

Supplementing the defect of one-way cipher by encryption of fingerprint

Research Members:

Jaewook Lee, Dohyun Ahn

This report has been read and reviewed by a teacher and advisory professor.
Both parties agree that this report is satisfactory and has met all requirements for the
project research.

Dec. 03, 2021

CONTENTS

국문초록

ABSTRACT

I. 서론

1. 연구 목적
2. 연구 방법

II. 이론적 배경

1. 암호화 및 해시 함수
 - (1) 단방향 암호화
 - (2) 비밀키 암호화
 - (3) 공개키 암호화
2. 해시 함수의 문제점
3. 서버 클라이언트 통신
4. 제시할 해결방안

III. 본론

1. 서버 클라이언트 통신
2. 사진 분석 함수
3. 암호화 및 보안

IV. 결론

1. 키 사용의 시간 효율성
2. 일반 해시와 비교했을 때 장점
3. 암호화의 키 결과값
4. 한계 및 의의

참고문헌

TABLES

[Table 1] 반복 등록과 한번 등록에 따른 시간 효율성

[Table 2] 제시한 해결방안과 해시 사용의 해킹 시간 비교

[Table 3] 해시를 사용했을 때 서버에 저장된 암호화된

[Table 4] 각각 다른 지문을 사용했을 때 서버에 저장된 암호화된
비밀번호

[Table 5] 각각 다른 지문과 각 지문으로 생성된 키

FIGURES

[Figure 1] 미국인들의 해킹에 대한 설문조사 결과

[Figure 2] 해시의 원리

[Figure 3] 해시 함수 입력값에 따른 출력값

[Figure 4] 가장 많이 쓰는 비밀번호

[Figure 5] 서버 클라이언트의 원리

[Figure 6] 코드 설계도

[Figure 7] 서버 소켓 - 회원가입

[Figure 8] 회원가입 출력창

[Figure 9] 서버 소켓 - 로그인

[Figure 10] 로그인 출력창

[Figure 11] 아이디와 비밀번호 비교

[Figure 12] 전체적인 클라이언트 소켓

[Figure 13] 사진 분석 함수

[Figure 14] 사진 분석 함수 분석 결과

[Figure 15] 해시값으로 바꾸는 코드

[Figure 16] 반복 등록과 한번 등록에 따른 시간 효율성에 대한 그래프

[Figure 17] 비밀번호 해킹 시간 프로그램

[Figure 18] 각 암호화 방법 별 해킹 시간을 나타낸 그래프

ABSTRACT

Supplementing the defect of one-way cipher by encryption of fingerprint

HS20412, HS20311

Jaewook Lee, Dohyun Ahn

As personal information is getting significant, account security is arising as one of the most important factors. This experiment was done to identify the drawback of one-way cipher, which most of the sites are using. One of the biggest flaws of a one-way cipher is that the same input results in the same output. This can cause Brute Force to easily take people's passwords and cause a chain effect: other people might consecutively get hacked. To prevent this, a fingerprint image was added in the process of signing up so that even though the input is the same, the resulting output will be different. After that, time efficiency and Brute Force time were measured.

As a result, it not only took more time to Brute Force the new program but also developed tolerance to other hacking tools. If this program is commercialized, the overall security level will increase.

국문초록

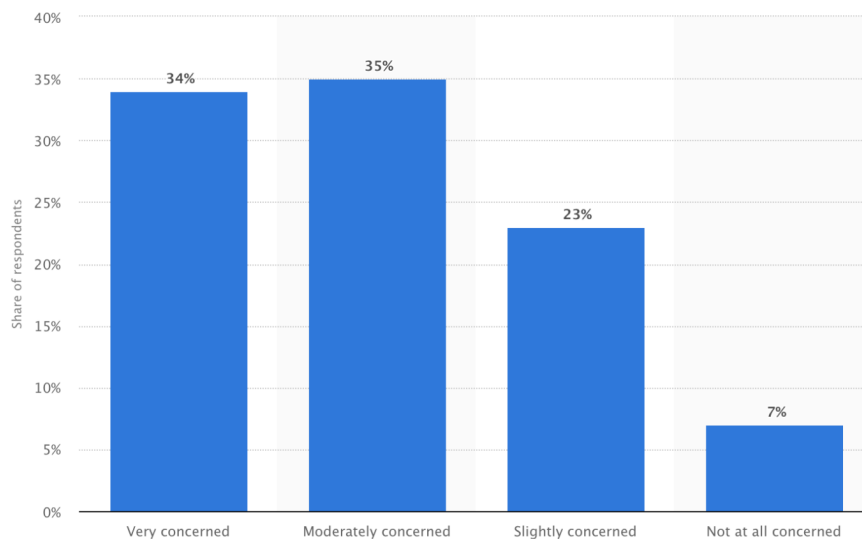
개인정보가 중요해짐에 따라 사람들의 계정을 보호하는 것이 중요해지고 있다. 많은 사이트에서 사용하고 있는 단방향 암호화(해시)의 단점 및 해결방안을 탐구하기 위해 이 연구를 진행하게 되었다. 단방향 암호화의 가장 큰 단점은 같은 입력 값에 따라 같은 출력 값이 나온다는 것이다. 입력 값에 따라 같은 출력 값이 나오게 된다면 무차별 대입 공격에 더욱 취약해져 다른 사용자들도 연쇄적으로 해킹을 당할 수 있다. 이를 방지하기 위해 회원가입 시 '지문'이라는 요소를 추가해 입력값이 같더라도 출력값이 다르게 나오는 프로그램을 만들었다. 그 후 이 프로그램의 시간 효율성 및 무차별 대입 공격 시간 등을 계산한 후 분석하였다.

연구 결과, 단방향 암호화에 비해 해킹하는데 훨씬 많은 시간이 걸렸을 뿐만 아니라 무차별 대입 공격이 아닌 다른 해킹 방법을 상대로도 보안이 강해졌다. 만약 이 프로그램이 사용화 되어 더욱 다양한 웹사이트에서 지문을 사용한다면 전체적인 보안 수준이 상승할 것이다.

I. 서론

1. 연구 목적

정보화 시대가 되면서 암호의 중요성은 지속적으로 커지고 있다. Statista에 의하면 미국인의 34%는 해킹에 대해 많이 걱정하고 있으며 7%의 미국 시민을 제외한 모든 설문 대상은 자신이 해킹당할 상황에 대한 걱정을 표출하였다. 이를 통해 사람들이 자신의 개인정보 보안과 비밀번호에 크게 신경을 쓰고 있다는 것을 알 수 있다. 사물 인터넷, 암호화폐 등 보안성이 중요하게 여겨지기 시작하면서 개인적이고 뚫리지 않는 암호화 방법이 주목받고 있다.



[Figure 1] 미국인들의 해킹에 대한 설문조사 결과

이 연구를 시작하기 전 <QR코드 피싱을 방지하기 위한 프로그램 연구>에 대한 논문을 작성하였다. QR코드 피싱을 방지하기 위해 사용한 핵심 기술 중 하나는 해시였다. 해시는 임의의 데이터를 고정된 길이의 데이터로 매핑해 다이제스트로 바꾸는 함수로, 평문에서 암호문으로 암호화할 수는 있지만 암호문에서 평문으로 보고회는 거의 불가능하다는 장점을 가지고 있었다. 이 해시 함수는 많은 사이트에서 로그인할 때 사용되고 있는데, 사용자가 처음 회원가입을 할 때, 비밀번호를 치면 그것을 해시로 변환한 후 저장하고 비밀번호는 저장하지 않는다. 그리고 이런 식으로 비밀번호가 다시

입력되었을 때 다이제스트끼리 비교된다. 하지만 해시 함수의 가장 큰 취약점은 Brute Force, 즉 무차별 대입 공격이다. 해시 함수는 똑같은 입력값에 대한 출력값이 같기 때문에, 해커가 해시화된 패스워드들이 저장된 서버를 해킹했을 때 똑같은 비밀번호를 사용하는 사람들을 통해 패스워드를 알아낼 수 있다. 이에 대한 해결방안으로 지문을 통한 암호화를 개발하게 된다면, 사용자들마다 모두 다른 출력값을 이끌어내 무차별 대입 공격을 대비할 수 있을 것이다.

2. 연구 방법

Python을 통해 지문 패턴 인식 프로그램 및 암호화 프로그램을 생성해 가상의 서버와 클라이언트를 만들 것이다. 그 후, 일반 해시를 쓴 서버-클라이언트와 암호화된 지문을 키로 사용한 서버-클라이언트의 보안성을 무차별 대입 공격으로 확인할 계획이다. 또한, 암호화된 지문을 키로 사용해 로그인 할 때마다 키를 들고오는 방식과 로그인 할 때마다 지문을 입력하는 방식의 속도 차이를 구할 것이다. 뿐만 아니라 같은 비밀번호를 가지고 있는 여러 개의 아이디가 존재한다고 가정한 후, 이들을 해킹하는데 걸리는 시간 차이를 정리할 계획이다.

II. 이론적 배경

1. 암호화 및 해시 함수

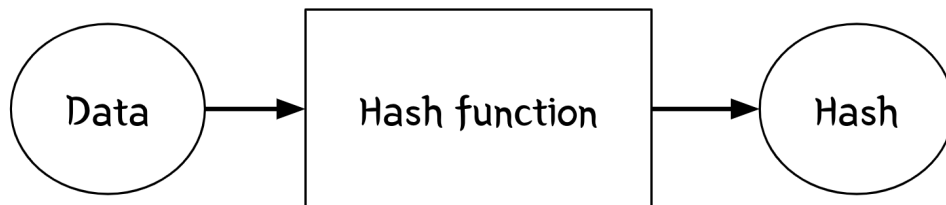
암호화는 특별한 방법을 사용하여 읽지 않고서는 이해하는 것이 불가능하도록 정보를 전달하는 방법이다. 암호화는 크게 세 가지로 나뉘어져 있는데, 바로 단방향 암호화, 비밀키 암호화, 공개키 암호화이다. 다음은 각 암호화의 개념이다.

(1) 단방향 암호화

단방향 암호화는 암호화된 암호문을 다시 평문으로 바꾸는 과정인 복호화가 불가능하다는 특징을 가지고 있다. 주로 신원 증명과 인증 과정에서 사용된다. 단방향 암호화의 가장 대표적인 예시가 해시 함수이다. 해시 함수는 임의의 길이의 데이터가 주어졌을 때, 그것을 언제나 고정된 길이의 데이터로 바꿔주는 함수이다. 그리고 동일한 입력값이 사용된다면 언제나 똑같은 출력값이 나온다. 해시 함수의 특성 상 빠른 데이터 검색을 위해 사용되고, 또 데이터의 무결성을 입증하는 용도로 사용된다. 항상 동일한 값을 출력하기

때문에 데이터가 제3자에 의해 변환되었는지 확인할 수 있다.

해시 함수의 특징은 크게 세 가지로 나뉘는데, 바로 역상 저항성, 제2 역상 저항성, 충돌 저항성이다. 먼저 역상 저항성(Reverse Phase Resistance)은 해시값이 주어져 있더라도 해시값을 통해 입력값을 도출하는 것이 거의 불가능해야 한다는 특징이다. 두번째, 제2 역상 저항성(Second Reverse Phase Resistance)은 입력값과 해시값이 주어졌을 때 똑같은 해시값을 반환하는 서로 다른 데이터를 찾아내기 힘들어야 한다는 특징이다. 마지막으로, 충돌 저항성(Collision Resistance)은 서로 다른 입력값을 넣었을 때 같은 해시값이 나오면 안된다는 특징이다. 충돌 저항성이 있는 이유는 충돌 공격을 방지하기 위해서이다.



[Figure 2] 해시의 원리

(2) 비밀키 암호화

비밀키 암호화는 말 그대로 비밀키를 사용하여 암호화와 복호화하는 과정을 말한다. 대칭 암호화 혹은 단일키 암호화라고도 하는데, 공개키 암호화에 비해 암호화/복호화 속도가 빠르고 알고리즘이 단순하며 파일 크기가 작다는 장점이 있다. 하지만, 사용자가 증가하면서 관리해야 할 키의 수가 많아지고 복호화 키를 아는 사람은 누구나 암호문을 복호화할 수 있어 복호화 키의 비밀성을 철저히 유지시켜야 한다는 단점이 있다.

(3) 공개키 암호화

공개키 암호화는 공개키와 개인키라고 하는 서로 다른 두 개의 키를 사용한다는 특징이 있다. 데이터를 암호화할 때 사용하는 키(공개키)는 공개하고, 복호화할 때 사용하는 키(개인키)는 비밀로 하다보니 수신자에게 공개된 공개키를 통해 암호화한다는 점은 비슷하지만, 복호화는 개인키를 가진 사람만 할 수 있다는 점에서 보안성이 더 뛰어나고 관리해야 할 키의

개수가 적다는 장점이 있다. 그러나, 비밀키 암호화에 비해 암호화/복호화 속도가 느리며 알고리즘이 복잡하고 파일의 크기가 크다는 단점이 있다.

2. 해시 함수의 문제점

제2 역상 저항성과 충돌 저항성을 통해 서로 다른 입력값이 같은 출력값(해시값)을 갖는 것은 방지하였지만, 같은 입력값에 대해 같은 해시값을 출력한다는 점은 변함이 없다(**Figure 3**). 무차별 대입 공격(**Brute Force**)에 취약하다는 뜻이다. 만약 서버에서 해커가 해시화된 비밀번호가 있는 데이터들을 탈취한다면 무차별 대입 방식으로 비밀번호를 알아낼 수 있을 것이다. 뿐만 아니라 해시 함수의 장점인 빠른 속도 또한 여기서는 불리하게 작용한다. 빠른 속도 덕분에 무차별 대입 공격이 빠른 속도로 끝날 수 있고, 또한 끝난다면 그와 똑같은 해시값을 가진 아이디들을 해킹할 수 있을 것이기 때문이다. 예를 들어, 한 사람의 아이디가 `hatebrute11`이고 그에 할당된 해시값 `A`가 있다고 가정해보자. 또한, 그와 똑같은 비밀번호를 가진 계정이 **10**개가 있다고 할 때, 한 해커가 아이디와 그에 할당된 해시값들이 모여있는 데이터를 빼돌리고, 무차별 대입 공격을 통해 `hatebrute11`의 비밀번호를 알아낸다면, 그 해커는 `hatebrute11`과 똑같은 해시값을 가진 모든 사용자들의 비밀번호를 알아낸 것이다. Statista에서 가져온 자료에 따르면 ‘**123456**’이 가장 유명한 비밀번호였고 ‘**picture1**’와 ‘**password**’가 뒤를 이었다. 이렇게 흔한 비밀번호가 해킹되고 그 아이디에 할당된 해시값을 찾게 된다면 이와 같은 값을 가진 모든 아이디의 비밀번호가 유출되는 참사가 일어날 수 있다.

```

Python 3.8.9 (v3.8.9:a743f8192b, Apr 2 2021, 08:12:53)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license()" for more information.
>>> import hashlib
>>> a = 'hello'.encode()
>>> hash_obj = hashlib.sha256()
>>> hash_obj.update(a)
>>> print(hash_obj.hexdigest())
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
>>> b = 'hello'.encode()
>>> hash_obj = hashlib.sha256()
>>> hash_obj.update(b)
>>> key = hash_obj.hexdigest()
>>> print(key)
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
>>> c = 'hi'.encode()
>>> hash_obj = hashlib.sha256()
>>> hash_obj.update(c)
>>> print(hash_obj.hexdigest())
8f434346648f6b96df89dda901c5176b10a6d83961dd3c1ac88b59b2dc327aa4

```

같은 입력값:
같은 출력값

다른 입력값:
다른 출력값

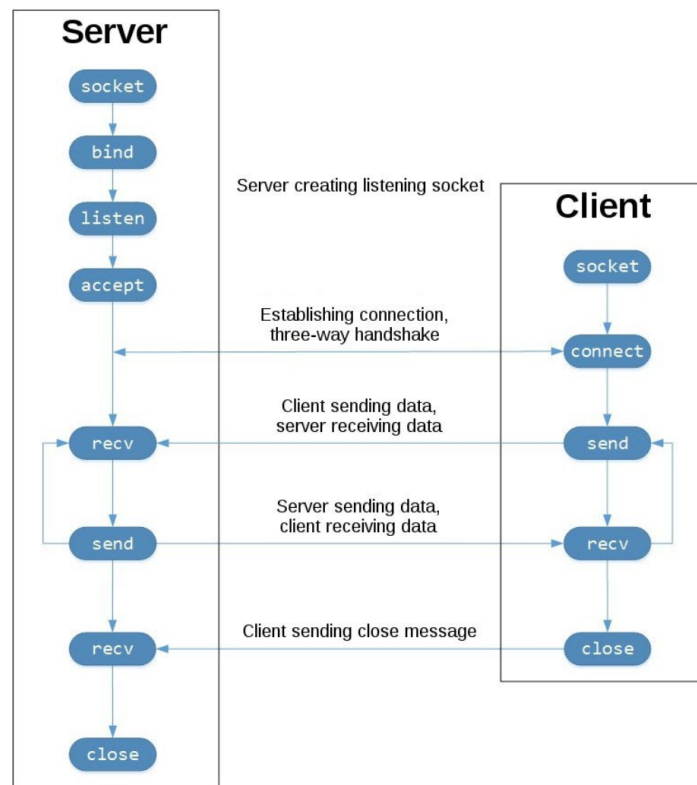
[Figure 3] 해시 함수 입력값에 따른 출력값



[Figure 4] 가장 많이 쓰는 비밀번호

3. 서버 클라이언트 통신

소켓 프로그래밍은 프로그램이 네트워크에서 데이터를 보내고 받을 수 있도록 만들어진 것이 네트워크 소켓이다. 네트워크에 연결하기 위해서는 정해진 프로토콜에 맞게 만들어져야 하고, 이 연구에서는 **OSI 7 Layer**의 4번째 계층, **TCP/IP** 소켓을 사용하였다. **Figure 5**에 나와 있듯이 소켓 프로그래밍은 서버 소켓과 클라이언트 소켓으로 나뉜다. 클라이언트 소켓은 처음 소켓을 생성하고, 또 서버 소켓에 연결을 요청합니다. 그 후 데이터를 송수신하고 소켓을 닫습니다. 서버 소켓에서는 소켓을 생성하고 서버가 접속할 IP주소와 포트번호를 만든 소켓에 결합시킨다. 그 후 클라이언트의 요청을 기다리고(**figure ()**의 **listen**)하고 요청이 도착하면 받아들인다. 그 후 클라이언트 소켓과 송수신한다. 모든 송수신이 완료되었을 때 소켓을 닫는다.



[Figure 5] 서버 클라이언트의 원리

4. 제시한 해결방안

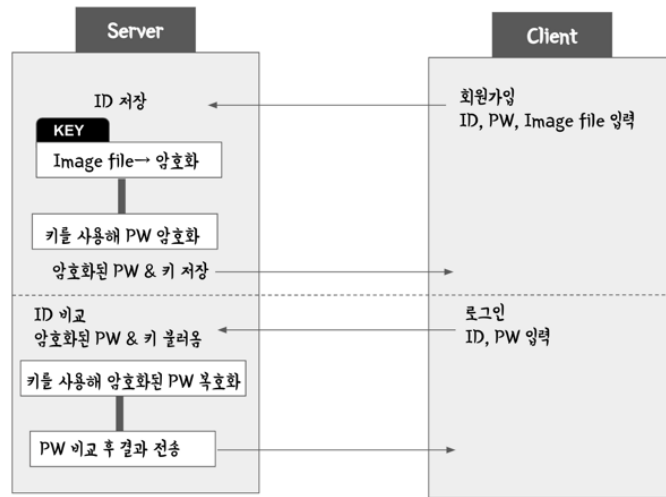
대부분의 웹사이트들은 회원가입을 할 때 아이디와 비밀번호를 입력하고 서버에서 해시 함수를 사용하여 그 비밀번호를 암호문으로 변환한 후 저장을 하고 아이디에 할당한다. 이 때, 비밀번호는 폐기시킨다. 그리고 로그인을 할 때 아이디랑 비밀번호를 입력하면 그 비밀번호를 해시 함수를 통해 암호문으로 변환시킨 뒤 아이디에 할당된 암호문과 비교하고 같을 시 로그인을 시키는 방식이다. 하지만 여기서 비밀번호가 같으면 암호문이 같아 해킹의 위험성이 있다는 문제점이 있다.

이러한 문제점을 해결하기 위해 지문을 사용하는 방식을 도입시키려고 한다. 우선, 회원가입을 할 때 아이디, 비밀번호, 지문을 입력한다. 기존의 방식처럼 비밀번호를 바로 암호화시키지 않고 지문을 먼저 암호화시킨 후 키로 사용한다. 여기서 키란 평문에서 암호문을 바꿀 때 사용하는 것이다. 키로 사용되는 지문을 통해 비밀번호를 암호화한 후, 키와 암호화된 비밀번호를 저장하고 아이디에 할당시킨다. 로그인을 할 때는 아이디와 비밀번호를 입력하면 기존에 서버에 저장된채로 아이디에 할당되어있던 비밀번호를 키를 사용해 복호화하고, 그 비밀번호를 비교하여 같을 시 로그인시킨다. 이런 방법을 사용하면 사람들마다 모두 비밀번호가 같더라도 키(암호화된 지문)이 다르기에 해커의 공격이 어렵다는 장점이 있다.

Ⅲ. 본론

1. 서버 클라이언트 통신

먼저 클라이언트에서 회원가입을 한 후 입력된 정보들을 서버로 보낸다. 서버에서는 ID를 저장하고 받은 이미지 파일을 암호화한다. 이 암호화된 이미지 파일(지문)은 키로 사용되어 비밀번호를 암호화하는데 사용된다. 서버에는 ID와 할당된 암호화된 비밀번호, 그리고 키(암호화된 지문)가 저장된다. 그 후 다시 클라이언트에 로그인을 하라는 신호를 보낸다. 그 후 클라이언트에서 로그인을 위해 ID와 비밀번호를 적으면 그 정보들이 다시 서버로 전송된다. 다시 서버에서 ID를 비교하고 ID가 같다면 그에 할당되어 있던 암호화된 비밀번호와 키를 불러온 후 키를 사용해 암호화된 비밀번호를 복호화 한다. 평문으로 돌아온 비밀번호와 전송된 비밀번호를 비교해 같다면 'Login successful!', 다르다면 'Login not successful!'을 출력한다.



[Figure 6] 코드 설계도

Figure 7의 빨간색 테두리 부분은 회원가입을 담당하는 부분으로 각각 ID1, PW1, File1에 아이디, 비밀번호, 그리고 지문 이미지 파일을 넣는다. 출력창은 Figure 8처럼 나온다.

```

1Server.py - /Users/kis/Downloads/_1Server.py (3.8.9)

HOST = '127.0.0.1'
PORT = 9999

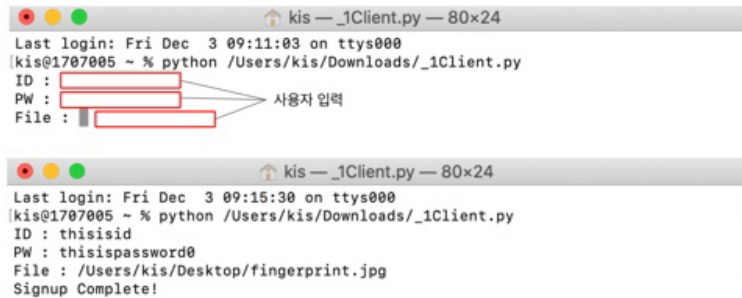
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server_socket.bind((HOST, PORT))
server_socket.listen()

client_socket, addr = server_socket.accept()
print('Connected by', addr)
for i in range(0, 3):
    try:
        data = client_socket.recv(1024)
        if(i==2):
            File=data.decode()
            i+=1
        elif(i==1):
            PW=data.decode()
            i+=1
        elif(i==0):
            ID1 = data.decode()
            i+=1

        if not data:
            print('Disconnected by ' + addr[0]+'.',addr[1])
            break
        print('Received from ' + addr[0]+'.',addr[1] , data.decode(), ID1)
        a=1
    except ConnectionResetError as e:
        print('Disconnected by ' + addr[0]+'.',addr[1])
        break
  
```

[Figure 7] 서버 소켓 - 회원가입



```
kis — _1Client.py — 80x24
Last login: Fri Dec 3 09:11:03 on ttys000
kis@1707005 ~ % python /Users/kis/Downloads/_1Client.py
ID : 
PW : 
File : 
Signup Complete!
```

사용자 입력

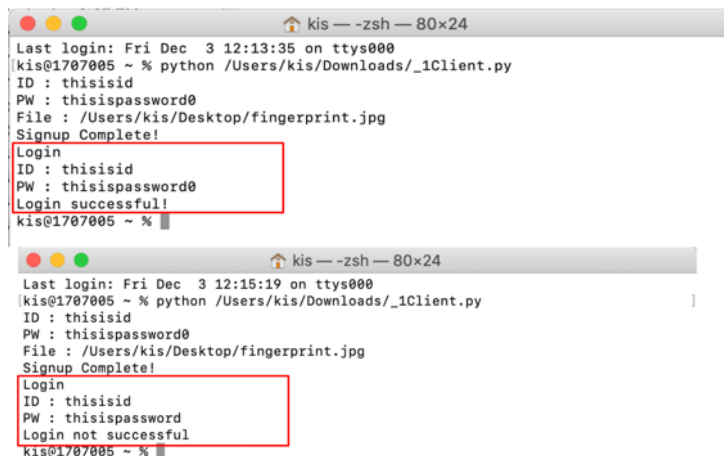
```
kis — _1Client.py — 80x24
Last login: Fri Dec 3 09:15:30 on ttys000
kis@1707005 ~ % python /Users/kis/Downloads/_1Client.py
ID : thisisid
PW : thisispassword0
File : /Users/kis/Desktop/fingerprint.jpg
Signup Complete!
```

[Figure 8] 회원가입 출력창

Figure 9는 로그인을 담당하는 부분으로 각각 ID1_1, PW1_1에 아이디와 비밀번호를 저장한다. 출력창은 Figure 10처럼 나온다.

```
for i in range (0, 2):
    try:
        data1 = client_socket.recv(1024)
        print(i)
        if(i==1):
            PW1_1=data1.decode()
        if(i==0):
            ID1_1 = data1.decode()
        if not data:
            print('Disconnected by ' + addr[0],':',addr[1])
            break
    except ConnectionResetError as e:
        print(' Disconnected by ' + addr[0],':',addr[1])
        break
```

[Figure 9] 서버 소켓 - 로그인



```
kis — -zsh — 80x24
Last login: Fri Dec 3 12:13:35 on ttys000
kis@1707005 ~ % python /Users/kis/Downloads/_1Client.py
ID : thisisid
PW : thisispassword0
File : /Users/kis/Desktop/fingerprint.jpg
Signup Complete!
Login
ID : thisisid
PW : thisispassword0
Login successful!
kis@1707005 ~ %
```

```
kis — -zsh — 80x24
Last login: Fri Dec 3 12:15:19 on ttys000
kis@1707005 ~ % python /Users/kis/Downloads/_1Client.py
ID : thisisid
PW : thisispassword0
File : /Users/kis/Desktop/fingerprint.jpg
Signup Complete!
Login
ID : thisisid
PW : thisispassword
Login not successful
kis@1707005 ~ %
```

[Figure 10] 로그인 출력창

ID끼리 먼저 비교한 후 같으면 PW1_1에 있는 비밀번호와 password(암호화된 비밀번호)를 복호화한 비밀번호를 비교한 후 같으면 data(임의의 값)를 클라이언트에게 전송한다.

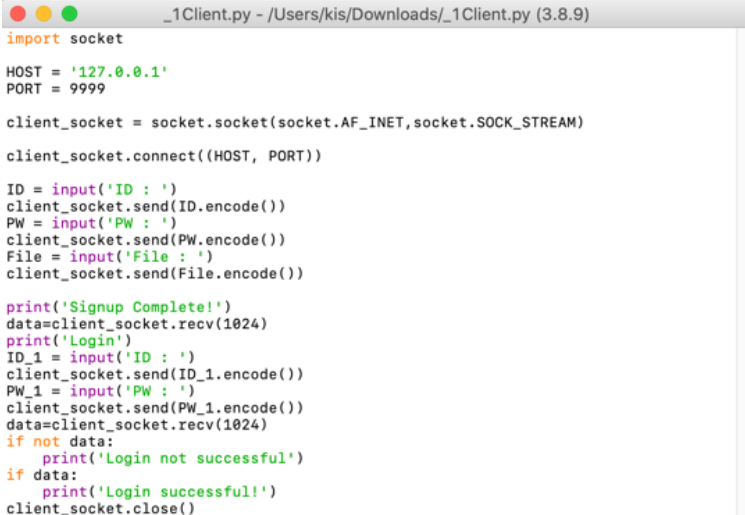
```

-----
if(ID1_1==ID1):
    if(PW1_1==cryptocode.decrypt(password, key)):
        client_socket.send(data)
client_socket.close()
server_socket.close()

```

[Figure 11] 아이디와 비밀번호 비교

클라이언트에서 input()을 사용해 회원가입에서 ID, PW, File을 받은 후 전송하고 수신을 대기한다. 그 후 서버에서 데이터를 받으면 로그인을 시작해 ID와 PW를 서버로 다시 보낸다. 그 후 data가 도착하면 'Login successful'을, 그리고 data가 도착하지 않으면 'Login not successful'을 출력한다.



```

_1Client.py - /Users/kis/Downloads/_1Client.py (3.8.9)
import socket

HOST = '127.0.0.1'
PORT = 9999

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

ID = input('ID : ')
client_socket.send(ID.encode())
PW = input('PW : ')
client_socket.send(PW.encode())
File = input('File : ')
client_socket.send(File.encode())

print('Signup Complete!')
data=client_socket.recv(1024)
print('Login')
ID_1 = input('ID : ')
client_socket.send(ID_1.encode())
PW_1 = input('PW : ')
client_socket.send(PW_1.encode())
data=client_socket.recv(1024)
if not data:
    print('Login not successful')
if data:
    print('Login successful!')
client_socket.close()

```

[Figure 12] 전체적인 클라이언트 소켓

2. 사진 분석 함수

Figure 13처럼 fingerprint()라는 함수를 만들고 이미지 파일을 받는다. 파일을 읽은 후 그레이 스케일로 변화 후 200x200크기로 바꾼다. 이 지문 이미지의 평균값을 구한 후 평균값을 기준으로 0과 1로 변환한다. 그 후 2진수 문자열을 16진수 문자열로 변환한 후 그 값을 리턴해준다. 0과 1로 변환된 200x200사이의 지문은 Figure 14와 같다.

```
def fingerprint(file_1):
    a=file_1
    img = cv2.imread(a)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    gray = cv2.resize(gray, (200,200))
    avg = gray.mean()
    bin = 1 * (gray > avg)
    print(bin)
    dhash = []
    for row in bin.tolist():
        k = ''.join([str(i) for i in row])
        dhash.append('%02x'%(int(k,2)))
    dhash = ''.join(dhash)
    return dhash
```

[Figure 13] 사진 분석 함수



[Figure 14] 사진 분석 함수 분석 결과

3. 암호화 및 보안

Figure 15에서 위의 리턴된 16진수 값을 **z**에 저장한다. 그 후 빨간 테두리 안에 있는 부분이 **z**를 해시 함수에 넣은 후 해시값으로 바꾼다. 그리고 이 값이 비밀번호를 암호화 할 키로 사용된다. **Cryptocode**모듈을 이용해 **PW1**에 저장되어 있는 비밀번호를 **key**를 통해 암호화 한 후 **password**에 저장한다. 여기서 해시값으로 바꿀 때는 **SHA-256**으로 바꿔 더욱 보안성을

높인다.

```
z=fingerprint(File1)
h = z.encode()
hash_obj = hashlib.sha256()
hash_obj.update(h)
key = hash_obj.hexdigest()

password=cryptocode.encrypt(PW1, key)
print(password)
```

[Figure 15] 해시값으로 바꾸는 함수

IV. 결론

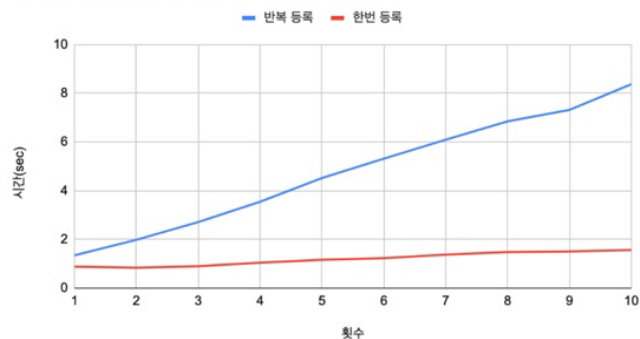
1. 키 사용의 시간 효율성

몇몇 사람들은 클라이언트가 로그인할 때마다 서버에서 키를 꺼내오는 것보다 바로바로 지문을 입력해 계산하는 것이 더 효과적이라고 말할 수도 있다. 서버 안 아이디에 키를 할당하는 것이 더 효율적이라는 것을 증명하기 위해 로그인 횟수에 따른 시간 효율성을 정리하였다.

	지문 반복 등록	지문 한번 등록
1	1.348901033	0.8814089298
2	1.982553244	0.838242054
3	2.712980032	0.9010169506
4	3.54549098	1.04553771
5	4.521087885	1.161166668
6	5.314428091	1.229015112
7	6.094354868	1.37676692
8	6.849169731	1.484253883
9	7.318875074	1.505735874
10	8.374009132	1.568352938

[Table 1] 반복 등록과 한번 등록에 따른 시간 효율성

반복 등록과 한번 등록에 따른 시간 효율성



[Figure 16] 반복 등록과 한번 등록에 따른 시간 효율성에 대한 그래프

x축은 로그인 횟수, y축은 그에 따라 걸리는 계산 시간이다. 파란 선은 로그인할 때마다 지문 등록을 하는 경우고 빨간 선은 한번만 등록하는 경우이다. 지문을 반복적으로 등록할 경우에 처음 로그인 할 때는 1.349초가 걸렸지만 10번 로그인을 시도할 경우에 걸리는 시간은 8.374초였다. 무려 6.2배가량 늘어났다. 반면 지문을 처음 회원가입 할 때만 등록하는 경우에 처음 로그인 시 걸리는 시간은 0.881초였고, 10번 로그인을 시도할 시에

걸리는 시간은 **1.568초**로 지문 반복 등록에 비해 시간 효율성이 압도적으로 높은 것을 볼 수 있다. 뿐만 아니라 지문데이터가 클 수록 시간이 더 걸리는데, 등록 시에만 계산을 하게 된다면 그 많은 계산 과정이 모두 생략될 수 있을 것이다.

2. 일반 해시와 비교했을 때 장점

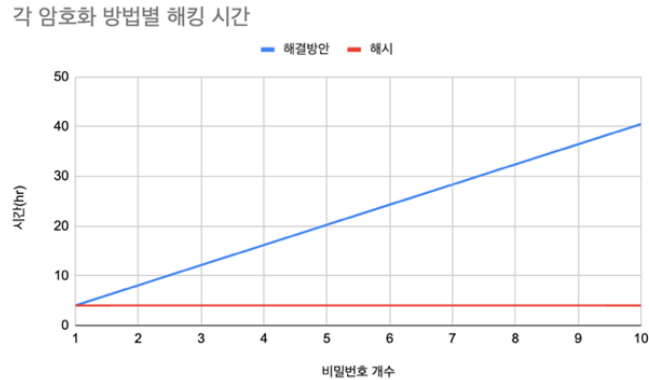
해시의 단점에서 서술했듯이 해커가 해시값을 가진 상태로 한 비밀번호를 무차별 대입 공격을 통해 해킹하게 된다면 동일한 해시값을 가진 동일한 비밀번호들이 모조리 해킹될 수 있다. **Figure 18**의 프로그램을 사용해 **8**자리 소문자 비밀번호를 사용했을 때 동일한 비밀번호들이 해시값을 가지고 있는 해커에게 무차별 대입 공격으로 인해 해킹되는 시간을 표로 정리했다.

Your password is **8** characters long and has **208,827,064,576** combinations.
It takes **4.05** hours or **0.17** days to crack your password on computer that tries **25,769,803,776** passwords per hour. This is based on a typical PC processor in 2007 and that the processor is under 10% load.

[Figure 17] 비밀번호 해킹 시간 프로그램

	해결방안	해시
1	4.05	4.05
2	8.1	4.05
3	12.15	4.05
4	16.2	4.05
5	20.25	4.05
6	24.3	4.05
7	28.35	4.05
8	32.4	4.05
9	36.45	4.05
10	40.5	4.05

[Table 2] 제시한 해결방안과 해시 사용의 해킹 시간 비교



[Figure 18] 각 암호화 방법 별 해킹 시간을 비교한 그래프

먼저 빨간 선, 즉 해시 함수를 사용했을 때 한 비밀번호를 해킹했다면 동일한 10개의 비밀번호를 해킹 하는데는 0시간이 걸릴 것이기 때문에 아무리 많은 비밀번호가 존재하더라도 해시값이 같은 한 알아내기 위해서는 0시간이 소요할 것이다. 하지만 암호화 값이 모두 다른 지문을 사용한 코드에서 비밀번호를 알아내기 위해서는 모든 아이디를 4.05시간동안 해킹해야 한다. 그렇기에 10개의 동일한 비밀번호라도 해시값이 모두 다르기 때문에 총 해킹 시간은 40.5시간이다.

무차별 대입이 아닌 공격에도 보안이 보장되어 있다. 만약 해커가 직접 서버에 접근한다고 가정했을 때 이 코드를 사용하는데 훨씬 많은 시간이 소요 될 것이다. 그 이유는 원래 해쉬화된 비밀번호를 뚫을 때 걸리는 시간이 n 이면 이 경우엔 키도 뚫어야 한다. 키를 뚫는데 소요되는 시가니 m 이라면 총 해킹하기 위해서는 $n*m$ 시간이 걸릴 것이다.

3. 암호화의 키 결과값

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
'hello'	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824	2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824
'hello0'	5a936ee19a0cf3c70d8cb0006111b7a52f45ec01703e0af8cdc8c6d81ac5850c	5a936ee19a0cf3c70d8cb0006111b7a52f45ec01703e0af8cdc8c6d81ac5850c	5a936ee19a0cf3c70d8cb0006111b7a52f45ec01703e0af8cdc8c6d81ac5850c	5a936ee19a0cf3c70d8cb0006111b7a52f45ec01703e0af8cdc8c6d81ac5850c	5a936ee19a0cf3c70d8cb0006111b7a52f45ec01703e0af8cdc8c6d81ac5850c
'hello0!'	4ebffaacc35d6573369f065b938dcc61acbf2a8c6d81ac5850c198604d63440	4ebffaacc35d6573369f065b938dcc61acbf2a8c6d81ac5850c198604d63440	4ebffaacc35d6573369f065b938dcc61acbf2a8c6d81ac5850c198604d63440	4ebffaacc35d6573369f065b938dcc61acbf2a8c6d81ac5850c198604d63440	4ebffaacc35d6573369f065b938dcc61acbf2a8c6d81ac5850c198604d63440
'Hello0!@@'	c6f6751a8ea1d3fd407f29b9bdf2cd6124bb88b73ff6da10c8c6c0c725c140d7	c6f6751a8ea1d3fd407f29b9bdf2cd6124bb88b73ff6da10c8c6c0c725c140d7	c6f6751a8ea1d3fd407f29b9bdf2cd6124bb88b73ff6da10c8c6c0c725c140d7	c6f6751a8ea1d3fd407f29b9bdf2cd6124bb88b73ff6da10c8c6c0c725c140d7	c6f6751a8ea1d3fd407f29b9bdf2cd6124bb88b73ff6da10c8c6c0c725c140d7




[Table 3] 해시를 사용했을 때 서버에 저장된 암호화된 비밀번호

위의 table 3은 해시 함수를 반복적으로 사용했을 때 서버에 저장되는 해시값(암호화된 비밀번호 값)이다. 비밀번호들은 모두 다른 해시값을 가지고 있지만 각각의 해시값은 Trial 1에서 Trial 5까지 모두 같다. 해시는 애초에 같은 값이 입력된다면 똑같은 값이 출력되기에 Trial1에서 Trial5까지의 값은 모두 같다.

	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
'hello'	ZbqLEE4="R5pGbncG6DAaSqB6ncBQ="9mid4EcDym2mfeGQsn6E0w=="AU3IiHjWcxsaQkywrTmnQ==	pkWKNkk="eVImDLKk9beHLJjMFRzOOW="7Ng08nXvpyHTxedQX02/vA="O1JH39/xrYGbOu8xxf4Gow==	EmvPMd4="B0NnqOMGvPC4vzAE9loGUQ="XhxrYKy6Xu3JBQ9yJFptlg="EKm8DLGzJab9yMMRse7vQ==	ON78ix0="IZIK4ChU1wiSJecI46Mwlg="HtfyDag0xvI7CGV/xM+pYA="1PSrBmp6dGWxd5BmEVHLYg==	XGjIPI0="1L0q5/Nc8RC1XWJDEYg48A="lwNUQY9dVXS+8EKSIQ4wg=="rwlTav/QRSSXVx4zaqJVA==
'hello0'	fw7g0avD*e+iPlyfYQnKoXlo7wquRQ="e7JdyAzbnpI/2VWtsAgFGPw="Qqs5K3uVqqQHzo91WRGQ+g==	BeIGW54M*777pOUmnz9y+REF027niQ="xWTA96jcl/CUw6PfwYcDvA="FnMYuFV6BwGOpr+Auip7EA==	0drkr2a*ZxdxkeoBy560TIR0GYnYdA="Tma5q9EEIRUAaroDJrJCHQgQA="ULUqNoW6Adrc/Wegq3+dLg==	dqjISkhk*Y9n1naYwmOwL8Y4ELwmA="LrSpLntfzrzef+ORDL9gQA="ULUqNoW6Adrc/Wegq3+dLg==	SxmPdFdb*ntPWJhlu n5NqNSKkZAIQA="H W1syOCwmXX/aM5c30aN0w="RbBzb2kc57vqPvSFMuJ2DQ==
'hello0!'	Sp4Ky9NWNQ="IS4iVLE/nAAuFUu+5H1sQ="JO0J5Hq+JFINs10xvrodA=="w06SxIRCSxcFU3BpJUNbwQ==	eGa4yd/odQ="KGgGf+wqEww7/cHblkLrgQ="IMkJim2wyxuGWfPXHMdRg="SbG9JGQdZM9T SYRbeb4mfA==	5B7XIPYrpg="IRZwl3G/XlZmL4V8pLIqQ="Ldu+hmmfll26lkQXdQNoLw="dqr5/h0nWnn e7IPbRoz5g==	Qh5rjP5mWA="kerefXjZgxgBM2ZYT1zx2ZSw="W5GpSJ5SDSQ6vmSsuatUw="SKo8q*Z582Z3OIS+ILY1uQ==	aNXRU7DXCw="36C xjZgxgBM2ZYT1zx2ZSw="W5GpSJ5SDSQ6vmSsuatUw="SKo8q*Z582Z3OIS+ILY1uQ==
'Hello0!@@'	jpthagmBlvKv*AJEYEk6+zy+hGXOtdLhPA="mD7H6lteNrCddPx2IEIG3w="56m7qEWv0pEqgnZrAsrg==	U34K8gAnuwXriPqFJg="DW6OA3vx+jmz2M/HLN8uSQ="fq70knDosl+wez0Q75IEdA==	O7dW3oPJAKbY*8jldYbKH6VGZitZSbk/yLQ="M4TF+PWlxtq0VifQJ+5rg="9rQO/hj8J2vOqglw0ow==	4LaH/0VNH7N6*vGMGoNc1GhlbzNtXm5wJ4A="uNldEAnnyx9l64yydzbgw="rbbk1a/CXEILPvNITKwg==	P+RAG52HhCj*qO2694pt61xAQd9JZScFJA="JbVztu0mlk+mcopn KGTvI6g="eaADJIBxS+ywQZV9kMH/yg==

[Table 4] 각각 다른 지문을 사용했을 때 서버에 저장된 암호화된 비밀번호

Table 4의 모든 칸은 다른 값을 보유하고 있다. 그 이유는 지문을 키로 사용하였고, 또 지문은 모든 사람들이 다르기 때문에 코드를 실행할 때마다 다른 값이 나오게 된다. 뿐만 아니라 키가 해시값을 사용했기에 전체적인 암호가 너무 길어지는 것 또한 방지할 수 있다.

지문				
키	b12355b09d7698e91252 2d72a6eec16e83f0e2bc8 68c2159a4e00504dc294 4ad	e58a22f58bbcc02f5 11dba1d9a03caf44c 40e61b1598422337 12ef796b61e30	62ba0fae145265fe29 38fae3372fc51c69f2e 0d77c8a24997bc940 3ec7e9fb89	f14226305291fb3ffb7 ce476e374387acdee 5366a82684b1cb339 0c4e5b7706

[Table 5] 각각 다른 지문과 각 지문으로 생성된 키

위 Table()은 각 지문이 가지는 **key**값을 보여준다. 200x200의 사진을 모두 이진수로 바꾼 다음 또 16진법으로 바꿨기에 길이가 매우 길어져야 하지만, 해시 함수를 사용해 고정된 길이의 값으로 변환할 수 있었다.

4. 한계 및 의의

연구를 하기 위해 만든 코드이기 때문에 한번의 로그인만 보여줄 뿐만 아니라 서버에 아이디를 저장하고 또 그에 따른 비밀번호와 키를 임시적으로 할당할 수 밖에 없었다. 또 여러 아이디, 비밀번호, 그리고 키를 저장하지 않았기에 연구에서는 단순히 ID두개가 일치 하는지만 확인하면 됐지만, 만약 일상생활에 쓰이게 된다면 리스트 사이에서 ID를 뽑아 와야 할 것이다. 서버에 아이디, 비밀번호, 그리고 키를 임시적으로 할당했기에 수시로 프로그램을 다시 돌려야 했다.

비록 임시적으로 저장할 수 밖에 없었지만, 만약 실제로 사용된다면 보안성을 대폭 높여질 것이다. 기술이 발전해 휴대폰으로도 간단히 지문인식을 할 수 있기 때문에 회원가입을 할 때 지문인식을 하는 간단한

기술만 도입된다면 무차별 대입 공격으로 인해 비밀번호가 빠져나갈 확률은 희박해진다.

REFERENCE

- Buchholz, K., & Richter, F. (2021, February 9). Infographic: The most popular passwords around the world. Statista Infographics. Retrieved December 3, 2021, from <https://www.statista.com/chart/16922/most-popular-passwords-2017-and-2018/>.
- Kunst, A. (2019, September 3). U.S. concern about account hacking 2017. Statista. Retrieved December 3, 2021, from <https://www.statista.com/statistics/380908/concerns-about-personal-accounts-being-hacked/>.

정진호, 차영욱. (2020). 접속로그 기반의 동적솔트를 이용한 패스워드
보안강화. 한국정보기술학회논문지, 18(9), 89-97.

허용준, 홍충선, 이대영. (2002). Mobile IPv6에서 단방향 공개키 암호화 기법과
비밀키를 이용한 등록 프로토콜. 한국통신학회 학술대회논문집, (),
2222-2225.

Brute Force Calculator. (n.d.). Retrieved November 5, 2021, from
https://tmedweb.tulane.edu/content_open/bfcalc.php?uc