

## 캡처 과정

### 1. 초기화

가장 처음 사용자에게 고유식별번호(ucode)를 부여한다. 고유식별번호는 사용자마다 고유하게 부여되며 6자리의 영어 대문자로 구성된다. 랜덤으로 생성하며 기존 DB에 동일한 코드를 가진 사용자가 존재하는지 검사하며 존재하지 않는 경우 디바이스의 기기식별번호(pid)를 가져와 데이터베이스(user\_code.db)에 저장한다(필요시 pid 암호화). 또한, 고유식별번호와 캡처 번호(날마다 0000으로 초기화, 최대9999를 가지며 그 이상 캡처는 불가능하게 금지함)를 사용자의 디바이스에 저장한다.

user\_code.db

ucode	pid
MGQOHR	Q18MK5XI9M
MWMKKF	B47QS6ZY0P
TRGRGK	H43JW2ZL5R

그림1. user\_code.db 구성 예시

### 2. 캡처

#### 2-1) 사용자 정보 DB 저장 및 워터마크 이미지 생성

지정된 웹사이트에서 캡처를 하면 캡처 당시의 사용자 정보가 DB에 저장되고, 그 정보를 기반으로 캡처한 이미지에 삽입될 워터마크 이미지를 생성한다.

DB에 삽입될 사용자 정보는 idx(워터마크 문구), year(YYYY), month(MM), day(DD), time(TT:TT:TT), ip(ip주소), device(모델명), pid(기기식별번호), url(캡처한 웹사이트 주소)이다. 이때, ip, device, pid는 미리 생성한 key를 이용해 암호화된 후 DB에 저장된다.

워터마크 문구는 (사이트 고유문자)+(월일)+(ucode)+(숫자)로 구성된다. 캡처한 웹사이트마다 사이트 고유문자가 지정되어 있으며, (ucode)와 (숫자)는 '1.초기화'에서 사용자 기기에 저장한 정보를 이용한다. 이때 숫자는 한 번 캡처할 때마다 1씩 증가하며 최대 9999가 넘는 경우 캡처를 금지한다.

워터마크 문구를 구성하면 DB에 함께 저장한다. 워터마크 이미지의 크기는 (148,13)으로 동일하다. 나눔스퀘어 폰트 10pt를 이용해 문구를 작성한 후 이를 이미지로 바꾸어 워터마크 이미지를 생성할 수 있다.

DAU0614MGQOHR0000

그림2. 워터마크 이미지 예시

#### 2-2) 워터마크 삽입

워터마크 삽입은 크게 두 단계로 나뉘어진다. 우선 공간영역에 LSB를 이용해 가로/세로 선

을 삽입해 위치정보를 나타낸다. 이후 이미지를 YUV 채널로 분리한 후 Y채널만 dwt1으로 변환한다. DWT1 변환 후 L영역을 8x8 block으로 나눈 후 각 block마다 DCT-SVD를 적용한다. SVD에서 s벡터의 첫 번째 값에 워터마크를 반복적으로 삽입한다. 삽입이 끝나면 역 SVD, 역 DCT변환을 거치고 모든 block에 워터마크 삽입이 종료되면 원래 shape으로 만든 후에 IDWT 변환을 적용해 워터마크가 삽입된 이미지를 얻는다. Lena(512x512)이미지는 약 0.16초정도의 삽입 시간이 소요되고, 고해상도(1920x1080) 이미지는 약 1.6초의 시간이 소요된다.



그림3. 원본 이미지와 워터마크 삽입 이미지

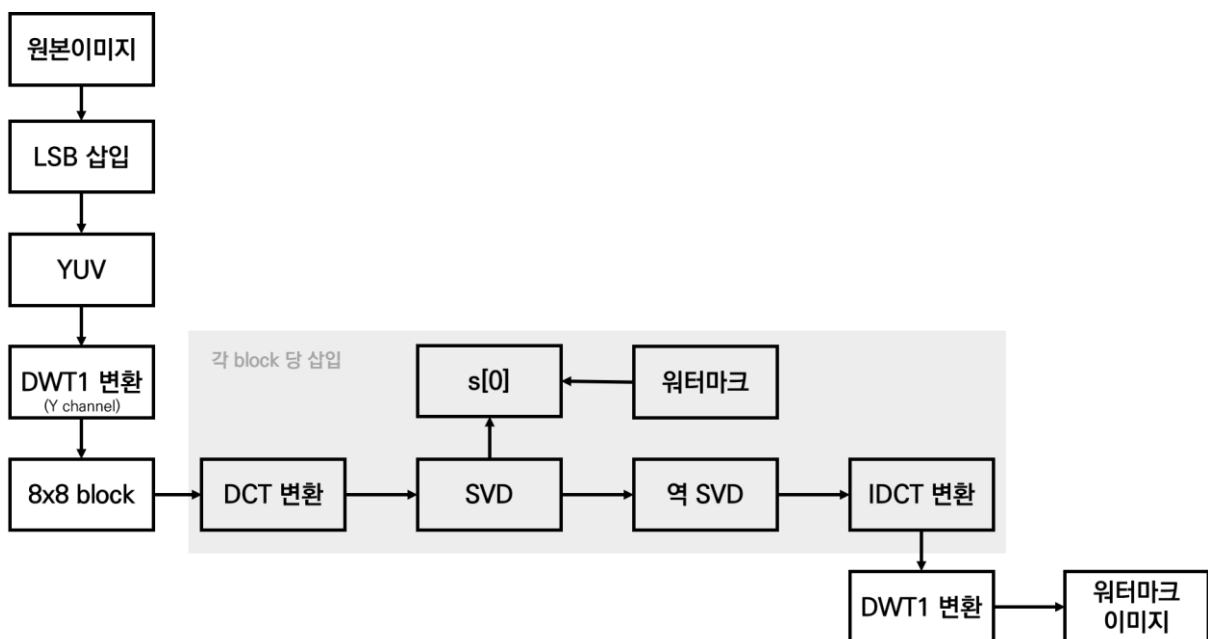


그림4. 워터마크 삽입 과정

### 3. 워터마크 추출

워터마크 추출 시 이미지가 crop되었는지 확인하기 위해 이미지의 크기가 화면 크기와 동일한지 확인한다(현재는 가상 환경으로 캡처 이미지의 크기를 안다고 가정하였으며, 실제 사용시에는 모든 화면 크기에 대해 비교하고 복원해야 한다).

화면 크기가 동일한 경우(원본 화면 크기를 알고 있다고 가정), 복원 과정 없이 바로 워터마크를 추출한다. 이때, 원본 이미지 없이도 추출(Blind)이 가능하다. 워터마크 된 이미지를 YUV 채널로 변환한 후, 이 중 Y 채널에 DWT1변환을 적용한다. 변환 후 L영역을 8x8 block으로 나누어 각 block에 DCT-SVD 변환을 통해 삽입한 워터마크의 정보를 추출한다.


워터마크가 삽입된 이미지	삽입한 워터마크
	DAU0614MGQOHR0000
	추출한 워터마크
	DAU0614MGQOHR0000

그림5. 워터마크 된 이미지로부터 추출한 워터마크

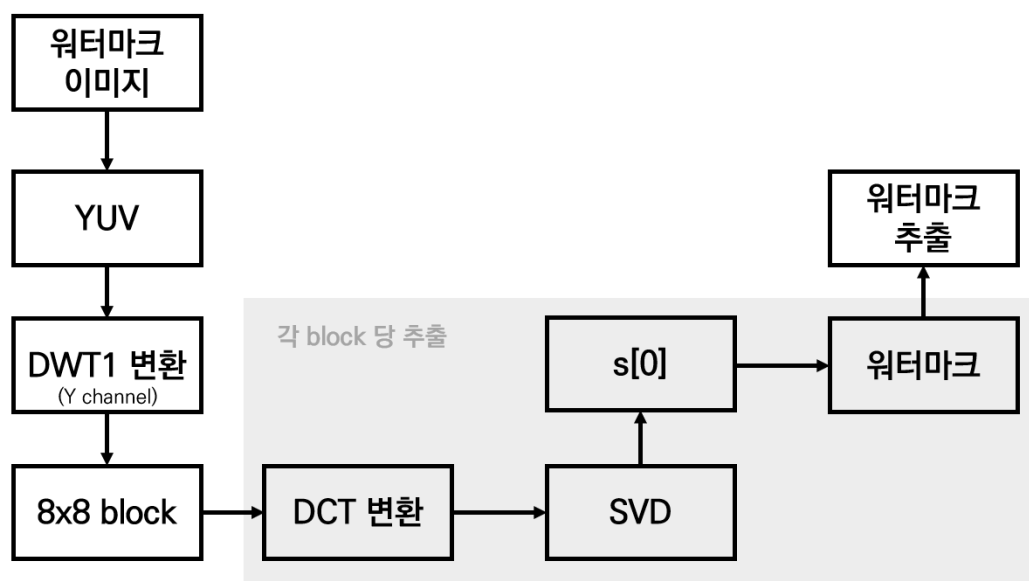


그림6. 워터마크 추출 과정

화면 크기가 동일하지 않은 경우, crop된 이미지를 원래 크기로 복원해야 한다. 복원할 때는 LSB를 활용해 공간영역에 삽입한 위치 정보를 다시 추출한다. 현재는 가로/세로 선이 교차하는 중간 영역이 모두 포함된 경우에만 복원이 가능하도록 작성하였다. 추출하면 그림7과 같은 위치정보를 확인할 수 있다.



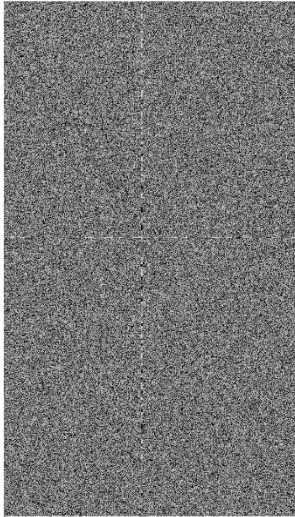
워터마크가 삽입된 이미지	크롭된 이미지	위치정보 추출
		

그림7. Crop된 이미지로부터 위치정보 추출

추출한 위치정보에서 가로/세로 선을 확인할 수 있고 이를 기준으로 원래 크기로 이미지를 복원한 후 앞서 동일한 과정을 거쳐 워터마크를 추출할 수 있다.

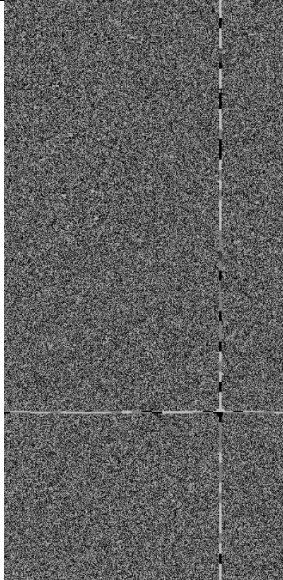


위치정보 확대	복원한 이미지	워터마크 추출
		

그림8. 위치정보를 이용해 복원한 이미지로부터 워터마크 추출



#### 4. OCR 인식 및 사용자 추적

추출한 워터마크 이미지로부터 OCR을 이용해 글자를 인식한다. 현재 워터마크 크기가 작아 OCR 인식이 제대로 되지 않는 경우가 존재한다. 그림9는 제대로 글자를 인식했을 때의 결과이다.

추출한 워터마크 이미지	OCR 인식(+전처리) 결과
DAU0614MGQOHR0000	: 'DAU0614MGQOHR0000'

그림9. OCR 인식 결과

OCR 후 얻은 idx정보를 이용해 DB로부터 사용자를 찾고, KEY table로부터 각각의 key를 가져와 ip/device/pid 정보를 복호화한다.

```
1 print(suspect)
{'idx': 'DAU0614MGQOHR0000', 'year': 2020, 'month': 6, 'day': 14, 'time': '12:46:11', 'ip': '119.25.133.114', 'device': 'SM-G930S', 'pid': 'Q18MK5XI9M', 'url': 'http://webtoon.daum.net/webtoon/viewer/81878'}
```

그림10. 사용자 추적 결과

#### 5. 이미지 화질 평가

원본 이미지와 워터마크가 삽입된 이미지를 비교한다. 비교를 위해 PSNR, SSIM, NCC 기준을 사용했다. 전체 이미지에 대해 화질을 평가한 결과 표1과 같다. RGB 각 채널에 대해서도 화질을 비교했을 때 PSNR이 40dB이상의 값을 가져 최종 목표를 달성할 수 있었다.

PSNR	SSIM	NCC
45.57	0.99	0.99

표1. 전체 이미지에 대한 화질 비교

	PSNR	NCC
R	45.56	0.99
G	45.57	0.99
B	45.58	0.99

표2. 각 채널에 대한 화질 비교

#### 6. 이미지 공격

##### 6-1) JPEG compression

워터마크를 삽입한 PNG 이미지를 JPG 변환시킨 후 워터마크를 추출해보았다. 노이즈가 조금 추가되었지만 눈으로 글자를 인식할 수 있을 정도의 결과를 얻었다.




워터마크가 삽입된 이미지	JPEG compression	워터마크 추출
		

그림11. JPEG 압축 공격

## 6-2) Median/Average/Gaussian filtering

워터마크를 삽입한 이미지에 Median filtering, Average filtering, Gaussian filtering을 각각 적용한 후 워터마크를 추출해보았다. Median filtering 결과 노이즈가 심해 시각적으로도 글자 인식이 어려웠지만 이미지에 따라 추출되는 정도가 달랐다. 다른 이미지에서는 글자 인식이 가능한 정도로 추출된 경우가 존재했다. Average와 Gaussian 필터링 후 추출한 워터마크에 노이즈가 있지만 시각적으로 명확하게 글자를 인식할 수 있었다.




Median	Average	Gaussian
		

그림12. Filtering 공격

## 6-3) 이미지 변형(이미지 위 다른 이미지 추가)

워터마크를 삽입한 이미지에 다른 이미지를 추가했을 때 워터마크를 추출하였다. 그 결과 아주 약하게 흐릿한 부분이 존재하였고 시각적으로 명확하게 글자를 인식할 수 있었다.

워터마크가 삽입된 이미지	이미지 변형	워터마크 추출 결과
---------------	--------	------------

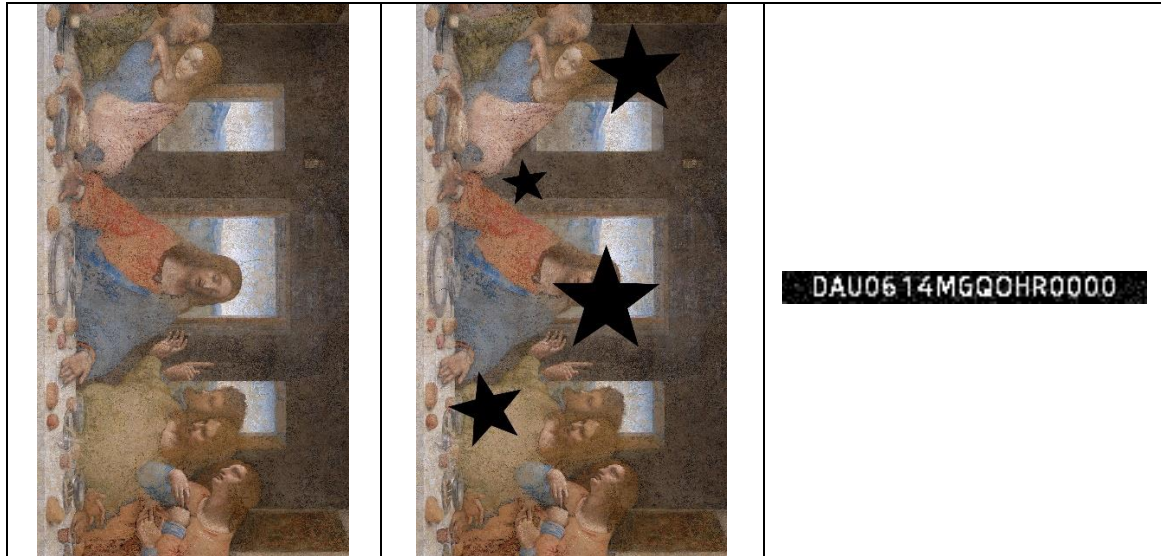


그림13. 이미지 변형

## 결과 및 한계점

캡처 시 사용자의 정보를 DB에 추가함과 동시에 캡처 이미지에 사용자의 정보를 찾을 수 있는 정보를 이미지화 시켜 삽입하고, 워터마크를 추출해 다시 사용자 정보를 찾을 수 있다. 기존 연구는 주로 저화질 흑백 이미지를 이용해 Robust한 워터마크 삽입을 목적으로 했다면, 본 연구는 실생활에서의 활용도를 고려해 고화질 컬러 이미지에 상대적으로 빠른 속도로 워터마크를 삽입하는 것을 목적으로 했다. 또한, 그림 14처럼 기존 논문에서는 crop 시 좌측처럼 원래 이미지의 크기가 유지된 상태에서 워터마크를 추출했고, 우측의 crop에서는 추출이 거의 불가능했다. 이를 보완하기 위해 본 연구에서 공간 영역에도 위치 정보를 삽입하였고, 캡처 화면이기 때문에 이미지의 크기가 한정된 점을 이용해 원래 크기로 복원시켜 워터마크 추출을 가능하게 했다.



그림14. 이미지 변형

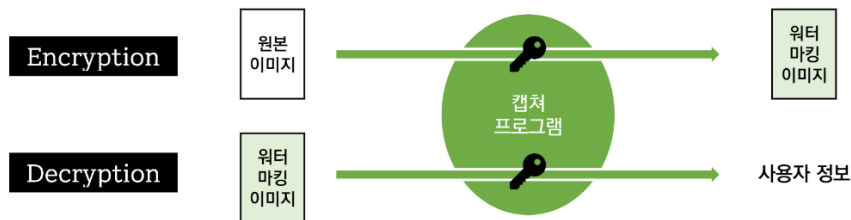
단, Crop된 이미지의 복원을 위해 특정 조건이 만족되어야 한다는 한계점을 가진다. 우선 앞서 6-1), 6-2)처럼 이미지를 손상시키는 공격이 들어가지 않은 PNG 이미지여야 하며 워터마크를 충분히 추출할 수 있을 정도의 영역이 남아있어야 한다. 이를 위해 글자를 이미지화 한 워

터마크의 크기를 줄여 한 이미지에 많이 삽입될 수 있도록 했다.

워터마크를 많이 삽입할수록 crop에 강인하지만 반대로 워터마크 삽입 시간이 오래 소요되는 trade-off가 발생했다. 적절히 빠른 삽입 속도와 공격에도 강인한 워터마크 이미지 생성을 위해 DWT1 변환을 통해 L영역에 삽입하되 8x8 block을 사용해 워터마크를 삽입했다. 하지만 6에서 언급한 공격 외에 salt & pepper 등 공격을 적용하면 워터마크를 추출하지 못하는 문제점을 보여 공격에 보다 강인하도록 성능 향상이 필요했다.

캡처 시스템에서 외부로 키가 노출되지 않는다는 특징이 존재하여 사용자 정보를 암호화할 때 대칭키 방식을 이용하였다. 암호화 시 필요한 key는 최초 1회만 생성하고 ip, device, pid 각각 한 개 씩 가지며 모든 사용자에게 동일하게 적용한다.

### 3) 캡처 프로그램 특이점



1. 사람마다 다른 대칭키를 사용하면 복호화 시 **brute force** 방식으로 모든 키를 적용해서 찾아야 함
2. 캡처 프로그램은 사용자 간 **대칭 키 교환이 불필요** (암호화 및 복호화 시 자체 중재자 역할 수행)

동일한 대칭 키 사용해 암호화 및 복호화

장점 : 속도가 빠름  
단점 : 키 교환하면서 대칭키가 노출될 수 있음