1) 프로젝트 개요

a) 문제 상황 및 동기

고등학생 때 '암호의 해독'이라는 책을 읽고 암호라는 분야에 관심을 가지게 되었다. 그에 따라 암호학 스터디에 참여하기도 하였고 이번 학기에 핵심 교양 중 '암호학의 이해'라는 과목을 수강하게 되었다. 과목 자체는 무척이나 흥미로웠으나 수업을 듣는 것과 실제 실습 사이에는 큰 간극이 존재함을 머지않아 깨달을 수 있었다. 암호 관련 과제를 푸는데 문자 하나하나 분석하고 경우의 수를 여러 가지로 나누어 같은 작업을 수없이 반복해야 하는 것이 굉장히 불편하였다. 특히, 짧은 문장에서 그러한 현상이 두드러졌다. 기존의 통계 특성과 차이가 나타나기 때문이다. 그러다 보니 암호화나 복호화를 자동으로 해 주는 프로그램을 짠다면 훨씬 편하게 과제를 할 수 있고, 암호의 이해는 물론 프로그래밍 공부에도 큰 도움이 되지 않을까 하는 생각이 들게 되었다. 그리하여 나에게 주어진 이러한 문제 상황을 어떻게 하면 효과적으로 해결할 수 있을지 이번 기회를 통해 고민해 보는 시간을 가지고 싶었다.

b) 가능한 해결 방안 (방향성)

① 직접 손으로 암호문을 계산하되, 그 과정을 효율적으로 만들기 위한 보조 계산 기법을 구상하도록 한다. 그리고 그 효율성을 입증하기 위한 증명 과정 또한 완료하도록 한다.

② 암호화 과정과 복호화 과정을 모두 지원하는 하나의 JAVA 프로그램을 작성한다. 사용자는 암호화 기능을 사용할 지, 복호화 기능을 사용할 지 선택할 수 있으며 경우에 따라 평문 또는 암호문을 입력 받고 추가적으로 key 값과 같은 부수적인 요소들을 받는 방식을 취하는 함수를 만들도록 한다. 이에 대해서는 공백 처리나 입력 요소의 예외 케이스들을 구분하는 것이 중요하다.

③ 암호화 과정은 자동으로 완료해 주고, 복호화 과정은 부분적으로, 즉 문제 해결에 도움을 주는 보조 도구 형식의 JAVA 프로그램을 작성한다. 암호화 과정의 해결 방법은 방법 ②와 동일하게 하고, 복호화 과정의 경우는 해독을 위한 도움말을 주는 방식을 취한다. 예를 들어, substitution cipher의 복호화에 있어서 중요한 것은 평문의 통계 특성을 파악하여 그 정보를 바탕으로 문자를 대응시키는 것이 중요하다. 이 경우에서 암호문에 나타나는 문자들의 출현 빈도를 자동으로 분석해 주는 등의 기능까지만 지원하는 방식으로 하는 것이다.

c) 내가 선택한 방법

방법 ②와 방법 ③을 절충하여 문제 해결을 하도록 한다. 최대한 decryption을 자동적으로 수행해 주는 함수를 만들도록 하되, 경우의 수가 굉장히 많아 효율적인 방법을 찾기 어렵거나 코드 작성 시 예외 케이스들이 과도하게 많이 등장하는 경우 복호화를 위한 힌트, 방향성만을 제시해 주는 함수를 만들도록 한다.

d) 예상되는 모습, 기대 효과

여러 가지 암호들의 암호화 및 복호화 과정을 더욱 깊게 공부하면서 암호에 대한 이해도를 높

일 수 있다. 이전에 손으로 계산하면서 암호를 공부할 때는 계산 과정을 따라가는 것에조차 다소 벅찬 느낌을 받았고 이해하는 데 시간이 오래 걸렸는데 이번 기회를 통해 암호라는 분야 자체에 대한 식견을 높일 수 있을 것이라 생각된다. 그리고 이는 추후 다른 암호들의 구조를 이해하거나 직접 하나의 암호 체계를 만들어 볼 때 탄탄한 밑바탕이 되어줄 것이다.

2) 프로젝트 계획

a) 작업 계획 (계획)

주차	작업 계획
1주차 (2018.10.28 ~ 2018.11.3)	main 함수 구현 프로그램이 실행될 때 어떤 기능을 사용할 것인가? 어떤 암호 방식을 선택할 것인가? (선택지 제공) try catch 구문을 이용해 적정 input을 제외한 다른 모든 값들을 예외 처리하도록 하기
2주차 (2018.11.4 ~ 2018.11.10)	•Shift cipher(=Caesar cipher)의 encryption 및 decryption 함수 구현 1) Encryption 함수: 암호화하고자 하는 문장과 shift시킬 key 값을 입력값으로 받는다. 이때, 문장 혹은 단락이 끝날 때까지 입력이 강제 종료되면 안 되므로 공백 등의 처리에 유의하며 특별한 입력을 통해 문장의 끝을 알린다. 이후에 key 값을 입력받는데 이는 무조건 integer 값을 받도록 한다. 이외의 경우에는 예외 케이스를 잡아 주게 하여 경고 메시지를 출력하도록 만든다. 2) Decryption 함수: 복호화 하고자 하는 문장을 입력받는다. 이때, 암호문은 공백 없이 받도록 한다. (암호문에 띄어 쓰기가 되어 있으면 해독이 무척이나 쉽기 때문) 복호화 방법은 전수 조사를 기준으로 한다. Key 값의 경우의 수가 26가지(mod 26의 경우는 의미가 거의 없으므로 사실상 25가지)에 불과하므로 각 경우에 대한 shift 결과를 모두 출력하여 보여주도록 한다. •Substitution cipher의 encryption 함수 구현 ➔ Encryption 함수: 평문의 각 알파벳을 암호문으로 바꿀 때 어떤 알파벳으로 받을지 그 대응 관계를 설정하도록 만들고 그 이후에 암호화 시킬 문장을 입력 받을 수 있도록 만든다.
3주차 (2018.11.11 ~ 2018.11.17)	•Substitution cipher의 decryption 함수 구현 ➔ Decryption 함수 ⇨ 평문의 통계 특성을 반영하여 그대로 암호문을 해독할 수 있게 만들거나, ⇨ 암호를 해독할 수 있는 힌트까지만 제공하도록

	코드를 짠다. (추후 결정) • Vigenère cipher의 encryption 및 decryption 함수 구현 1) Encryption 함수: 평문을 입력 받고 key length를 몇으로 적용할 지 사용자에게 확인을 받고 이후에 그 length에 해당하는 key word를 받아 그에 따라 평문을 나누어 암호화하도록 한다. 이때 key 값이 여러 개라는 점을 제외하면 shift cipher와 매우 유사한 원리임을 인지한다. 2) Decryption 함수: ⇨ kasiski test와 index of coincidence 등의 연산 방법을 활용하여 암호문을 해독할 수 있게 만들거나, ⇨ 암호를 해독할 수 있는 힌트까지만 제공하도록 코드를 짠다. (추후 결정)
4주차 (2018.11.18 ~ 2018.11.24)	•Transposition cipher의 encryption 및 decryption 함수 구현 1) Encryption 함수: 입력 받은 평문을 5글자씩 나누고 5글자로 이루어진 문자열쌍들을 어떠한 순서로 섞을 것인지 숫자 입력을 통해 그 배열을 정한다. 그리고 평문의 문자들의 개수가 완전히 5로 나누어지지 않을 경우에는 임의의 더미 데이터가 들어가게 만든다. 2) Decryption 함수: ⇨ 배열의 순서쌍을 분석하여 암호문을 해독할 수 있게 만들거나, ⇨ 암호를 해독할 수 있는 힌트까지만 제공하도록 코드를 짠다. (추후 결정)
5주차 (2018.11.25 ~ 2018.12.1)	•Block cipher (SPN structure)의 encryption 함수 구현 ➔ Encryption 함수: 입력 받은 평문을 SPN structure를 기준으로 XOR 연산과 Permutation 연산을 통해 암호화 과정을 생성한다. # Block cipher에 한하여 decryption 과정은 만들지 않을 것이다. Block cipher의 구조가 매우 다양하고 XOR 연산이나 순서쌍 생성에 대한 경우의 수가 굉장히 많이 나오기 때문이다.
6주차 (2018.12.2 ~ 2018.12.8)	프로그램 수정 및 보완

7주차 (2018.12.9 ~ 2018.12.12)	프로그램 테스트 및 마무리 & 제출

b) 작업 계획 (실제 진행)

주차	작업 상황
1주차 (2018.10.28 ~ 2018.11.3)	프로그램 작동의 중심이 될 main 함수의 뼈대 구축 ⇨ try catch와 while, switch를 이용하여 암호 및 복호화 기능 선택 가능 Encryption package 생성
2주차 (2018.11.4 ~ 2018.11.10)	암호별로 암호화 및 복호화 클래스를 생성해 놓으면 파일 관리가 번거로워질 거라고 생각하여 main 함수 안에 모두 구현하기로 결정 ⇨ main 함수 이외의 package 및 class 모두 삭제 main 함수의 작동 및 기능 선택을 위한 설명 추가 Shift cipher의 Encryption 함수 구현 ⇨ 입력받은 평문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ 암호 key 값으로 임의의 정수를 받도록 정의 ⇨ 입력받은 key 값만큼 mod 연산 후 shift가 이루어지도록 구현 ⇨ shift가 이루어진 형태의 암호문 출력 ⇨ command 선택으로 return Shift cipher의 Decryption 함수 구현 *전수조사(brute force)를 따름 ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ 정돈된 암호문을 for 루프에 대입시켜 한 칸씩 shift 연산을 이루어지고 출력되도록 함. 입력된 암호문이 평문이 아니라는 가정 하에 루프를 25번 반복 ⇨ command 선택으로 return Substitution cipher의 Encryption 함수 구현

	⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ Substitution Box (S-box)를 선언 S-box: A부터 Z까지 암호 문자와의 대응쌍의 집합 ⇨ 입력받은 S-box를 배열에 대문자로 저장 ⇨ reformedText 배열 속의 character들을 S-box에 따라 변환 ⇨ 변환 완료된 암호문을 출력 ⇨ command 선택으로 return
3주차 (2018.11.11 ~ 2018.11.17)	Vigenère cipher의 Encryption 함수 구현 ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ key로 사용할 단어 입력 ⇨ keyword를 대문자로 변환 후 keyarr 배열에 shift 수치로 저장하기 위해 integer로 형 변환 ⇨ 규칙에 따라 문자별로 shift 연산 수행 Transposition cipher의 Encryption 함수 구현 ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ key cycle = 5이므로 배열 인자 수가 5의 배수가 되어야 함. 따라서 인자 수가 5의 배수가 아닌 경우에는 더미 데이터를 넣어주도록 함. ⇨ 5개 문자쌍들의 순서를 정수의 형태로 입력받음 ⇨ 기존 reformedText 배열의 인자들을 5개씩 일정하게 묶은 이차원 배열 charArray를 선언 ⇨ charArray의 모든 인자들을 루프를 돌며 바뀐 순서대로 출력 ⇨ command 선택으로 return (Block cipher의 함수를 일부 만들어 업로드하였으나 이후 계획 변경으로 삭제)
4주차 (2018.11.18 ~ 2018.11.24)	Substitution cipher의 Decryption Helper 함수 구현 *완전한 복호화가 아니라 사용자에게 복호화에 대한 힌트를 제시하는 함수 ⇨ 평문 알파벳의 출현 빈도수를 출력

	⇨ 빈출되는 digram과 trigram 쌍 출력 ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ alphabet 배열 선언. 이를 바탕으로 for 루프를 통해 각 알파벳의 출현 빈도수 분석 ⇨ 가장 출현 빈도수가 큰 문자가 평문의 E와 대응될 확률이 높다는 것을 알려주도록 문장 출력 순열 함수 기록
5주차 (2018.11.25 ~ 2018.12.1)	Substitution cipher의 Decryption Helper 함수 수정 ⇨ 출현 빈도수 2위에서 9위까지의 암호 문자들을 순서대로 출력 ⇨ 빈출 문자들의 대응 가능 알파벳 제시 ⇨ 모음 위주의 풀이를 추천하는 힌트 제시 ⇨ digram과 trigram을 사용하라는 힌트 제시
6주차 (2018.12.2 ~ 2018.12.8)	Transposition cipher의 Decryption 함수 구현 *전수조사(brute force)를 따름 ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ 암호문을 5개씩 나누어 120가지의 순서쌍들을 모두 테스트할 것이므로 순열 class 선언 ⇨ integer 변수 pairs를 선언하여 새로운 charArray 배열 선언 ⇨ for 루프를 통해 5개씩의 배열 인자들의 순서쌍들을 뒤섞으며 출력 ⇨ command 선택으로 return 4주차 순열 함수 삭제
7주차 (2018.12.9 ~ 2018.12.12)	Vigenère cipher의 Decryption 함수 구현 (keyword 필요) ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ keyword를 입력받음 ⇨ 입력받은 keyword를 바탕으로 역으로 shift 연산이 수행될 수 있도록 함 (Encryption 과정과 매우 유사한 과정)

	Vigenère cipher의 Key length 추정 함수 구현 (= Kasiski Test 함수 구현) ⇨ 입력받은 암호문의 알파벳을 모두 대문자로 변경하고 알파벳 이외의 특수문자들은 삭제되도록 reformedText 배열 선언 (array of characters) ⇨ 추정되는 key length를 입력받음 ⇨ 두 경우의 index에 해당하는 문자들에 대한 kasiski test를 수행하도록 함 ⇨ Index of coincidence 결과값이 0.038에 근사한다면 틀린 것이고, 0.065에 근사한다면 맞는 key length를 받은 것 (프로젝트 코드에서는 계산값의 오차를 고려하여 참과 거짓의 경계값으로 0.055 설정) 코드 다듬기

3) 프로젝트 결과

a) 계획서 대비 변경점

1. Encryption 및 Decryption 함수들의 구현 과정의 순서가 다소 바뀌었다.

2. Vigenère cipher의 decryption 함수를 두 가지로 나누어 구현하였다.

=> 키를 안다는 전제 하에 암호문과 키를 입력받아 평문을 출력해 주는 함수

=> 키 자체를 모르는 상황에서 키 길이를 분석하기 위한 함수

3. Block cipher 구현 도중 함수 자체를 삭제해 고전 암호 4가지의 암호화 및 복호화에 집중하였다.

b) 깃허브 링크 주소

https://github.com/mindam99/JAVA_prj2018

c) 동작 데모 영상 링크 주소

1. 초기 프로그램 실행 후의 커맨드 입력 창 모습

https://youtu.be/kWZbN3PC7X8

2. Shift cipher Encryption

https://youtu.be/4spKqWsPHP0

3. Substitution cipher Encryption

https://youtu.be/wNG6obTv8lA

4. Vigenère cipher Encryption

https://youtu.be/TfnRUVyXA3c

5. Transposition cipher Encryption

https://youtu.be/h3Opp6l4ecI

6. Shift cipher Decryption

https://youtu.be/4ZTOjm3RS28

7. Substitution cipher Decrypting Hints

https://youtu.be/fK4C6DMqo6s

8. Vigenère cipher Decryption with key

https://youtu.be/RcN2Ig7EruU

9. Vigenère cipher key length guessing

https://youtu.be/CtH-Hd5sup8

10. Transposition cipher Decryption

https://youtu.be/vOXmLVGt538

4) 회고

a) 느낀 점

이 프로젝트를 하는 과정에서 가장 많이 느꼈던 것은, 다른 프로그래밍 과제들을 하면서도 자주 느껴왔던 것이지만, 머릿속에서 자유롭게 떠오르는 수학적 과정들을 프로그램 소스 코드로 옮기는 것이 꽤나 어렵다는 점이었다. 나에게는 단순한 덧셈과 나눗셈 등의 연산이 컴퓨터 입장에서는 여러 타입의 변수들을 새로이 선언 받고, 조건에 맞는 연산자를 이용해야 하는 복잡한 과정이기 때문이다. 특히, 각종 암호 기법들의 복호화 코드를 작성할 때 이를 많이 느꼈다. 평문을 암호화시키는 코드는 모두 비교적 수월하게 완성할 수 있었기에 그 역과정인 복호화도 어렵지 않게 문제 해결이 가능할 것이라고 생각했는데 실상은 전혀 그렇지 않았기 때문이다. Transposition cipher의 경우만 하더라도 암호화 과정에서는 내가 직접 문자열의 순서를 지정해 줄 수 있기 때문에 각 문자 간의 변환이 비교적 자유로웠지만 복호화 과정에서는 발생 가능한 모든 순서쌍을 탐색하는 전수 조사의 방법을 활용함에 따라 이를 위해 순열을 활용해야 했는데, 그 기능의 구현 과정에서 많은 어려움을 느꼈다. 그래도 오랜만에 혼자서 머리를 꽁꽁 싸매고 길게 고민하는 시간을 가질 수 있었을 뿐만 아니라 친구들과 프로젝트에 대한 이야기로 교류를 할 수 있어서 즐거웠다.

처음에는 장기적으로 하나의 프로젝트를 수행해야 한다는 사실이 부담되었으나 코딩 중 막히는 부분을 차근차근 해결해 나가고 내가 구현하고자 한 기능들을 하나하나 완성할 때마다 계획에서 목표했던 바를 이룰 수 있었다는 점에서 짜릿함을 느낄 수 있었다. 그리고 나는 프로그래밍 자체를 사실상 대학에 와서 첫걸음을 뗀 것이기 때문에 잘 하는 친구들을 보고 내심 위축되기도 하였으나 나만의 페이스로 열심히 공부하면 된다는 것을 깨달을 수 있었고, 포기하지만 않는다면 어떤 문제라도 풀 수 있겠다는 자심감도 얻을 수 있었다.

b) 향후 프로젝트 진행 시 고려해야 할 사항

(i) 암호화 및 복호화 프로젝트의 추가 진행

이번 프로젝트에서 substitution cipher의 경우, 완전한 복호화를 지원하는 대신 각 문자의 출현 빈도를 분석해 줌으로써 평문의 문자들과 대응시킬 수 있는 힌트를 제공하는 데에서 그쳤다. 향후 프로젝트를 추가 진행한다면 평문의 문자들과 대응시킨 후, 실제 단어 단위로 잘라 대응쌍이 맞는지 점검해 주고, 그에 따라 힌트를 제공하는 데에서 그치는 것이 아니라 온전한 복호화가 이루어지도록 만들고 싶다. 그리고 그에 대한 S-box 역시 분석하게 만들고 싶다. 이를 위해서는 문자들의 집합 속에서 실제 단어쌍을 추출할 수 있는 메소드의 구현에 중점을 맞추어야 할 것 같다.

Vigenère cipher의 경우, key length에 대해 kasiski examination을 제공하거나, keyword를 안다는 가정 하에 복호화가 이루어질 수 있도록 함수를 구현하였다. 향후 프로젝트 추가 진행 시 key length를 계산하여 계산값을 기준으로 key word의 순서쌍을 모두 생성시켜 자동으로 복호화를 진

행하게 할 수 있도록 하고 싶다. 이를 위해서는 루프를 돌면서 key length를 찾아내고 keyword를 생성하여 shift 연산을 수행할 수 있도록 해야 할 것이며, substitution cipher의 경우와 마찬가지로 실제 단어 단위로 나눌 수 있는 방법의 고민이 필요할 것이다.

Transposition cipher의 경우, key cycle이 5인 경우로 고정하여 암호 및 복호 함수를 구현하였는데 향후에는 유동적인 key cycle에 대하여 기능이 이루어지는 함수를 만들고 싶다. 이를 위해서는 가변적인 길이를 가지는 배열의 관리가 관건이라고 생각한다.

또, 이번 프로젝트에서는 고전 암호에 대해서만 코딩을 하였다. 그러므로 이제 현대 암호를 구현하는 방향으로 새로이 추가 진행하고 싶기도 하다. 현대 암호의 경우, 주로 비트 연산을 활용하기 때문에 아스키 코드라는 좁은 우물에서 벗어날 필요가 있다. 그리고 복호화 함수 구현의 경우, 전수 조사가 아니라 각 암호 기법에 따른 효율적인 복호화 방식을 선택하여 최적의 코드를 연구하여 작성해야 한다.

(ii) 다른 프로젝트의 진행 시

규칙적인 코딩 시간 관리: 프로젝트를 하면서 나름대로 규칙적으로 코딩을 했다고 생각했으나 자기 성찰을 해 보니 다소 그러지 못 했음을 알 수 있었다. 왜냐하면 가끔씩 마지막으로 만들어 놓은 코드의 구조를 잊어버려 다시 파악하는 것에 시간을 소비하였기 때문이다. 따라서 장기적인 프로젝트를 할 때에는 무엇보다 코딩의 페이스를 유지할 수 있는 규칙적인 시간 관리가 필요하다고 생각한다.

코딩 전 알고리즘 계획: 코딩을 하기 전에 알고리즘을 대략적으로 짜고 시작하느냐 안 하느냐가 일의 능률에 생각보다 훨씬 큰 영향을 줌을 알게 되었다. 머릿속에서는 자연스럽게 이루어지는 수학적 연산들이 실제 프로그램 언어로 옮길 때에는 훨씬 복잡해지기 때문에 손에 키보드를 쥐기 전에 매우 간단하게라도 구조도를 그리는 것이 좋다고 생각한다. 앞으로 여러 프로젝트를 수행할 때 일을 수월하게 하기 위해 이를 고려하여 좋은 습관을 길러야겠다.

5) 참고한 자료

* 암호학 관련 서적

A. J. Menezes, V. O. P. C., and S. A. Vanstone, Handbook of applied cryptography. Boca Raton: CRC Press, 2001.

B. J.-P. Aumasson, Serious cryptography: a practical introduction to modern encryption. San Francisco: No Starch Press, 2018.

C. B. Schneier, Applied Cryptography: Protocols, Algorithms, and Source Code in C. John Wiley & Sons, Inc., 2007.

* PPT 자료

A. 핵심 교양 (GEST124) [암호학의 이해] 수업 자료: 고전 암호