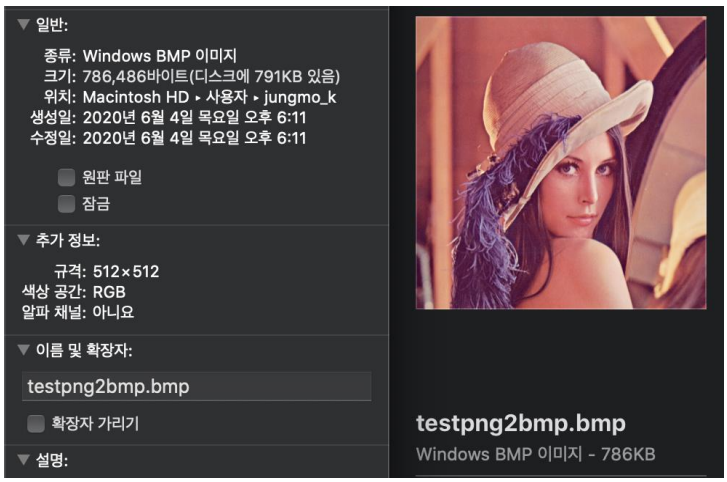
⇒ 해상도 조절 구현 완료. 확대 시 시간이 시간이 많이 소요.

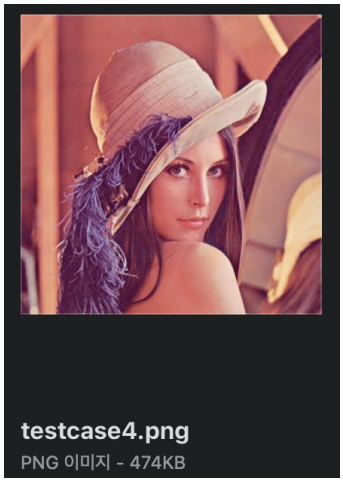
→ non-functional requirement의 5초 제한은 지키지 못했으나 이는 python이라는 개발 환경 자체의 문제도 존재함.

test case에 대한 test 완료.

- 각 기능의 구현 성공에 대한 기준

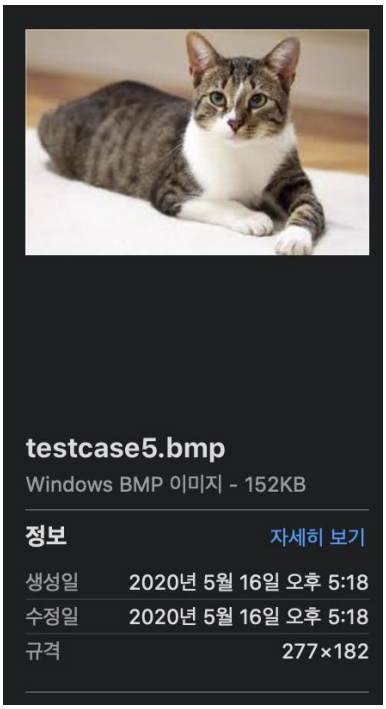
포맷 변환 : 파일 속성을 통해 제대로 이루어졌음을 확인 가능함.



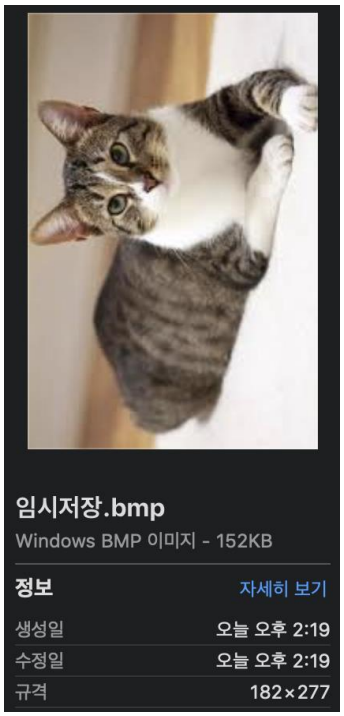


대칭, 회전, 잘라내기 : 실제 GUI를 추가하여 육안으로 결과 확인 가능

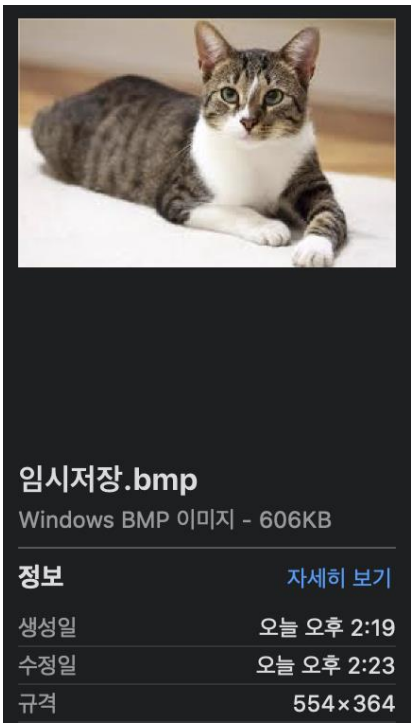
속성 창의 가로 세로 해상도를 통해 실제로 대칭 / 회전이 되었는지 확인이 가능함.



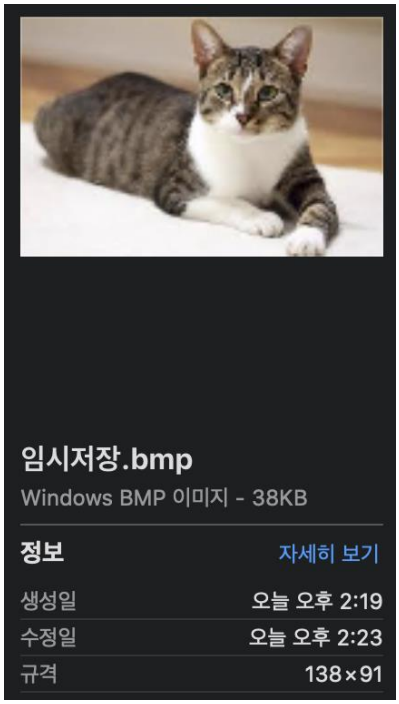
testcase5.bmp



90도 회전한 경우

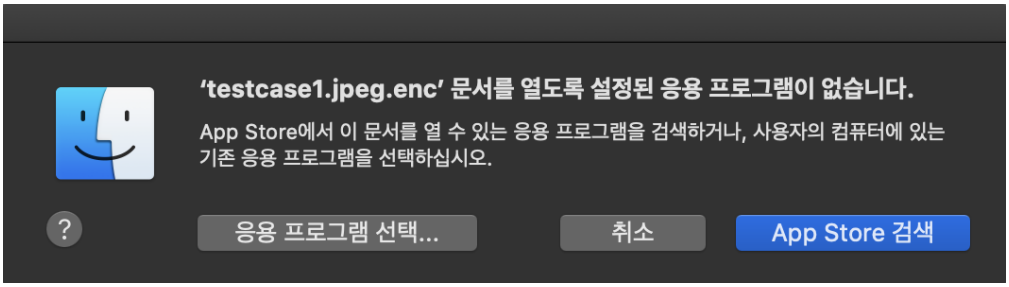


해상도 확대한 경우

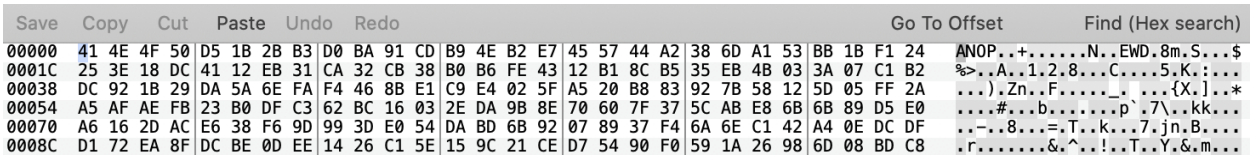


해상도 축소한 경우

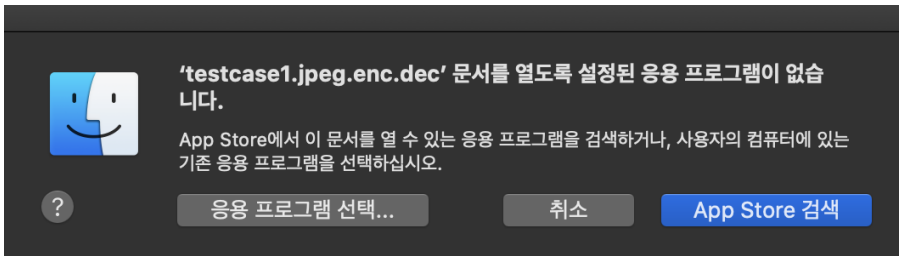
AES-256 / SHA-256 기반의 암호화 : 암호화 여부 및 파일 확인 가능

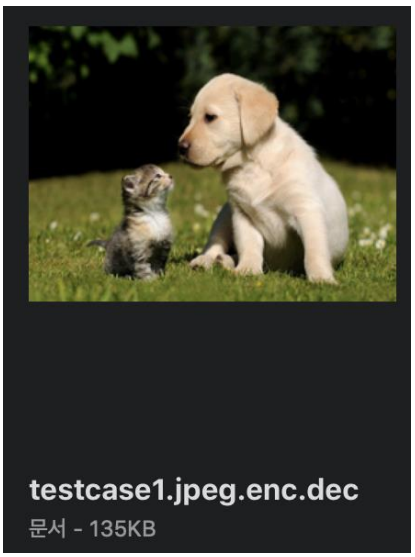


일반적인 파일 확장자 및 signature가 아니므로 열 수 있는 프로그램이 없다고 뜸.



Signature 부분이 우리 프로젝트를 나타내는 ANO로 변했음을 확인 가능.





복호화의 경우 파일 확장자 때문에 열 수 있는 프로그램이 없다고 뜨지만, 파일 binary data 자체는 I mage data 이므로 이렇게 정상적인 image로 출력됨을 확인할 수 있다.

결과

- 전체적인 Success criteria :
  ⇒ 자유 회전 시 해상도 맞추는 부분과, JPEG 디코딩 / 해상도 확대 / 자유 각도 부분의 non functional requirement(performance) 부분을 제외하고는 모두 충족.

- python 이라는 언어의 환경을 고려하고, 수식적인 부분을 제외한 기능 개발 요건 측면에서 봤을 때
  - Success criteria 충족 및 test case all pass = 24 / 24 = **100%**

- python 이라는 언어의 환경과 수식적인 부분 및, non-functional requirement의 performance를 고려해 success criteria 입장에서 보았을 때
  - Success criteria 충족 및 test case 일부 제외 충족 = 23.5 / 24 = **97.9%**
    (test case 2.3 절반 충족했다 가정 시)