

2009 졸업연구 결과보고서

스टे가노그래피를 활용한 정보은닉 응용기법 연구

Steganography

팀 명 히든카드(Hidden Card)

팀 원 이 철
 류 수 형
 김 영 동
 엄 광 용

2009. 10

중부대학교 정보보호학과

1. 연구제목

스테가노графи를 활용한 정보은닉 응용기법 연구

2. 연구 목적 및 필요성

- 최근 디지털 저작권 관리(DRM : Digital Rights Management) 분야가 정보보호기술의 중요한 부문으로 부각
- 따라서 디지털 저작권 관리의 핵심기술인 Watermarking, Fingerprinting의 기본이 되는 Steganography를 활용한 정보은닉 응용기법을 연구하여 정보보호기술의 응용력을 배양하는 것이 바람직하다.

3. 연구 내용

- 스테가노그래피 기법 조사 : 인간 시각 시스템을 기반으로 하는 LSB(Least Significant Bits) 기법 등 스테가노그래피 기법에 대한 자료조사 및 연구. 시험용 원본 영상 확보 및 은닉 영상 제작
- 적용 가능한 암호기법 조사 : 은닉 영상에 적용 가능한 암호기법(DES, Transposition Cipher 등) 조사 및 적용방법 연구
- 스테가노그래피 기법과 암호화 기법을 병합하는 방법 연구 및 취약점 분석을 통해 기존 은닉 영상 추출기법에 강인성을 가지는 암호화 기법을 확정
- 스테가노그래피와 암호화 병합 방법을 프로그래밍에 반영하기 위한 프로그래밍 기법과 활용 가능한 함수 기능 확인
- 확정된 방법을 시스템으로 구현하기 위한 시스템 구성방안을 마련하고 실험환경을 구축 하고 기법별 프로그래밍 작업 개시
- 프로그래밍 작업과 병행하여 연구 결과를 워터마킹 및 핑거프린팅에 접목시키는 방안을 연구, 최종 결과 보고에 반영

4. 연구 결과

(1) 목표 : 보호대상 자료(은닉 영상)에 적합한 암호기법을 적용하여 변환 후 화질 저하를 최소화하는 조건에서 원본 영상에 삽입하고, 기존의 추출기법으로는 은닉 영상을 복원할 수 없는 수준의 스테가노그래피를 구현

(2) 결과의 활용 : 스테가노그래피 연구 결과를 디지털 콘텐츠의 저작권 보호나 불법 유통 추적 등에 활용되는 워터마킹 및 핑거프린팅에 적용하는 방안까지 제안하는데 목적을 두고 연구를 진행

목 차

1. 서론	6
2. 관련연구	7
1) DRM	7
2) 스테가노 그래피	10
(1) 스테가노 그래피 원리	11
(2) Cover 이미지 실험영상 해설	12
3. 스테가노 그래피 삽입방법	13
1) 비트플레인 삽입 방법	13
2) Permutation 병합 방법	14
3) 비트 플레인 분산삽입 방식	18
4) Pack 형식	19
5) Pack형식 embedding 방식	22
4. 결론	25
부록	27
참고문헌	51

그림 목차

(그림 가1-1 스테가노그래피 원리)	11
(그림 가2-1 Lena 영상 해설)	12
 (그림 나1-1 비트 플레인 삽입방법)	13
 (그림 나2-1 Permutation 병합 방법 원리)	14
(그림 나2-2 Logo 이미지 Permutation 실행화면)	14
(그림 나2-3 Permutation 에 쓰일 cover image, logo image)	15
(그림 나2-4 Permutation 작업후 비트플레인 삽입 결과 화면)	15
(그림 나2-5 Permutation 프로그램 실행화면)	16
(그림 나2-6 Permutation 실행 결과)	17
(그림 나2-7 Permutation logo 파일 적용 결과)	17
 (그림 나3-1 LSB 방식의 비트 플레인 삽입방식)	18
(그림 나3-2 비트플레인 분산 삽입 방식 이해)	19
 (그림 나4-1 Pack 형식 프로그램 실행 화면)	20
(그림 나4-2 logo-unpack.exe 실행화면)	22
(그림 나4-3 Pack 형식 logo 파일 실행 전, 후 결과)	22
 (그림 나5-1 비트 플레인분산 방식 embedding bit-emd.exe 실행화면)	23
(그림 나5-2 비트 플레인분산 방식 embedding bit-ext.exe 실행화면)	24

1. 서론

최근 인터넷의 보급 확산과 영상, 오디오 및 동영상 등 대용량 디지털 자료의 가공·처리기술 발전에 편승하여 이들 디지털 콘텐츠의 불법적인 복제나 유통으로 인한 지적재산권 보호문제가 크게 부각되고 있다.

디지털 콘텐츠의 불법 복제와 유통은 저작권자에게 심각한 경제적 손실을 입힐 뿐만 아니라 창작 의욕을 상실시키는 심각한 경제·사회적인 문제이다. 따라서 디지털 콘텐츠의 건전한 거래 질서를 확립하고 창작 활동을 활성화시키기 위해서는 디지털 콘텐츠에 대한 안전하고 견고한 보호 기술 연구와 실용화가 요구되고 있다.

1) 연구 개요

- 최근 디지털 저작권 관리(DRM : Digital Rights Management) 분야가 정보보호 기술의 중요한 부문으로 부각.
- 따라서 디지털 저작권 관리의 핵심기술인 Watermarking, Fingerprinting의 기본이 되는 Steganography를 활용한 정보은닉 응용기법을 연구하여 정보보호기술의 응용력을 배양하는 것이 바람직하다.

2) 관련연구

- **Watermarking** : 디지털 콘텐츠 원본에 사람의 청각이나 시각으로 인식할 수 없는 고유의 표식 등을 삽입, 저작권을 표시하고 불법 복제를 방지하는 기술. 즉 저작권 정보를 삽입, 불법 유통의 문제가 발생 시 저작권자를 확인하는 수단으로 활용.
- **Fingerprinting** : 디지털 콘텐츠에 구매자 사용내역 정보를 삽입, 사후에 발생할 수 있는 불법 복제자를 추적하는데 사용되는 기술. 즉 사용자 정보를 삽입, 불법 유통 문제가 발생 시 불법 유통자를 추적하는데 활용.
- **Steganography** : 이미지, 오디오, 비디오 또는 텍스트 등의 디지털 미디어에 전달 하고자 하는 비밀 메시지를 숨겨서 전송하는 기술적 방법. 즉 디지털 자료에 정보를 은닉하는 기술.

3) 연구목표

1. 스테가노그래피 기법 조사 : 인간 시각 시스템을 기반으로 하는 LSB(Least Significant Bits) 기법 등 스테가노그래피 기법에 대한 자료조사 및 연구. 시험용 원본 영상 확보 및 은닉 영상 제작.
2. 적용 가능한 암호기법 조사 : 은닉 영상에 적용 가능한 암호기법(DES, Transposition Cipher 등) 조사 및 적용방법 연구.
3. 기존방법의 취약점 해결 : 스테가노그래피 기법과 암호화 기법을 병합하는 방법 연구 및 취약점 분석 을 통해 기존 은닉 영상 추출기법에 강인성을 가지는 암호화 기법을 사용.
4. 현재 나와있는 기존 자료들을 수집하여 어떤 방법들이 이용되고 있으며, 어떻게 되는지 조사.
5. 확정후 시스템으로 구현하기 위한 시스템 구성방안을 마련하고 실습환경을 구축 하고 기법별 프로그래밍 작업 개시.
6. 프로그래밍 작업과 동시에 Image 로 쓰일 자료들을 제작 과 발표용으로 제작.

2. 관련연구

1. DRM (Digital Rights Management)

1) DRM 기술의 정의

‘Digital Rights Management’의 약자로 디지털 저작권 관리를 의미한다. 콘텐츠 제공자의 권리와 이익을 안전하게 보호하며 불법복제를 막고 사용료 부과와 결제대행 등 콘텐츠의 생성에서 유통·관리까지를 일괄적으로 지원하는 기술이다. 여기에는 적법한 사용자만 콘텐츠를 사용하고 적절한 요금을 지불하도록 만드는 디지털 저작권 관리기술, 저작권 승인과 집행을 위한 소프트웨어 및 보안기술, 지불·결제기술이 모두 포함된다. 음악공유서비스로 유명한 냅스터가 2001년 MP3 저작권 보호를 위하여 채택한 것이 시초이며, 온라인 콘텐츠가 유료화되며 중요한 기술로 떠올라 미국 매사추세츠공과대학에서는 미래 10대 핵심정보기술로 선정하였다.

콘텐츠 식별자인 DOI(Digital Object Identifier), 전자상거래에 필요한 데이터를 기록하는 인덱스(INDECS), 불법복제와 변조방지를 위한 워터마킹 기술을 뒷받침으로 하고 있다. DOI는 디지털 콘텐츠에 부여하는 식별번호이며, 워터마킹은 기밀정보를 디지털 데이터에 숨긴 후 저작권 분쟁이 발생했을 때 디지털 저작권자가 누구인가를 확인할 수 있는 기술이다. 콘텐츠마다 보안인증시스템을 장착하여 일정한 사용료를 지불하지 않으면 그 콘텐츠를 이용하지 못하게 한다.

2) 디지털 콘텐츠 권리 (DRM) 에 대해 알아보자

DRM[디알엠]은 웹을 통한 유료 콘텐츠의 안전한 배포를 보장하고, 보다 중요한 것은 불법 배포를 방지하기 위해 개발된 서버 소프트웨어의 한 종류이다. DRM 기술은 냅스터와 같은 사용자간 파일 교환 프로그램들의 광범위한 사용으로 급격히 증가된 상용 제품의 온라인 프라이버시 보호 수단으로 개발되고 있다. 비록 온라인 콘텐츠가 저작권법에 의해 보호 받고는 있다지만, 불법적인 웹 사용을 단속하고 범법 행위자를 잡는 것은 현실적으로 매우 어렵다. DRM 기술은 온라인 밀렵꾼을 범행이 일어난 후 체포하는 마구잡이식 전략보다는, 보다 확실한 문제 해결 접근방식으로서 애당초 웹 콘텐츠를 훔치는 것 자체가 불가능하도록 초점을 맞춘다. 많은 회사들이 다양한 접근방법과 기술에 기반을 둔 여러 가지 DRM 제품들을 내놓고 있다. 일반적으로 DRM 제품들은 서버 소프트웨어와 사용자용 플러그인 등, 운영에 필요한 모든 것들이 포함되어 있는 일괄 패키지 형태를 갖추고 있다.

반 복제와 반 접근 수단에 대한 법적 보호

위터마크를 한 종이에서부터 디지털 위터마크까지, 저작권자는 오랫동안 복제를 발견 또는 방지하기 위한 기술적인 수단을 펴하고 있다. 일찍이 컴퓨터 게임 프로그래머들은 복제된 게임들이 정확하게 작동되지 않도록 하는 코드를 기억장치에 기록하는 것과 같은 “혼란과 약간의 형식 변경”에 의존했었다. 허가 되지 않은 복제본의 유통 막기 위해서, 어떤 게임들은 사용자들에게 “매뉴얼의 17페이지의 첫번째 단어는 무엇인가?” 와 같은 질문에 대답하도록 요구한다. 그래서 그 게임을 하려면 매뉴얼의 복제도 필요하다.

사용자들은 소프트웨어를 등록 해야 할 필요가 있거나, 특별한 시리얼 번호를 가지고 있거나, 동글에 접속 해야 한다. 비디오 테이프, 오디오 CD, CD-ROM, DVD 그리고 또 다른 미디어들은 마찬가지로 복제를 통제하기 위해 다양한 시도를 하고 있다.

그러나 단지 그들은 잠재적인 침해자들을 막았기 때문에, 그들은 때때로 허가 받은 사용자에게 의한 사용을 막았다. 역사가, 취미 생활자 그리고 기록 보관자들은 지금 때때로 오래된 저작물을 연구하기 위하여 오래된 소프트웨어에 대한 복제 방지 장치들을 빠져나가기 위해 몸부림치고 있다.

더구나 최근에, 디지털 콘텐츠 권리 관리(DRM) 기술들은 무허가 사용을 막는 것과 실시를 허용 하는 시스템을 촉진하는 것 둘 다 추구하고 있다. 이상적으로 (저작권자의 입장에서), DRM은 저작권자에 의해 허가 받지 않은 사용을 막는 동안, 다른 부류의 사용자들(개인, 법인)에게 다른 조건에 근거하여 효율적으로 실시 허여를 하기 위한 제품들을 허가해야 할 것이다. DRM은 몇 번이나 저작물을 사용했는지, 어떤 수준의 저작물이 기억장치에 입력될 수 있는지, 그리고 저작권자들이 간접적으로 저작물의 사용을 감시하고 통제할 수 있도록 할 수 있다. 사용자들은 이러한 통제로부터 이익을 얻을 수 있다. 저작권자들이 제공한 저작물을 맞추기 위해 저작권자들에게 허가함으로써, 저작물들은 더 널리 이용될 수 있고 소비자들은 더 많은 선택권을 가지게 되었다.

DRM 체계는 합법적인 사용과 같은 정당한 사용도 막을 수 있다. DRM은 저작권을 취득할 수 없는 요소들에 접근을 통제함으로써, 저작물의 저작권을 취득할 수 없는 요소들을 잠글 수 있다.

DRM 모니터링은 사생활의 문제를 일으킨다. 사용자들 사이의 가격차별은 불공정한 것이다. DRM 제조자의 보증은 항상 기술자들에 의한 정밀한 조사를 저지하지 못한다. 다양한 DRM 시스템은 보증함에 있어 작동하는 것이 실패하고 있거나, 또는 가지고 있는 보증 된 내용에서 사용자를 막고 있거나, 또는 심지어 사용자의 시스템에서 잠재적인 버그 또는 보안상 결점을 소개하고 있다. 현존하는 저작물과 같이 형성 된 저작물의 개작과 발전을 방해함으로써, DRM 시스템

은 또한 혁신을 방해한다.

저작권자들을 위한 새로운 타입의 보호는 디지털 밀레니엄 저작권 법(DMCA)의 한 부분으로써 1998년 저작권법에 추가되어졌다. 1201조는 현재 저작권자가 사용할 수 있는 기술적 수단의 두 타입(복제를 막는 기술과 접근을 막는 기술)에 대하여 법적 보호를 제공한다. 17 U.S.C. § 1201. 전자 디지털 저작물은 복제와 배포가 쉽기 때문에 그 같은 법적 보호의 필요성을 말해 주고 있다. 또한 그 법률은 가입국에 다음과 같이 요구하는 **세계지적재산권 기구(WIPO)**¹⁾ 저작권 조약에 근거하여 필요성이 정당화 되었다 : 이 조약 또는 **베른 협약**²⁾ 과 법에 의해 허가되거나 관계있는 저작권자에 의해 허가되지 않은 그들의 저작물에 관해서 이를 금지하는 법률에 근거한 그들의 권리행사와 관련하여 저작자에 의해 사용 될 수 있는 효과적인 기술 수단의 우회에 대하여 효율적인 법적 구제와 충분한 요건을 갖춘 법적 보호를 제공해야 한다.

WIPO 저작권 협약 11조는 기술적 수단에 관계된 협정이다. 어떤 주식자들은 저작권자의 배타적 권리에 대한 명백한 정의와 기여 침해와 연대 책임을 통하여 미국법은 이미 충분한 요건을 갖춘 법적 보호를 제공한다고 주장한다.

보호되는 기술의 첫번째 부류는 복제를 막는 기술이다. 저작권자는 저작물의 복제방지를 위한 기술적 수단을 사용할 수 있다. 복제-방지 기술은 DVD와 비디오 게임에 대해서는 자주 사용되고, CD나 소프트웨어에는 통상 보다 적게 자주 사용된다. 스트리밍 미디어는 영화의 복제를 할 수 없는 영화를 시청자가 보도록 하기 위해 사용되어 질 수 있다. 복제-방지의 더 오래된 버전은 복제하는 것을 막기 위해 소프트웨어 디스크와 비디오 테이프에 관한 장치들을 포함하고 있다. 그래서 복제-방지 아이템을 소유한 어떤 사람은 그것을 사용할 수 있으나 다른 복제는 할 수 없다.

기술적인 보호의 다른 종류인, 접근하는 것을 막는 것은 허락 없이 저작물에 효과적으로 접근 할 수 있는 것으로부터 저작물의 복제에 접근하는 사람을 막는 것을 의도한다. 영화별 시청료 납부 방식은 종종 스크램블 형식으로 송신하다. 디지털 형식의 어떤 저작물들은 암호를 바꾼다. 그래서 어떤 이가 암호로 바꾼 e-book의 복제를 하였으나 접근을 막는 기술을 우회하는 것 없이는 그것을 읽을 수 없을 것이다. 그러한 기술은 저작물이 사용될 수 있는 하에서 그 상태를 통제하기 위하여 저작권자가 허락하는 것을 의도로 한다. 다양한 규정들은 저작물에 접근을 제한하는 코드를 충족시킬 수 있다.

1) WIPO(World Intellectual Property Organization) 세계지적재산권기구 : 1967년 지식소유권의 국제적 보호의 촉진과 국제협력을 위하여 설립한 국제기구

2) 베른 협약 (문학·예술적 저작물의 보호를 위한 베른 협약) : 저작권 보호에 관한 기본 조약이다. 협약은 저작물이 외국에서 보호받아야 할 최소 조건을 규정한다.

2 . 스테가노 그래피 (Steganography)

스테가노그래피(steganography)는 전달하려는 기밀 정보를 이미지 파일이나 MP3파일등에 암호화해 숨기는 심층암호 기술이다. 예를 들어 모나리자 이미지 파일이나 미국 국가 MP3파일 에 비행기 좌석 배치도나 운행 시간표 등의 정보를 암호화해 전달할수 있다.

물론 위의 방법은 근현대에서 쓰여진 방법이며, 스테가노그래피는 고대 그리스에서부터 이용되어온 것으로 현대 암호학에서 다루는 암호화(encryption)와는 다른 개념이다.

최근 스테가노그래피 기법은 겉보기에나 귀로 듣기에는 암호를 포함했을 때나 그렇지 않을 때나 전혀 차이가 없고, 전송시에도 일반 파일과 동일하게 보인다.

스테가노그래피의 유래는 기원전 5세기로 그리스의 왕 히스테우스는 다이루스 왕의 인질로 잡혀있었다.

그는 밀레투스 에 있는 그의 양아들에게 밀서를 전달하는 방법으로 노예의 머리를 깎고서 그 머리에 메시지를 문신으로 썼다.

노예의 머리카락이 자라서 문신이 보이지 않게 되자 그는 노예를 밀레투스로 보냈다.

이것이 문서로 기록된 인류 최초의 스테가노그래피였다.

고대 로마인들은 과일주스, 우유, 소변같은 자연원료를 이용해 만든 투명잉크를 사용해 편지를 썼다. 이런 편지는 불을 쬐어야만 읽을 수 있다.

이것은 불가시잉크(썼을때 일정 시간이 지나면 보이지 않는 잉크로 요오드용액 등에 넣어야 텍스트를 볼수 있다)같은걸 사용해서 전달하는 정보를 은폐한 것이다.

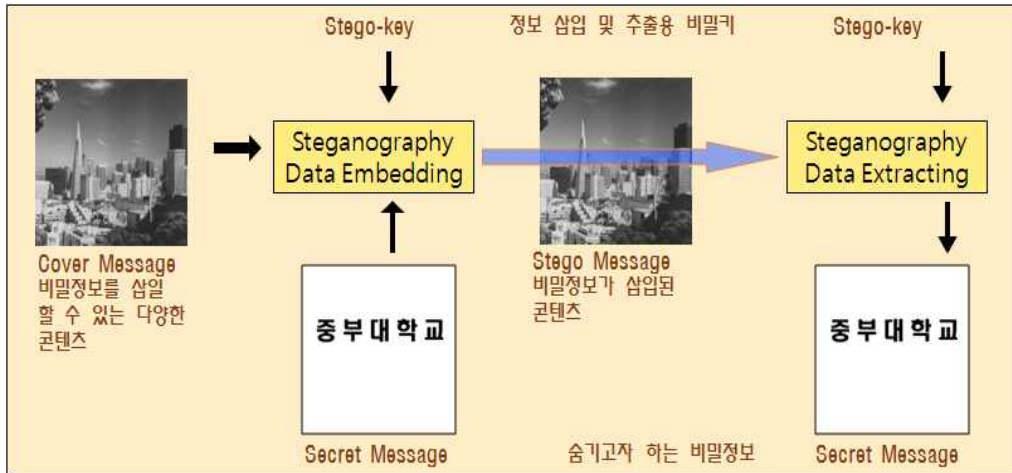
암호의 존재 여부 자체를 숨기기 때문에 해독도 어려울 뿐더러 이것이 암호인지 아닌지를 알아채야 하는 어려움이 가중된다.

2차대전기간 중 독일인들은 "마이크로도트"를 창안했다. 마이크로도트는 말그대로 비밀 메시지를 점 하나의 크기로 축소하는 것이다. 축소된 비밀메세지는 영문자 i의 점에 붙여지는 등 글씨에 포함된 점으로 위장되어 많은 양의 데이터를 감쪽같이 전송했다.

메시지의 존재와 전달상황 자체를 은폐하는 기법으로 가장 해독이 난해한 것으

로 알려진다. 스테가노그래피는 겉으로는 평범하게 보이는 문서에 숨겨진 메시지를 감추어두는 방법으로, 고전적인 방법은 보이지 않는 잉크 등이 있고, 인터넷 시대인 지금은 평범한 웹페이지의 작은 그림속에 메시지를 숨겨두는 방식으로 응용되고 있다.

1) 스테가노그래피 원리



(그림 가1-1 스테가노 그래피 원리)

Stego-key에 따른 분류

- Pure steganography : Stego-key 교환이 필요없는 방식
- Secret-key steganography : 대칭 암호와 같은 비밀키를 사용하는 방식
- Public-key steganography : 공개키와 개인키를 사용하는 방식

Cover Message 를 삽입할 이미지 선택후, secret Message 를 Stego-Key 를 이용하여 embedding 시켜 cover 이미지에 삽입시킨다. 이렇게 삽입시킬때 핵심은 영상 화질 저하를 최소화하는 등 원본의 외형적(화질, 음질 등) 변화를 최소화 하면서 많은 정보를 삽입하는 것이 핵심이다.

이렇게 삽입된 이미지는 육안으로는 쉽게 내용이 들어있다는 것을 알아 볼 수 없게 되어 비밀 메시지를 숨길 수 있다.

다시 반대로 복호화시에는 암호화 시에 사용하였던 Stego-Key를 이용하여 다시 복호화를 사용하여 추출하면 암호화에 사용하였던 Secret Message를 추출할 수 있다.

스태가노그래피의 요구조건

비가시성 (Imperceptibility)

- 통신 여부에 관한 사실 자체가 인지되지 않는 조건
 - 정보가 삽입된 커버 영상은 삽입된 비밀정보에 의해 화질 저하가 없는 조건
- ### 삽입 용량 (Capacity)
- 커버 영상에 의미있는 정보를 전달하기 위해 충분한 데이터량 삽입이 가능
- ### 보안성 (Security)
- 비밀 키 소유자만이 비밀 정보를 추출하고, 숨겨진 비밀 정보의 존재를 확인
- ### 비검출성 (Undetectable)
- 다양한 공격에도 비밀 정보가 검출되지 않고 안전
- ### 강인성(Robustness)
- 삽입된 정보는 의도적 또는 비의도적인 이미지 변형에 의해서도 삭제되지 않는 조건

2) Cover 이미지 실험영상 해설

1. 필요조건 : 다양한 화소값 변화율, 다양한 형태의 외곽선

2. Lena 영상

- 영상의 모델은 스웨덴 출신의 Lena Soderberg
- 1972.11 Playboy 표지 사진(Lenna로 표기)으로

게재된 누드사진의 일부

- 1973. 6-7월 남캘리포니아대 연구원이디지털 영상 연구를 목적으로 Playboy 표지의 Lenna 영상을 스캐닝
- Playboy지는 저작권 침해 사실을 발견 했지만 학술적 사용을 묵시적으로 허용

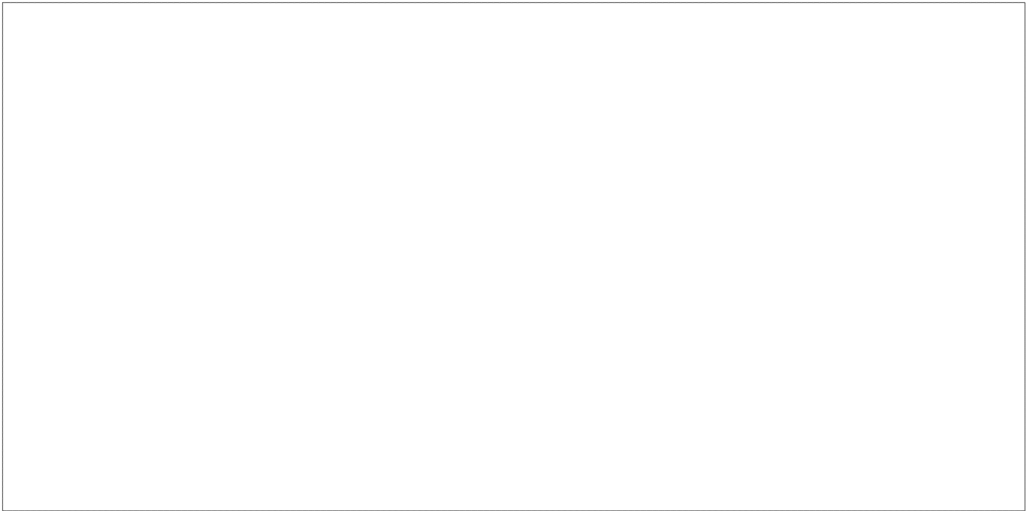


(그림 가2-1 Lena 영상 해설)

3. 스테가노 그래픽 삽입 방법

(1) 비트 플레인(Bit Plane) 삽입 방법

- 영상 화소 값의 특정 비트 위치에 비밀 정보를 적절한 연산방식에 의거 삽입하는 방식



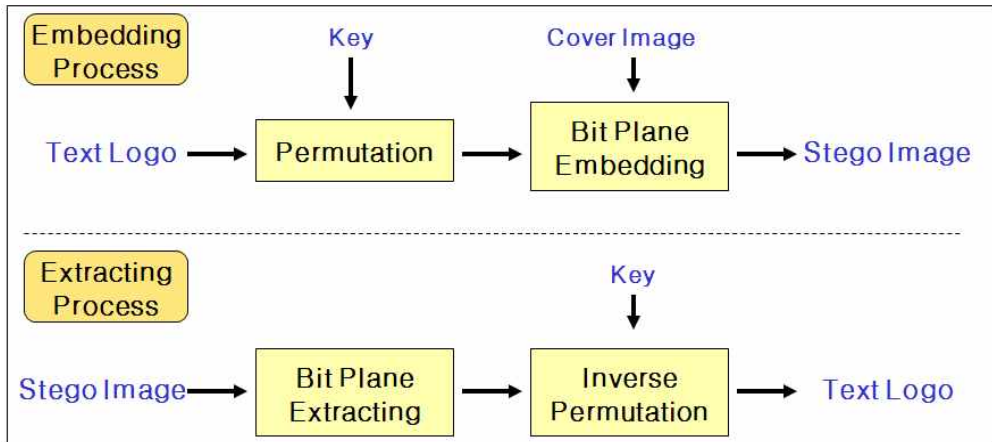
(그림 나1-1 비트 플레인 삽입방법)

비트플레인 삽입 방법을 사용시 하위 비트플레인으로 갈수록 화질 영향력이 향상 되고, 상위 비트로 갈수록 위의 그림과 같이 비인지성이 감소합니다.

이렇게 때문에 비트 플레인 삽입 방법을 사용할시에는 0~4비트 플레인쪽으로 삽입 하는 것이 더욱 안전하다.

(2) Permutation 병합 방법

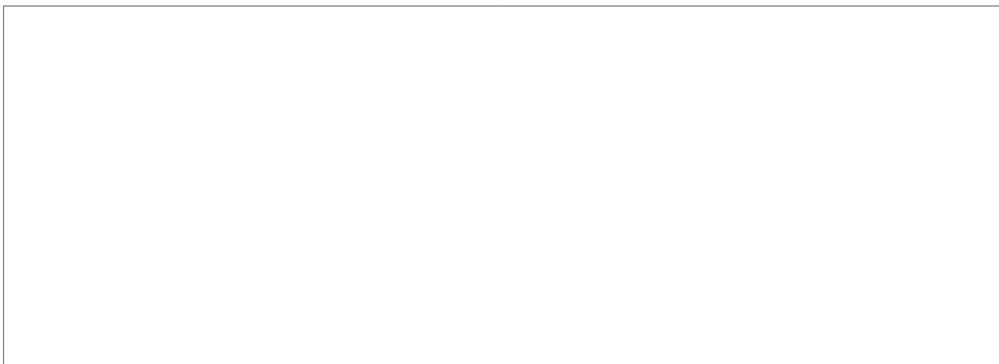
- 비밀정보를 우선 Permutation한 후 비트 플레인 삽입 방법을 추가하는 방식



(그림 나2-1 Permutation 병합 방법 원리)

Permutation 방법은 기존 비트플레인 삽입 방법을 사용하면, 비트플레인 추출 방법을 이용하면 로고 이미지가 보이는 문제점이 발생한다.

이 문제점을 해결하기 위하여 Permutation을 한 다음 cover 이미지에 logo 이미지를 삽입하는 방법으로 암호화 방법을 사용하게 되었다.



(그림 나2-2 Logo 이미지 Permutation 실행화면)

Permutation 방법은 중부대학교라는 로고 이미지를 Permutation 을 이용하여 좌우, 또는 상하 좌우로 섞어 버리는 것으로 위의 그림처럼 흑백TV 처럼 값이

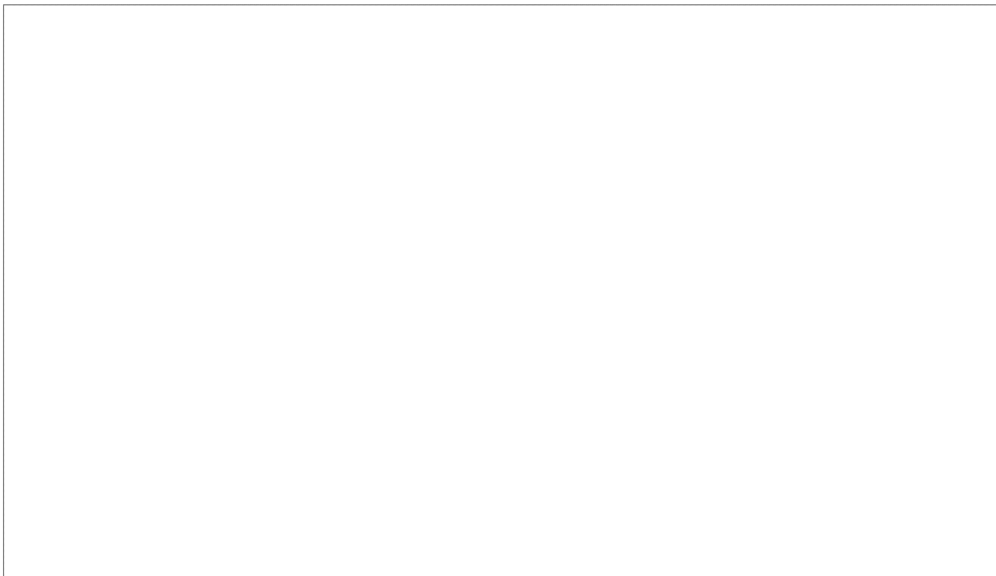
나오게 된다.



stgn-src-lena.raw 파일

txtlogo-flag-a.raw 파일

(그림 나2-3 Permutation 에 쓰일 cover image, logo image)



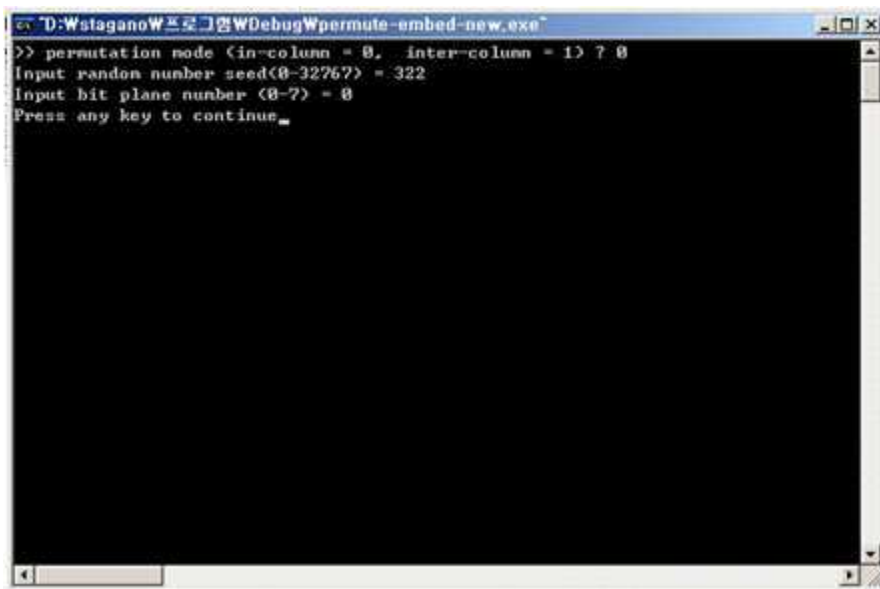
(그림 나2-4 Permutation 작업후 비트플레인 삽입 결과 화면)

Permutation을 추가하여 비트플레인 삽입을 하였을 경우 비트플레인 추출방법을 사용하여 특정 비트 값을 뽑아 낸다하여도 위의 그림처럼 비트플레인 방법은 7비트 플레인에서 Logo 이미지 값이 나와 버리지만, Permutation을 사용하게 되면 밑의 그림처럼 흑백 TV 처럼 보이는 듯한 효과가 발생하여 육안으로 보게된다면 이것이 무슨 내용인지 알아볼 수 없게 되어버린다.

하지만, Permutation 작업을 하게된다면, 비트플레인 추출 방법을 사용하여 추출하여도 알아볼수 없게 좌우, 상하로 섞어 놓지만, Permutation 작업을 하여 비트플레인 추출 방법을 상용하면 추출한 비트값이 흑백 티비처럼 사진에 무엇인가 있다는것 정도는 알아 볼수 있게 된다.

물론 그 내용이 무엇인지 알아볼수 없을 정도는 되지만, 이렇게 된다면 현재 이 사진에 무엇인가 있다는것을 알수 있기 때문에 보안상의 취약점이 남아 있게 된다.

Permutation 프로그램 실행 화면



(그림 나2-5 Permutation 프로그램 실행화면)

(**Permute-embed-new.exe**) ,(Permute-extract-new.exe)

permutation 방식을 이용하여 로고이미지와 커버 이미지를 합성 시키는 프로그램으로, permutation mode, random number seed 번호 설정, bit plane number 번호 설정을 할수있다.

(**Permute-embed-new.exe**) ,(Permute-extract-new.exe) 설명

Permutation mode :

in-column, inter-column = 로고 이미지를 좌우로 섞을것인가, 상하 좌우로 섞을

것인가 선택 모드

Input random number seed (0-32767) :

암호화를 위하여 0~32767 사이의 seed 번호를 지정 복호화 시에 이용한다.

Input bit plane number (0-7) :

비트 플레인을 0~7중 어떤것으로 할것인지 지정하는모드

※ 위의 모드는 Embed 와 Extract 프로그램 모두 동일합니다.



〈원본 이미지〉

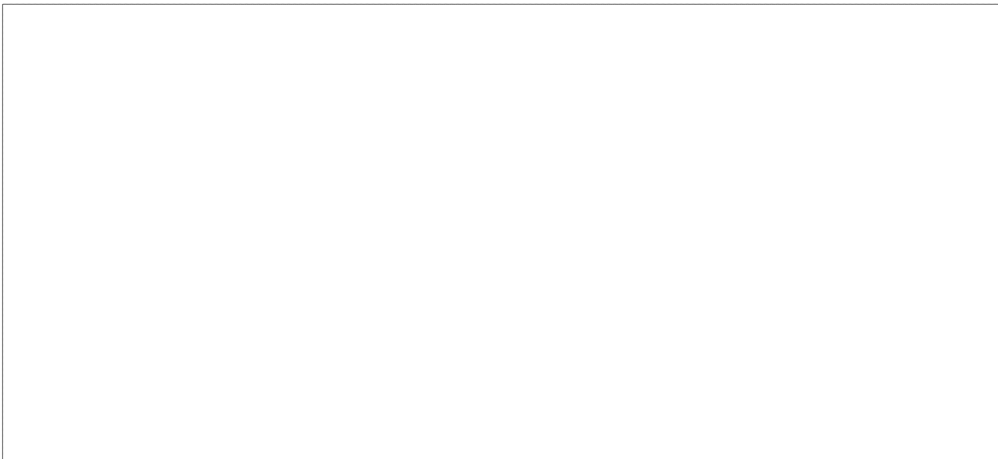
영 부 대 학 교

〈로고 이미지〉



〈Permutation 이미지〉

(그림 나2-6 Permutation 실행 결과)



(그림 나2-7 Permutation logo 파일 적용 결과)

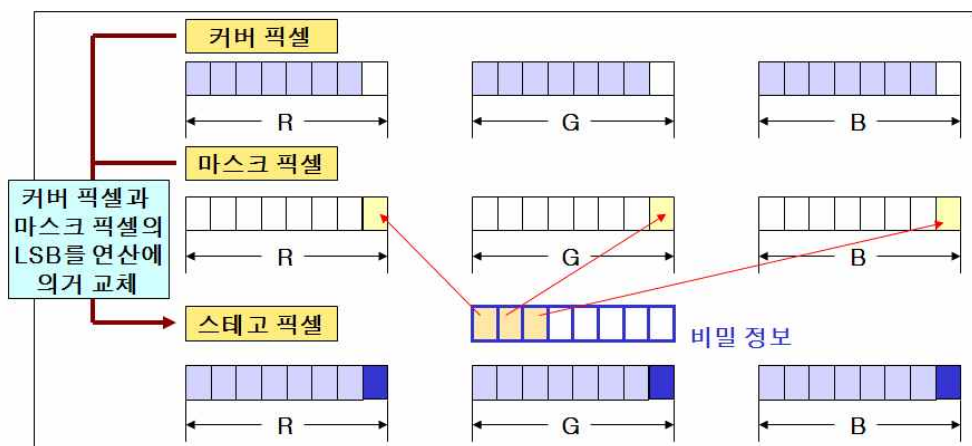
(3) 비트 플레인 분산삽입 방식

비트 플레인 방식이란 현재 LSB방식은 예를 들어첫번째 1을 0 비트플레인에, 두번째 0도 0 비트플레인에, 세번째 1도 0 비트플레인에 삽입하는 방식으로 모든 로고이미지를 0 비트플레인에 삽입 하는 방식이므로 0비트 플레인만 뽑아 낸다면 로고 이미지가 나타나 버리는 단점을 갖고 있다.

비트 플레인 분산방식은 기존에 비트 플레인 방식에서는 바이트 단위로 로고 이미지를 삽입하는 방식을 택했으나, 이 방법에서는 1바이트로 전송이 아니라 1비트 단위로 전송을 하게 되어 더욱더 세밀하게 이미지 작업을 하게 된다. 방식으로는 예를 들어 10111010 이라는 8비트 (1byte)를 비트 단위로 각각 비트 플레인 삽입 방식을 하는데, 이때 기존 비트 플레인 방식처럼 한곳에 몰아 놓는것이 아니라, 랜덤하게 전송하게 된다.

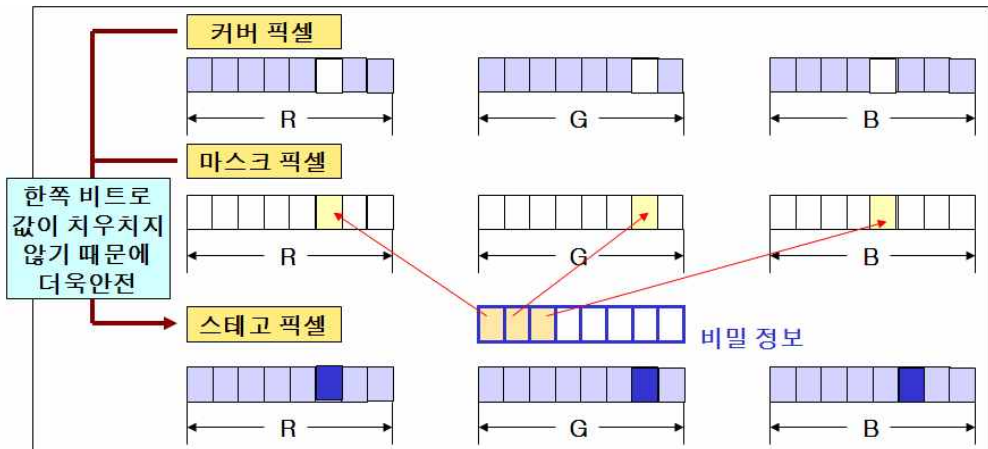
이렇게 해서 작업을 하게 된다면, 기존 비트 플레인 방식은 0비트 플레인을 뽑아 낸다면 이미지가 나와버리지만, 이 방법은 랜덤하게 들어가기 때문에 한쪽 비트만 뽑아도 제대로된 값이 나오지 않게 되어 더욱더 보완적으로 강화가 된다.

기존의 LSB 방식은 (그림 3-1 LSB 방식의 비트 플레인 삽입방식) 과 같이 이루어진다.



(그림 나3-1 LSB 방식의 비트 플레인 삽입방식)

하지만, 비트 플레인 분산방식은 특정 비트로만 같이 보내는것이 아니기 때문에 (그림 3-2 비트플레인 분산 삽입 방식) 처럼 이루어진다.



(그림 나3-2 비트플레인 분산 삽입 방식 이해)

(4) Pack 형식

Pack 이란 : pack 형식은 현재 나와있는 알집, Winrar 등과 같이 파일을 효율적으로 압축 하기 위한 프로그램 으로 졸업작품 연구용으로 제작된 것입니다.

비트플레인 방식을 하게 된다면 장점도 있지만, 단점이 이미지가 늘어나기 때문에 용량이 늘어 난다는 것이다.

1byte 단위로 작업하던것을 1bit 단위로 작업을 하게 되기때문에 가로 세로 8배 크기로 늘어나기 때문에 용량 면에서는 효율이 떨어지게 되는 단점이 발생하게 되는것이다.

그리하여 이 단점을 보완하기 위한것이 Pack 형식(압축)이다. 이미지상에서 불필요하게 연속 되는 공간의 이미지를 모아서 압축을 하는 방식이다.

```

C:\Windows\system32\cmd.exe
>> source file-name : logo-text.raw

>> code frequency calculation result
0 10851 11 1387 22 848 33 974
44 5759 55 891 66 928 77 748
88 6927 99 1040 aa 917 bb 5622
cc 1388 dd 1110 ee 1588 ff 201088

.. number of codes : 16 total frequencies : 292144

>> code shifting [original code -> new code]
ff --> 0 freq. : 203888 0 --> 1 freq. : 20851
88 --> 2 freq. : 6927 44 --> 3 freq. : 5759
bb --> 4 freq. : 5622 ee --> 5 freq. : 1588
11 --> 6 freq. : 1387 cc --> 7 freq. : 1388
dd --> 8 freq. : 1110 99 --> 9 freq. : 1040
33 --> a freq. : 974 66 --> b freq. : 928
aa --> c freq. : 917 55 --> d freq. : 891
22 --> e freq. : 848 77 --> f freq. : 748

.. total number of generated codes : 97325

>> frequencies of pixel's length:
1 20325 2 5884 3 2692 4 1308 5 1235
6 577 7 369 8 256 9 244 10 219
11 384 12 517 13 243 14 240 15 159
16 101 17 111 18 107 19 297 20 368
21 183 22 122 23 66 24 29 25 34
26 27 27 54 28 47 29 55 30 55
31 38 32 25 33 6 34 2 35 32
36 36 37 19 38 14 39 6 40 7
41 4 42 7 43 9 44 35 45 27
46 7 47 6 48 7 49 1 51 10
52 20 53 6 54 17 55 3 56 4
57 7 58 2 59 2 60 11 61 9
62 14 63 3 64 1 65 7 66 11
67 1 68 2 69 8 70 13 71 0
72 11 73 1 74 7 75 11 76 6
77 6 78 13 79 1 81 4 83 3
84 5 85 1 86 8 87 3 88 6
89 3 90 6 91 13 92 3 93 2
95 2 96 1 97 6 98 3 99 3
100 5 101 1 102 1 103 1 105 1
106 1 107 3 108 6 110 3 114 1
116 2 117 4 118 4 119 3 120 1
121 2 125 4 126 3 127 1 132 1
133 1 136 1 139 1 143 1 157 1
159 1 160 2 162 3 163 3 165 3
166 1 167 2 168 2 170 1 171 2
172 4 173 1 174 1 179 1 196 1
222 1 229 1 241 1 243 1 253 3
258 392

.. last iv = 639 ih = 279 real page = 1
.. logo file : total records 512, code records 638 (124,6%)

>> select the pages of logo file (1-9) : ?

```

```

>> select the pages of logo file (1-9) : ? 2
.. pages : 2 records/page : 320

>> created file-name 1 : logo-text-new1.raw
.. step : pos = 1 iv = 320 ih = 0

>> created file-name 2 : logo-text-new2.raw
2222.. step : pos = 2 iv = 318 ih = 296

>> process is terminated successfully !
계속하려면 아무 키나 누르십시오 . . .

```

(그림 나4-1 Pack 형식 프로그램 실행 화면)

(logo-pack.exe)

Source file-name을 지정하여, 현재 로고이미지의 빈도 조사를 하고, code shifting 을 통해서 16진수변환. 그리고, 연속적으로 사용되는 픽셀의 값을 찾는다. 이렇게 해서 압축을 하기 위한 총 빈도를 조사한 후, %값 나온 결과를 토대로 몇 장으로 만들 것인지를 정한다.

1) logo-pack.exe 설명

Source file-name : logot-text.raw

= 소스 파일로 Logo-text.raw 파일을 사용한다.

Code frequency calculation result

= 색이 사용된 빈도를 조사한다.

Number of codes : 16 total frequencies : 262144

= 사용된 코드는 16가지이며, 총 빈도수는 262144 이다 라는것을 나타냄.

Code shifting [original code -> new code]

= 코드를 16진수 형식으로 변환 시킨다.

Total number of generated code : 37325

= 변환된 총 코드는 37325 개이다.

Frequencies of pixel's length

= 연속적으로 사용된 빈도값을 조사한다.

Select the pages of logo file (1-9) : ? 2

= pack을 통하여 압축후 나온 % 값에 따라 장수를 정한다.

Created file-name 1 : logo-text-new1.raw

Created file-name 2 : logo-text-new2.raw

= 2장으로 정했을 경우 생성된 파일들의 이름

```

c:\D:\WstaganoW프로그램WDebugWlogo-unpack.exe

>> logo file reconstruction process

>> source file-name 1 : /stagano/RAW/logo-text-new1.raw
.. step iv = 319 ih = 512

>> code de-shifting [current code -> original code]
0 -> ff      1 -> 0      2 -> 88      3 -> 44
4 -> bb      5 -> ee      6 -> 11      7 -> cc
8 -> dd      9 -> 99      a -> 33      b -> 66
c -> aa      d -> 55      e -> 22      f -> 77

>> source file-name 2 : /stagano/RAW/logo-text-new2.raw
.. step iv = 318 ih = 296

>> codes are regenerated completely !

>> created file-name : /stagano/RAW/logo-text-rcv.raw

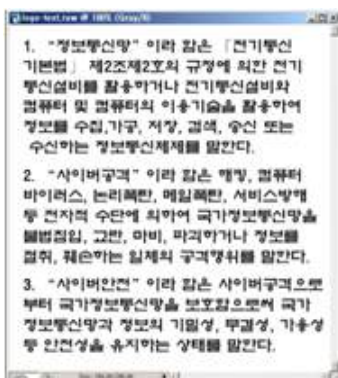
>> processe is terminated successfully !
Press any key to continue_

```

(그림 나4-2 logo-unpack.exe 실행화면)

(logo-unpack.exe)

Pack 을 통하여 압축된 파일logo-text- new1.raw , new2.raw 파일을 다시 불러 들여 코드를 오리지널 코드로 변경후, logo-text-rcv.raw 파일로 복호화 한다



< 텍스트 이미지 >



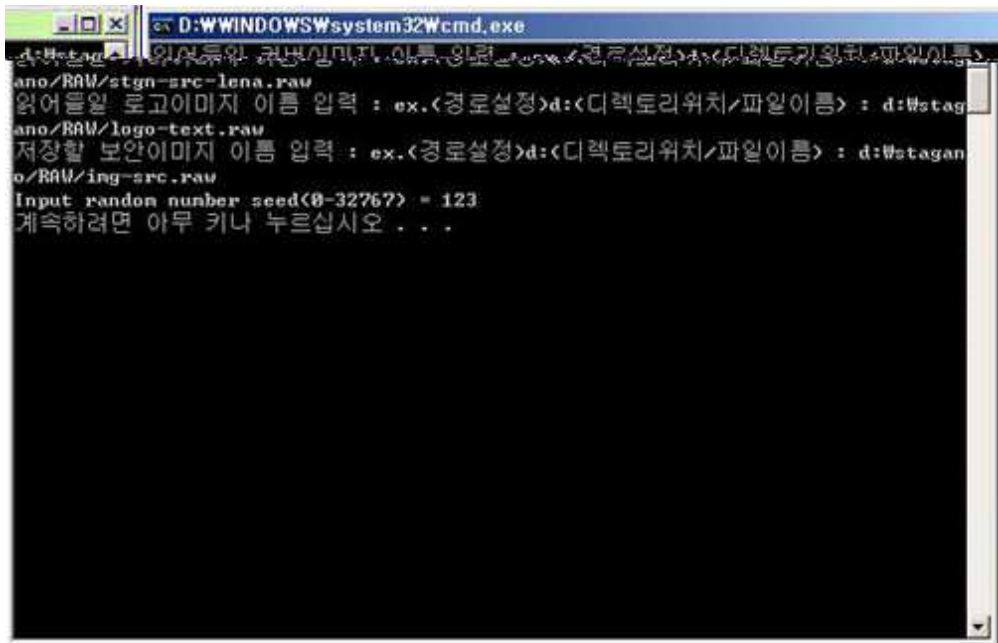
< 압축 이미지 >

(그림 나4-3 Pack 형식 logo 파일 실행 전, 후 결과)

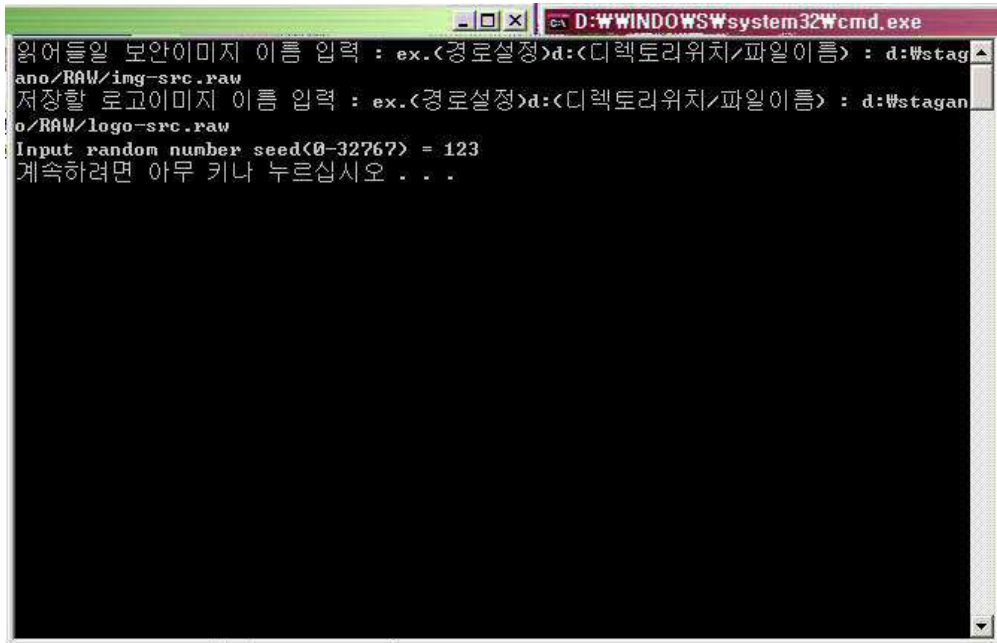
이렇듯, 비트 플레인 분산방식은 암호화하기에 좋으나, 8배로 크기가 늘어난다는 단점이 있었는데, 이런 Pack 이라는 압축 방식을 사용하여 그 단점을 해결되었다. 위의 (그림 4-3 pack 형식 logo 파일 실행 전, 후 결과)를 보는것과 같이 왼쪽에 글씨가 많이 써있어서 압축 효율이 100%가 넘어가기 때문에 두장으로 나누어 넣으면 오른쪽 그림처럼 나오게 된다. 저런 그림이 2장이 되는 것이다.

이렇듯, 압축이라는 암호화를 다시 한번 하기 때문에 안전성에서 더욱더 효율적이게 된다.

(5) 비트 플레인 분산방식 embedding



(그림 나5-1 비트 플레인분산 방식 embedding bit-emd.exe 실행화면)



```
D:\WINDOWS\system32\cmd.exe
읽어들일 보안이미지 이름 입력 : ex.<경로설정>d:<디렉토리위치\파일이름> : d:\stagan
ano\RAW\img-src.raw
저장할 로고이미지 이름 입력 : ex.<경로설정>d:<디렉토리위치\파일이름> : d:\stagan
o\RAW\logo-src.raw
Input random number seed(0-32767) = 123
계속하려면 아무 키나 누르십시오 . . .
```

(그림 나5-2 비트 플레인분산 방식 embedding bit-ext.exe 실행화면)

(bit-emd.exe) (bit-ext.exe)

- bit-emd.exe -

Pack형식을 이용하여 암호화가 된 Logo 파일을 Cover 이미지와 Embedding 시켜 완성된 스테가노 이미지를 만든다.

- bit-ext.exe -

복호화시 사용되는 파일 내용은 bit-emd.exe와 동일하다.

4. 결론

비트 플레인 추출방법은 (0-7)비트 중 어느 한 해당하는 비트에 값을 넣기 때문에 비트 플레인 추출 방법을 사용하게 특정 비트에서 값이 보여져 버리는 문제점이 발생하기 때문에 다른 방법을 찾던중 수업시간에 하는 Permutation 작업을 조금 보완하여 하기로 했다.

Permutation 을 보완한 것은 Permutation 후 흩어져 있는 로고 이미지를 한쪽으로 보내는 쉬프트 방법으로 진행하였다.

하지만 비트 값을 뽑아내는 부분에서 어려움을 겪게 되어 다른 방법을 모색하던중, 연구 하게 된 방법은 비트플레인 분산삽입 방식이었다. 기존의 LSB 방식은 해당 비트에만 값을 넣기 때문에 비트플레인 추출 공격에 취약하다는 것을 알고 있기 때문에 이 비트 플레인 삽입방식을 업그레이드 하여 특정 비트에만 넣는것이 아니라, %별로 나누어 해당하는 값의 픽셀 값이 나오면 랜덤하게 나누어 넣어 보완하는 것이다.

이렇게 하면 한 부분의 비트 값만 추출한다하여도 내용은 알아볼 수 없게 되고, 분산되어있기 때문에 보안성을 강화하게 되는 것이다.

하지만 이것또한 문제가 있었으니, 그것은 1byte 단위로 작업 하던것을 1bit 단위로 바꾸면서 그림의 용량이 8배로 늘어난다는 것이었다.

이런 문제점을 보완하기 위하여 찾아낸 방법이 바로 pack 형식입니다. Pack 형식은 현재 윈도우 등에서 사용하고 있는 알집, WinRAR 같은 압축프로그램입니다. 로고 이미지가 8배로 늘어나면서 필요 이상으로 빈공간이 많이 생기게 됩니다.

pack 형식은 그 빈공간을 압축하여 그림의 용량 효율을 최대한으로 높이는 것입니다.

연속적으로 들어가는 값을 모아 한 개의 값으로 바꾸는 작업이 이루어 지기 때문에 Logo 이미지에 글씨가 많이 없다면 압축량은 엄청나게 늘어나 512x512 사이즈의 상단 부분정도 밖에 안되는 용량으로 만들수있다.

Pack형식 프로그램은 또한 100% 클 512x512 사이즈 한 장이라는 기준으로 잡아서 100% 가 넘으면 두장으로 만들 수 있도록 선택할수 있다.

이렇게 하면 공격자가 추출방법을 사용하여 값을 찾아 내려고 한다하여도 무슨

내용은지 전혀 알수가 없어 보완성을 한층더 높인것이다.

이 Pack형식을 이용한 로고 이미지를 가지고 다시 Embedding 하며 그때 시드 번호를 주기 때문에 암호화가 더 되기 때문에 보안 취약점을 한층 더 업그레이드 하여 강력한 보안성을 가진 스테가노그래피를 만들 수 있다.

이러한 방법을 이용한다면, 향후 DRM 기술에서도 쓰일수 있으며, 군사용이나, 중요전문을 평범한 그림들과 같이 보내어 쉽게 발각되지 않고 보안 내용을 전송할수 있게된다.

위터마킹 이나, 핑거프린팅 등에 쓰일수 있도록 연구를 한다면 향후 디지털 저작권 문제의 해결방안으로도 연구가 가능하다.

부록

1. 프로그램 소스

1) Embedding

(1) embed

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define NV 512
#define NH 512
unsigned char src[NV][NH]; // Cover Image
unsigned char dst[NV][NH]; // Stegano Image
unsigned char logo[NV][NH]; // Secret Logo
unsigned char bpl, bit;
int i,j;
FILE*fp, *fpo;

main()
{

// cover image file(lena image) read
if((fp=fopen("/cygwin/home/watermark1/stgn-src-lena.raw","rb"))==NULL){
printf("\n>> File Read Error !\n"); exit(0);
}
fread(src, NV, NH, fp);
fclose(fp);

// secret logo file (중부대학교) read
if((fp=fopen("/cygwin/home/watermark1/txtlogo-flag-a.raw","rb"))==NULL){
printf("\n>> File Read Error !\n"); exit(0);
}
fread(logo, NV, NH, fp);
fclose(fp);

// bit plane embedding

// bit plane selection
printf("Input bit plane number (0-7) = ");
```

```

scanf("%d", &bit);
bpl=(unsigned char)pow(2,bit);
for(i=0; i<NV; i++)
for(j=0; j<NH; j++) {
if(logo[i][j]>=128) dst[i][j]=src[i][j]|bpl;
else dst[i][j]=src[i][j]&(~bpl);
}
// embedded image (stegano image) write
if((fpo=fopen("/cygwin/home/watermark1/stgn-dst-flag-a0.raw","wb"))==NULL){
printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(dst, NV, NH, fpo);
fclose(fpo);
}

```

(2) extract

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define NV 512
#define NH 512
unsigned char stgn[NV][NH]; // Stegano Image
unsigned char logo[NV][NH]; // Secret Logo
unsigned char bpl, bit;
int i, j;
FILE *fp, *fpo;

main()
{
// stegano image read
if((fp=fopen("/cygwin/home/watermark1/stgn-dst-flag-a0.raw","rb"))==NULL){
printf("Wn>> File Read Error !Wn"); exit(0);
}
fread(stgn, NV, NH, fp);
fclose(fp);
// bit plane extracting

```

```

// bit plane selection
printf("Input bit plane number (0-7) = ");
scanf("%d", &bit);
for(i=0; i<NV; i++)
for(j=0; j<NH; j++){
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;

printf("\n stgn : ");
for (i=0; i<2;i++) {
for(j=0;j<10;j++)
{printf(" %2x",stgn[i][j]);}
}
printf("\n extr : ");
for (i=0; i<2;i++) {
for(j=0;j<10;j++)
{printf(" %2x",logo[i][j]);}
}
printf("\n");
// =====

// extracted image (중부대학교) write
.... if((fpo=fopen("/cygwin/home/watermark1/extr-txtlogo-flag-a0.raw","wb"))==NULL){
printf("\n>> File Write Error !\n"); exit(0);
}
fwrite(logo, NV, NH, fpo);
fclose(fpo);
}

```

2) Permutation

(1) permute-embed

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define NV 512
#define NH 512

```

```

unsigned char src[NV][NH]; // Cover Image
unsigned char dst[NV][NH]; // Stegano Image
unsigned char logo[NV][NH], permuted_logo[NV][NH]; // Secret Logo
int key[NV][NH]; // Permutation Key
FILE *fp, *fpo;

```

```

void key_generation(int flag)
{
    int i, j, k, seed, flg[NH];
    // key-in the starting value of random number
    printf("Input random number seed(0-32767) = ");
    scanf("%d", &seed);
    srand(seed);
    for(i=0; i<NV; i++){
        key[i][0]=i; flg[0]=1; // initialize
        for (j=1; j<NH; j++){
            flg[j]=0;
            for(j=1; j<NH; j++){
                while(1){
                    k=(int)(rand()/(float)RAND_MAX*NH-0.5);
                    if (k>511) printf(">>> %3d ",k);
                    if (flg[k]==1) continue;
                    key[i][j]=k;
                    flg[k]=1;
                    break;
                }
            }
        }
        if (flag==0) for (i=0; i<NV; i++) key[0][i]=i;
    }
}

```

```

main()
{
    int i, j, k, seed2, permuted_mode;

```

```

unsigned char b, bit;
char imagename[100];
char logoname[100];
char writelogo[100];
char writeimge[100];
printf("읽어들일 커버이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(imagename);
printf("읽어들일 로고이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(logoname);
printf("저장할 로고이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(writelogo);
printf("저장할 보안이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(writeimge);

// cover image read
if((fp=fopen(imagename,"rb"))==NULL) {
printf("₩n>> File Read Error !₩n"); exit(0);
}
fread(src, NV, NH, fp);
fclose(fp);

// secret logo image (중부대학교) read
if((fp=fopen(logoname,"rb"))==NULL) {
printf("₩n>> File Read Error !₩n"); exit(0);
}
fread(logo, NV, NH, fp);
fclose(fp);

printf(">> permutation mode (in-column = 0, inter-column = 1) ? ");
scanf("%d", &permuted_mode);
key_generation(permuted_mode); // call key generation module

```

```

// logo image permutation
for (j=0; j<NH; j++) permuted_logo[0][j]=logo[0][key[0][j]];
for (i=1; i<NV; i++){
k=key[0][i];
for (j=0; j<NH; j++)
permuted_logo[i][j]=logo[k][key[k][j]];

fflush(stdin);
// permuted logo image write
if((fpo=fopen(writelogo,"wb"))==NULL) {
printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(permuted_logo, NV, NH, fpo);
fclose(fpo);

// embedding secret logo image
printf("Input random number seed(0-32767) = ");
scanf("%d", &seed2);
srand((unsigned)seed2);
for(i=0; i<NV; i++)
for(j=0; j<NH; j++) {
bit=rand()%5;
b=(unsigned char)pow(2,bit);
if(permuted_logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
}

// permuted stego image write
if((fpo=fopen(writeimge,"wb"))==NULL) {
printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(dst, NV, NH, fpo);
fclose(fpo);
}

```

(2) permute-extract

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define NV 512
#define NH 512
unsigned char stgn[NV][NH]; // Stegano Image
unsigned char logo[NV][NH], permuted_logo[NV][NH]; // Secret Logo
int key[NV][NH]; // Permutation Key
FILE *fp, *fpo;

void key_generation(int flag)
{
    int i, j, k, seed, flg[NH];

    // key-in the starting value of random number
    printf("Input random number seed(0-32767) = ");
    scanf("%d", &seed);
    srand(seed);
    for(i=0; i<NV; i++){
        key[i][0]=i; flg[0]=1; // initialize
        for (j=1; j<NH; j++) flg[j]=0;
        for(j=1; j<NH; j++){
            while(1){
                · k=(int)(rand()/(float)RAND_MAX*NH-0.5); if (k>511) printf(">>> %3d ",k);
                if (flg[k]==1) continue;
                key[i][j]=k; flg[k]=1;break;
            }
        }
    }
    if (flag==0) for (i=0; i<NV; i++) key[0][i]=i;
}
```

```

main()
{
int i, j, k, permuted_mode;
unsigned char bit, bpl;
char secretnam[100];
char writelogo[100];
printf("읽어들일 보안이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(secretnam);
printf("저장할 로고이미지 이름 입력 : ex.(경로설정)d:₩(디렉토리위치/파일이름) : ");
gets(writelogo);

// stegano image file read
if((fp=fopen(secretnam,"rb"))==NULL) {
printf("₩n>> File Read Error !₩n"); exit(0);
}
fread(stgn, NV, NH, fp);
fclose(fp);

printf(">> permutation mode (in-column = 0, inter-column = 1) ? ");
scanf("%d", &permuted_mode);

key_generation(permuted_mode); // call key generation module

// logo image extracting
printf("Input bit plane number (0-7) = ");
scanf("%d", &bit);
for(i=0; i<NV; i++)
for(j=0; j<NH; j++) {

```

```

bpl=stgn[i][j]>>bit;//오른쪽으로 비트연산
    if(bpl%2==0) permuted_logo[i][j]=0;//만약 bpl을 2로 나눈값의 나머지가 0이
    라면 permuted_logo의 i열j행에 0값을 부여한다.
    else permuted_logo[i][j]=255;//아니라면 255를 부여한다.
}

```

```

// logo re permutation

```

```

    for (j=0; j<NH; j++) logo[0][key[0][j]]=permuted_logo[0][j]; //j값이
    511이 될때까지 2차원 배열logo의 0열 key의 0열j행값이 가리키는 행에
    for (i=1; i<NV; i++) { //permuted_logo의 0열 j행 값을 부여한다.
    k=key[0][i]; //i가 511이 될때까지 k에 key의 0열 i행의 값을 부여한다.
        for (j=0; j<NH; j++) //j가 511이 될때까지 2차원배열 logo의 k열 key의 k열 j
        행값이 가리키는 행에 permuted_logo의 i열 j행 값을 부여한다.
        logo[k][key[k][j]]=permuted_logo[i][j];
    }
}

```

```

// extracted secret image write
if((fpo=fopen(writelogo,"wb"))==NULL) {
    printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(logo, NV, NH, fpo);
fclose(fpo);
}

```

3) Logo-Pack, Logo-Unpack

(1)Logo-pack

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define NV 512

```

```

#define NH 512
#define FULL 256
#define NN 262144
#define ALL_ZERO 0x00
#define ALL_ONE 0xff
#define test 510

unsigned char logo[NV][NH]; // Secret Logo
unsigned char ccode;
char *srcfile_ptr, *dstfile_ptr;
char *pos_ptr, ch[]="0" "1" "2" "3" "4" "5" "6" "7" "8" "9";
char file_dir[10],file_name[30],dst_ext[]="-new",name_ext[]=".raw";
int i, j, iv, ivs, ih, ihs, lng, flg, interval, pos;
int mxpos, code[257], newcode[257], mxcode, mxfreq, codefreq[257];
int totcode, totcode_org, totfreq, freq[256], frq[513];
FILE *fpi, *fpo;

void file_read()
{
    srcfile_ptr=calloc(50, sizeof(char));
    strcpy((char *)srcfile_ptr, (char *)file_dir); .....
    strcat((char *)srcfile_ptr, (char *)file_name);
    strcat((char *)srcfile_ptr, (char *) name_ext);
    printf("\n>> source file-name : %s", srcfile_ptr);
    if((fpi=fopen(srcfile_ptr,"rb"))==NULL){
        printf("\n>> File Read Error !\n"); exit(0);
    }

    fread(logo, NV, NH, fpi);
    fclose(fpi);
}

void freqcalc()
{
    for(i=0; i<256; i++) freq[i]=0;
    for(i=0; i<NV; i++) for(j=0; j<NH; j++) freq[logo[i][j]]++;
}

int codeshift()
{lng=0; ccode=logo[iv][ih];

```

```

while (1) {
lng++; ih++;
if (ih>=NH) {
ih=0; iv++;
if (iv>=NV) return(0);
}
if (ccode!=logo[iv][ih]) return(1);
if (lng==FULL) return(1);
}
}

void file_write_check()
{pos++;
if (pos>9) {printf("Wn>> too many files are created !!Wn"); exit(0);}
dstfile_ptr=calloc(50, sizeof(char)); pos_ptr=&ch[pos];
strcpy((char *)dstfile_ptr, (char *)file_dir);
strcat((char *)dstfile_ptr, (char *)file_name);
strcat((char *)dstfile_ptr, (char *)dst_ext);
strncat((char *)dstfile_ptr, (char *)pos_ptr, 1);
strcat((char *)dstfile_ptr, (char *)name_ext);
printf("WnWn>> created file-name %1d : %s", pos, dstfile_ptr);
if((fpo=fopen(dstfile_ptr,"wb"))==NULL){
printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(logo, NV, NH, fpo); fclose(fpo);
}

void fill(int xcode)
{logo[iv][ih]=xcode;
ih++; if (ih!=NH) return;
ih=0; iv++; if (iv!=interval) return;
file_write_check();
printf("Wn.. step : pos = %1d   iv = %3d   ih = %3d", pos, iv, ih);
iv=0; ih=18; // ih=18 --> iv/ih saving area
for(i=0; i<NV; i++) {
for(j=0; j<NH; j++) {
if ((i==0) && (j<ih)) continue;
logo[i][j]=0x01;// logo image clear (pixel code 0x01)
}
}
}

```

```

}
}

```

```

void codetrans(int cnt, int pcode)
{int k, bpl[9];
for (k=cnt-1; k>=0; k--) bpl[k]=(int)pow(2,k);
for (k=cnt-1; k>=0; k--) {
if ((bpl[k]&pcode)==bpl[k]) fill(ALL_ONE); else fill(ALL_ZERO);
}
}

```

```

void prereform(int pcode, int lng)
{
lng++;
pcode=31-pcode;
if (pcode==31) goto pre_ff_coding;
if (lng==1) { totcode=totcode+2; goto pre_skp;}
totcode=totcode+2;
if (lng==2 || lng==3 || lng==4) totcode=totcode+2;
if (lng>=5) totcode=totcode+10;
pre_skp:
totcode=totcode+5; return;
pre_ff_coding: // oxff series coding
totcode=totcode+2;
if (lng==FULL) { totcode++; return;}
totcode=totcode+9; return;
}

```

```

void reform(int pcode, int lng)
{
lng++;// recovery of length
pcode=31-pcode;
if (pcode==31) goto ff_coding;// oxff pixel series coding
if (lng==1) { fill(ALL_ONE); fill(ALL_ZERO); goto skp;} // single pixel coding
fill(ALL_ZERO); fill(ALL_ONE);// not 0xff/complex pixel coding
if (lng==2) { fill(ALL_ONE); fill(ALL_ONE);}
if (lng==3) { fill(ALL_ONE); fill(ALL_ZERO);}
if (lng==4) { fill(ALL_ZERO); fill(ALL_ONE);}
if (lng>=5) { fill(ALL_ZERO); fill(ALL_ZERO); codetrans(8, -lng);}
}

```

```

    skip:codetrans(5, pcode); return;
ff_coding: // oxff series coding
fill(ALL_ONE); fill(ALL_ONE);
if (lng==FULL) { fill(ALL_ONE); return;}
fill(ALL_ZERO); codetrans(8, -lng); return;
}
void recalclng(int cd, int lng)
{lng++;
cd=31-cd; frq[lng]++;
}
void main()
{int k, count, ncount, page;
int gencode[50000];

printf("파일이 저장되어있는 디렉토리 입력 ex.(경로설정)d:W(디렉토리)/ : ");
gets(file_dir);
fflush(stdin);
printf("파일이름 입력 : ex.(textlogo) : ");
gets(file_name);
file_read();// source file read
freqcalc();// code frequency calculation
totcode=0; totfreq=0; count=0;
printf("\n\n>> code frequency calculation result\n");
for (k=0; k<256;k++)// code frequency calculation process
if (freq[k]!=0) {
code[count]=k; codefreq[count]=freq[k];
totcode++; totfreq=totfreq+freq[k];
printf("%3x %7d", code[count],codefreq[count]);
count++; if ((count%4)==0) printf("\n");
}
printf("\n.. number of codes :%3d total frequencies : %10d", count, totfreq);
pos=0;
// code shifting procedure
// sorting by code frequencies' order
lp:mxpos=pos; mxcode=code[pos]; mxfreq=codefreq[pos];
for (k=pos+1; k<count; k++) {
if ((mxfreq<codefreq[k]) || (mxfreq==codefreq[k] && mxcode>code[k])) {
mxpos=k; mxcode=code[k]; mxfreq=codefreq[k];
}
}

```

```

}
if (pos!=mxpos) {
code[mxpos]=code[pos]; codefreq[mxpos]=codefreq[pos];
code[pos]=mxcode; codefreq[pos]=mxfreq;
}
pos++; if (pos<(count-1)) goto lp;
printf("\n\n>> code shifting [original code -> new code]"); // code shifting by
high frequencies' code
for (k=0; k<count; k++) {
if ((k%2)==0) printf("\n");
printf(" %2x --> %2x   freq. : %7d      ", code[k], k, codefreq[k]);
}
//preprocess of code regeneration
for (k=0; k<256;k++) newcode[k]=ALL_ONE;
for (k=0; k<count; k++) newcode[code[k]]=k; // new code generation
count--; // total number of non-zero frequency code
iv=0; ih=0; lng=0; ccode=0; flg=1; ncount=0;
while (flg) {
flg=codeshift(); gencode[++ncount]=(lng-1)*256+newcode[ccode]; // lng : length of
code - 1
}
printf("\n.. total number of generated codes : %5d\n", ncount);
//calculation of length of same pixel code's series
iv=0; ih=0; for (k=1; k<=512; k++) frq[k]=0;
for (k=1; k<=ncount; k++) {
mxcode=gencode[k]; lng=(mxcode & 0xff00)/256; mxcode=mxcode & 0xff;
recalclng(mxcode, lng);
}
printf("\n>> frequencies of pixel's length\n");
for (k=1; k<=512; k++) {
if (frq[k]!=0) printf(" %3d %6d      ", k, frq[k]);
}
//calculation of logo file size when codes are regenerated
totcode=0; // clear totcode for prereform routine
for (k=1; k<=ncount; k++) {
mxcode=gencode[k]; lng=(mxcode & 0xff00)/256; mxcode=mxcode & 0xff;
prereform(mxcode, lng);
}
// code regeneration procedure

```

```

page=1; totcode_org=totcode;
totcode=totcode_org+(count+1)*13+page*18+5;// each code 13, iv/ih 9*2 each
page, code count 5, inverse 2
ivs=totcode/NV; ihs=totcode%NH;
printf("Wn.. last iv = %3d ih = %3d real page = %1d", ivs, ihs, page);
printf("Wn.. logo file : total records 512, code records %5d (%4.1f% %%)Wn", ivs,
(ivs*100.)/512.);
printf("Wn>> select the pages of logo file (1-9) : ? ");
scanf("%d",&page);
interval=(ivs+1)/page; if (((ivs+1)-page*interval)>0) interval++;
printf(".. pages : %1d records/page : %3d", page, interval);
iv=0; ih=0; flg=0; pos=0;

for(i=0; i<NV; i++) for(j=0; j<NH; j++) logo[i][j]=0x01; // logo image clear (pixel
code 0x01)
codetrans(9, interval); codetrans(9, 0); // iv(interval) & ih reserve (for full size)
codetrans(5, count);// save the number of original code
for (k=0; k<=count; k++) {
codetrans(5, k); codetrans(8, code[k]); }// save the original/new code pairs
for (k=1; k<=ncount; k++) {
mxcode=gencode[k]; lng=(mxcode & 0xff00)/256; mxcode=mxcode & 0xff;
reform(mxcode, lng);
}
if (~(iv==0) && (ih==18))) {
ivs=iv; ihs=ih; iv=0; ih=0; codetrans(9, ivs); codetrans(9,ihs); file_write_check();
printf("Wn2222.. step : pos = %1d iv = %3d ih = %3d", pos, ivs, ihs);
}
printf("WnWn>> process is terminated successfully !Wn");
}

```

(2)Logo-Unpack

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#define NV 512
#define NH 512
#define FULL 256

```

```

#define NN 262144
#define ALL_ZERO 0x00
#define ALL_ONE 0xff
#define LAST_EOF 32767
#define test 510
unsigned char logo[NV][NH]; // Secret Logo
unsigned char rcv_logo[NV][NH]; // Recovery of Secret Logo
unsigned char ccode;
int mxpos, code[257], newcode[257], mxcode, mxfreq, codefreq[257];

char *srcfile_ptr, *dstfile_ptr;
char *pos_ptr, ch[]="0" "1" "2" "3" "4" "5" "6" "7" "8" "9";
char file_dir[30],
    file_name[30], // source file-name
    src_ext[]="-new", dst_ext[]="-rcv", name_ext[]=".raw";
int account;

int i, j, iv, ivs, rcv_iv, ih, ihs, rcv_ih, lng, flg, interval, pos;
int totcode, totcode_org, totfreq, freq[256], frq[513];
FILE *fpi, *fpo;

void file_read()
{pos++;
srcfile_ptr=calloc(50, sizeof(char)); pos_ptr=&ch[pos];
strcpy((char *)srcfile_ptr, (char *)file_dir);
strcat((char *)srcfile_ptr, (char *)file_name);
strcat((char *)srcfile_ptr, (char *)src_ext);
strncat((char *)srcfile_ptr, (char *)pos_ptr, 1);
strcat((char *)srcfile_ptr, (char *) name_ext);
printf("\n\n>> source file-name %1d : %s", pos, srcfile_ptr);
if((fpi=fopen(srcfile_ptr,"rb"))==NULL){
printf("\n\n>> File Read Error !\n"); exit(0);
}

    fread(logo, NV, NH, fpi); fclose(fpi); iv=0; ih=0;
}

void file_write()
{dstfile_ptr=calloc(50, sizeof(char));
strcpy((char *)dstfile_ptr, (char *)file_dir);

```

```

strcat((char *)dstfile_ptr, (char *)file_name);
strcat((char *)dstfile_ptr, (char *)dst_ext);
strcat((char *)dstfile_ptr, (char *)name_ext);
if((fpo=fopen(dstfile_ptr,"wb"))==NULL){
printf("\n>> File Write Error !\n"); exit(0);
}
printf("\n>> created file-name : %s", dstfile_ptr);
fwrite(rcv_logo, NV, NH, fpo); fclose(fpo);
}

void rcv_fill(int cnt, int xcode)
{int k;
for (k=1; k<=cnt; k++) {
acount++;
rcv_logo[rcv_iv][rcv_ih]=xcode;
rcv_ih++;
if (rcv_ih<NH) continue;
rcv_ih=0; rcv_iv++; if (rcv_iv<NV) continue;
flg=LAST_EOF;
printf("\n\n>> codes are regenerated completely !\n");
file_write(); rcv_iv=0; rcv_ih=0;
}
}

int coderetrans(int cnt)
{int k, pcode;
pcode=0;
for (k=cnt; k>=1; k--) {
pcode=pcode*2; if (logo[iv][ih]==ALL_ONE) pcode++;
ih++;
if ((iv<ivs) && (ih<NH)) continue;
if ((iv==ivs) && (ih<ihs)) continue;
ih=0; iv++;
if (iv<=ivs) continue;
if ((iv>ivs) && (ihs!=NH)) continue;
file_read();
ivs=coderetrans(9); ihs=coderetrans(9);
if ((logo[ivs][0]==1) && (ihs==0)) {ivs--; ihs=NH;}
printf("\n.. step iv = %3d ih = %3d", ivs, ihs);
}
}

```

```

return(pcode);
}

void main()
{int k, count, cnt, rcv_code;
printf("파일이 저장되어있는 디렉토리 입력 ex.(경로설정)d:W(디렉토리) :");
gets(file_dir);
fflush(stdin);

printf("source file-name(원본파일이름) 입력 : ex.(textlogo) : ");
gets(file_name);
printf("\n>> logo file reconstruction process");
account=0;
flg=0; pos=0; rcv_iv=0; rcv_ih=0;
ivs=Nv; ihs=NH;// ivs, ihs presetting
file_read();
ivs=coderetrans(9); ihs=coderetrans(9);
if ((logo[ivs][0]==1) && (ihs==0)) {ivs--; ihs=NH;}
count=coderetrans(5);
printf("\n.. step iv = %3d ih = %3d", ivs, ihs);

printf("\n\n>> code de-shifting [current code -> original code]"); // code shifting
for (k=0; k<=count; k++) {
mxcode=coderetrans(5); code[mxcode]=coderetrans(8);
if ((k%4)==0) printf("\n");
printf(" %2x --> %2x ", mxcode, code[mxcode]);
}
for(i=0; i<NV; i++) for(j=0; j<NH; j++) rcv_logo[i][j]=0; // rcv_logo image
clear(black)

while (flg!=LAST_EOF) {
mxcode=coderetrans(2);
switch (mxcode) {
case 1:cnt=coderetrans(2);
if (cnt==0) cnt=256-coderetrans(8); else cnt=5-cnt;
rcv_code=code[31-coderetrans(5)];
rcv_fill(cnt, rcv_code);
break;
case 2:rcv_code=code[31-coderetrans(5)]; cnt=1;

```

```

rcv_fill(cnt, rcv_code);
break;
case 3:cnt=coderetrans(1); rcv_code=code[0];
if (cnt!=1) cnt=256-coderetrans(8); else cnt=FULL;
rcv_fill(cnt, rcv_code);
break;
default:printf("Wn>> process is cancelled abnormallyWn"); exit(0);
}
}
if ((rcv_iv!=0) || (rcv_ih!=0)) { printf("Wn>> process is terminated abnormallyWn");
exit(0);}
printf("WnWn>> processe is terminated successfully !Wn");
}

```

4) 비트플레인 embedding

(1) bit-embed

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define NV 512
#define NH 512
unsigned char src[NV][NH]; // Cover Image
unsigned char dst[NV][NH]; // Stegano Image
unsigned char logo[NV][NH]; // Secret Logo

int i, j, seed2;
FILE *fp, *fpo;

void main()
{
int count0=0,count1=0,count2=0,count3=0,count4=0;
unsigned char b, bit;
char imagename[100];
char logoname[100];
char writeimg[100];

printf("읽어들일 커버이미지 이름 입력 : ex.(경로설정)d:W(디렉토리위치/파일이름) : ");
gets(imagename);

```

```

printf("읽어들일 로고이미지 이름 입력 : ex.(경로설정)d:W(디렉토리위치/파일이름) : ");
gets(logoname);
printf("저장할 보안이미지 이름 입력 : ex.(경로설정)d:W(디렉토리위치/파일이름) : ");
gets(writeimage);
// cover image read
if((fp=fopen(imagename,"rb"))==NULL) {
printf("Wn>> File Read Error !Wn"); exit(0);
}
fread(src, NV, NH, fp);
fclose(fp);
// secret logo image (중부대학교) read
if((fp=fopen(logoname,"rb"))==NULL) {
printf("Wn>> File Read Error !Wn"); exit(0);
}
fread(logo, NV, NH, fp);
fclose(fp);
// bit plane embedding
// bit plane selection
// embedding secret logo image
printf("Input random number seed(0-32767) = ");
scanf("%d", &seed2);
srand(seed2);
for(i=0; i<NV; i++)
{
for(j=0; j<NH; j++)
{
bit=(unsigned char)rand()%5;
ch:
bit=(unsigned char)rand()%5;
if(bit==0 && count0<=131072)
{
b=(unsigned char)pow(2,bit);
if(logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
count0++;
}
else if(bit==1 && count1<=52429)
{
b=(unsigned char)pow(2,bit);

```

```

if(logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
count1++;
}
else if(bit==2 && count2<=39322)
{
b=(unsigned char)pow(2,bit);
if(logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
count2++;
}
else if(bit==3 && count3<=26214)
{
b=(unsigned char)pow(2,bit);
if(logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
count3++;
}
else if(bit==4 && count4<=13107)
{
b=(unsigned char)pow(2,bit);
if(logo[i][j]>=128) dst[i][j]=src[i][j]|b;
else dst[i][j]=src[i][j]&(~b);
count4++;
}
if(bit==0 &&count0>131072)
goto ch;
if(bit==1 && count1>52429)
goto ch;
if(bit==2 && count2>39322)
goto ch;
if(bit==3 && count3>26214)
goto ch;
if(bit==4 && count4>13107)
goto ch;
}
}
// embedded image (stegano image) write
// permuted stego image write

```

```

if((fpo=fopen(writeimge,"wb"))==NULL) {
printf("Wn>> File Write Error !Wn"); exit(0);
}
fwrite(dst, NV, NH, fpo);
fclose(fpo);
}

```

(2) bit-extract

```

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#define NV 512
#define NH 512
unsigned char stgn[NV][NH]; // Stegano Image
unsigned char logo[NV][NH]; // Secret Logo

int i, j, seed2;
FILE *fp, *fpo;
main()
{
int count0=0,count1=0,count2=0,count3=0,count4=0;
unsigned char bpl, bit;
char logoname[100];
char writeimge[100];

printf("읽어들일 보안이미지 이름 입력 : ex.(경로설정)d:W(디렉토리위치/파일이름) : ");
gets(writeimge);
printf("저장할 로고이미지 이름 입력 : ex.(경로설정)d:W(디렉토리위치/파일이름) : ");
gets(logoname);
// stegano image read
if((fp=fopen(writeimge,"rb"))==NULL){
printf("Wn>> File Read Error !Wn"); exit(0);
}
fread(stgn, NV, NH, fp);
fclose(fp);
// bit plane extracting
printf("Input random number seed(0-32767) = ");

```

```

scanf("%d", &seed2);
srand(seed2);
for(i=0; i<NV; i++)
{
for(j=0; j<NH; j++)
{
bit=(unsigned char)rand()%5;
ch:
bit=(unsigned char)rand()%5;
if(bit==0 && count0<=131072)
{
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;
//logo[i][j]=128;
count0++;
}
else if(bit==1 && count1<=52429)
{
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;
logo[i][j]=128;
count1++;
}
else if(bit==2 && count2<=39322)
{
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;
//logo[i][j]=128;
count2++;
}
else if(bit==3 && count3<=26214)
{
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;
logo[i][j]=128;

```

```

count3++;
}
else if(bit==4 && count4<=13107)
{
bpl=stgn[i][j]>>bit;
if(bpl%2==0) logo[i][j]=0;
else logo[i][j]=255;
logo[i][j]=128;
count4++;
}
if(bit==0 &&count0>131072)
goto ch;
if(bit==1 && count1>52429)
goto ch;
if(bit==2 && count2>39322)
goto ch;
if(bit==3 && count3>26214)
goto ch;
if(bit==4 && count4>13107)
goto ch;
}
}

if((fpo=fopen(logoname,"wb"))==NULL){
printf("\n>> File Write Error !\n"); exit(0);
}
fwrite(logo, NV, NH, fpo);
fclose(fpo);
}

```

참고문헌

[1] 네이버 지식인 노하우 > 생활상식 참조

http://kin.naver.com/knowhow/entry.php?d1id=5&dir_id=5&docid=493258&qb=RFJNIOybkOumrA==&enc=utf8&pid=fWaeLsoi5TVssu1r4sGsss--430411&sid=StFMgOIL0UoAACvRKOQ

[2] 정보보호프로젝트 실무 PPT 자료 저작권보호 수업자료 참조

[3] “비주얼 C++을 이용한 디지털 영상처리” 참조

(강동준 하종은 저) 출판사 : 사이텍미디어