**ssh-key**

**MAC OS 기준 설명입니다.**

ssh-key는 Secure Shell key의 약자이다. 쉘(터미널)에서 네트워크를 통해 데이터를 주고 받을 때 상대 컴퓨터와 내 컴퓨터가 서로의 신원을 확인하기 위해 사용하는 키이다. 깃허브는 2021년 8월 부로 HTTPS 비밀번호 인증 방식을 통한 인증을 폐기했다. 따라서 이전처럼 “git push”를 입력하고 아이디 비밀번호를 입력하면 인증 방식을 바꾸라는 메세지가 나오고는 한다.

"remote: Support for password authentication was removed on August 13, 2021. remote: Please see https://docs.github.com/en/get-started/getting-started-with-git/about-remote-repositories#cloning-with-https-urls for information on currently recommended modes of authentication."

비밀번호 인증 방식이 폐기되었으니 위 링크에서 권장하는 방식으로 바꾸라는 얘기다. 비밀번호 인증 방식은 브루트포스 알고리즘을 통한 공격과 비밀번호를 재사용하는 사람들이 많아 해커들의 공격에 취약했기에 폐기했다고 한다. 어쨌거나 그래서 현재 깃허브에서 권장하는 인증 방식은 크게 두 가지인데 여기서는 첫 번째 인증 방식인 ssh-key를 통한 인증을 다루려고 한다.

ssh는 원격 저장소(내가 파일을 전송할 컴퓨터(서버))와 로컬 저장소(내 컴퓨터)가 서로의 신원을 확인하기 위해 대조하는 키이다. 터미널에서는 다음 명령어를 통해 생성한다.

ssh-keygen -t ed25519 -C “user\_email@example.com”

**“user\_email@example.com” 안에는 본인의 이메일 주소를 넣어야 한다.**

ssh-keygen 명령어는 암호화된 키를 생성하는 명령어이다. 뒤에 달린 -t 옵션은 type 옵션으로 어떤 유형으로 암호화할 것인지 고르는 옵션이다. 널리 사용되는 RSA방식이나 DSA등의 옵션도 있지만 깃허브에서는 ed25519라는 암호화 방식을 권장한다. ed22519 방식은 RSA와 DSA보다 뛰어난 암호화 방식이라고 한다. 암호화에 대해 다루는 포스팅이 아니므로 넘어가겠다. ed25519(소문자 ed25519가 맞다.) 다음으로 오는 -C 옵션은 Comment의 약자로 주석을 다는 옵션이다. 깃허브에서는 메일을 통해 push한 사람의 로그를 남길 수 있기 때문에 -C “[자신의이메일@메일주소.com](mailto:자신의이메일@메일주소.com)"로 적는다. 위의 커맨드를 입력하면

Enter a file in which to save the key

키를 저장할 파일 위치를 입력하라는 명령이 나오고 그 뒤에 주소가 나오게 되는데 그냥 엔터를 누르면 기본값으로 설정된 디렉토리에 ssh-key가 저장된다. 일반적으로 ssh-key는 기본 디렉토리에 저장하고 사용한다. 엔터를 누르고 ssh-key를 기본 디렉토리에 저장하기로 하면 passphrase를 입력해 ssh-key를 암호화하라는 메세지가 나온다. 이는 나중에 내 컴퓨터에서 다시 ssh-key에 접근하거나 ssh-key를 써야할 때 쓸 암호를 정하는 일이다. 누군가 자기 컴퓨터를 사용해 ssh-key를 건드릴 일이 없다면 그냥 엔터를 눌러서 패스워드 없이 가도 된다. 마지막으로 재확인을 물어보는 엔터를 누르면 ssh-key가 생성된다.

이제 ssh키를 생성했으니 생성한 키를 깃허브에 등록해보자. ssh키는 생성 시 두 개의 키를 만들어내는데 하나는 **개인키(private key)** 이고 하나는 **공개키(public key)**이다. 여기에 대해 알기 위해서는 우선 **대칭키**가 무엇이고 **비대칭키**가 무엇인지 알아야한다. 스파이 영화나 첩보물을 본 적이 한 번쯤 있을 것이다. 체격이 건장한 중년의 남자가 볕이 쏟아지는 카페 테라스에 앉아 무심한 척 커피를 홀짝인다. 신문을 들여다보는 중년의 남자. 그의 곁으로 젊은 여자가 *“쇠푸른 펭귄”*이라 속삭이며 지나간다. 남자는 **대칭키**를 알아듣고 자신이 가지고 있던 usb를 여자에게 건넨다. Usb를 건네받은 여자는 고맙다고 말한다. 그녀는 붉은 드레스의 뒷모습을 끝으로 인파 속으로 유유히 사라져간다. 그녀의 뒷모습을 지켜보며 커피를 홀짝이던 남자의 뒤로 안경을 쓴, 젊고 병약하며 어딘가 불안해 보이는 남성이 다가온다. 남자는 무언가를 직감하고 허리춤에 찬 권총에 손을 올리는데 젊은 남자가 떨리는 목소리로 “쇠푸른 펭귄”이라 말하고 머뭇거린다. 중년의 남자는 그의 신원을 확인하고 화들짝 놀라 끄집어낸 권총을 감출 새도 없이 인파 속으로 달려간다. 남자는 두리번거리며 붉은 드레스를 찾는다. 당연하게도 그녀는 어디에도 없다.

위와 같은 상황은 ***“쇠푸른 펭귄”***이라는 **대칭키**를 사용했기에 문제가 발생했다. 남자와 거래 당사자는 “쇠푸른 펭귄”을 암호로 usb를 건네받는데 합의했지만 중간에서 암호를 가로챈 여자가 등장해 모든 게 틀어졌다. 웹에서도 이렇게 중간에서 암호를 가로채 마치 원래 데이터를 주고 받기로 한 사람처럼 데이터를 갈취하는 해킹 방식이 있는데 이를 **MITM(Man In The Midde)** 공격이라고 한다. 이런 문제를 해결하기 위해 등장한 방식이 바로 **비대칭키** 암호화 방식이다.



이제 상황을 바꿔 그들이 **비대칭키** 방식으로 암호화를 했다고 가정하다. Usb의 데이터를 원하는 붉은 드레스의 여자가 다가오자 남자는 그녀에게 **공개키로** “**스웨터 펭귄”이라고 말한다. 그녀는 잠시 당황하더니 이내 무신경한 척 호흡을 가다듬고 “스웨터 펭귄”이라고 똑같이 회답한다.** 그러나 우리의 답은 “스웨터 펭귄”이 아닌 “쇠푸른 펭귄”이다. 남자가 여자를 쏘아보자 그녀는 도망가듯 빠른 걸음으로 걸어가 인파 속으로 사라진다. 잠시 뒤 아까 그 병약한 남자가 다가오자 중년의 남성은 다시 “스웨터 펭귄”이라고 말한다. 젊고 병약한 남자가 주머니에서 그의 **개인키 메모장을 주섬주섬 꺼낸다. 메모장에는 다음과 같이 쓰여있다.**

**스 – 쇠**

**웨 – 푸**

**터 – 른**

**들어야 할 답 – “리누스 토르발즈”**

**그의 개인키는 ‘스’는 ‘쇠’로 ‘웨’는 ‘푸’로 ‘터’는 ‘른’으로 바꿔 말하도록 안내한다.** “스웨터 펭귄”이라는 중년 남자의 **공개키에** 젊은 남자는 “쇠푸른 펭귄”이라고 답한다. 중년의 남자도 주섬주섬 자신의 **개인키** 메모장을 꺼내 테이블 위에 펼친다. 거기에는 이렇게 써 있다.

If(초성==ㅅ, ㅍ, ㄹ)

Return 리누스

Else if(초성==ㅅ, ㅇ, ㅌ)

Return 스티브

Else(초성==ㅂ, ㄷ, ㅎ)

Return 빌

사자 = 잡스

개미 = 게이츠

펭귄 = 토르발즈

중년의 남자가 “리누스 토르발즈”라고 대답하자 병약한 남자도 자신이 가진 usb를 건낸다. 이번에는 거래가 제대로 성사됐다. **공유된 공개키를 다시 개인키를 활용해 복호화 했기 때문이다.** 이렇듯 비대칭키 암호화 방식은 하나의 **공개키**는 거래 당사자간이 알도록 **공유**하되 **공개키로 만들어진 암호**의답을 알기 위해서는 **개인키**를 사용해야 하는 체계로 이루어져 있다. 당연히, 공개키는 서로 공유하지만 개인키는 공유하지 않는다. 개인키는 누구에게도 알려줘서는 안되는 키이다.

설명을 위해 많이 돌아왔다. 물론 ed25519나 RSA 등의 암호체계는 엄밀히 말해 저것보다는 정교하고 복잡한 암호체계이지만 핵심 아이디어는 비슷하다. 결국에 **비대칭키는 공유된 공개키와 개인키로 구성되며 공개키나 개인키로 암호화된 암호를 풀기 위해서는 공개키와 개인키 모두가 있어야 한다는 것.** 쉽게 이야기하기 위해 바보같은 예시를 들었다 생각해주시라. 애초에 길고 자세하게 알고 싶은 사람이었다면 두 번째 문단에서 깃허브 링크로 빠져나갔거나 구글에서 ed25519를 찾아보고 있으리라 생각하고 계속 이어 나가겠다.



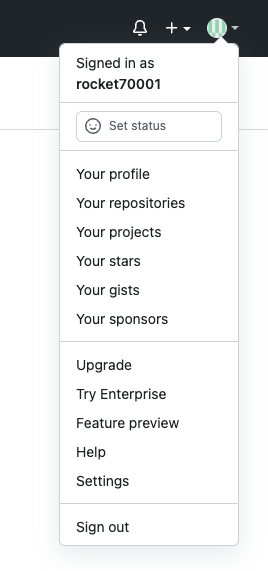
*↑해양 원유 유출 사고로 피해를 입었다가 구출된 필립 아일랜드의 쇠푸른 펭귄.* 체온 유지를 위해 기름을 닦고 스웨터를 입혀 두었다. 이 스웨터 덕분에 구조된 펭귄의 96%는 자연으로 돌아갈 수 있었다고 한다. 귀엽다.

이제 우리는 깃허브에서 권장하는 방식의 비대칭키 암호인 ed25519 방식으로 ssh키를 생성했고 비대칭키가 뭔지도 알았다. 깃허브에 우리의 공용키를 올리기만 하면 된다. 다음과 같은 명령어를 터미널에 입력해 공용키를 얻도록 하자.

pbcopy < ~/.ssh/id\_ed25519.pub

이 명령어를 입력해 제대로 공용키를 얻었다면 붙여넣기 시 공용키가 나온다.

이제 이 공용키를 깃허브에 붙여넣기만 하면 된다.





깃허브 페이지 우측 상단에 있는 프로필을 눌러 세팅으로 들어간다.

텍스트이(가) 표시된 사진

자동 생성된 설명



왼편의 메뉴를 보면 SSH and GPG keys가 있다.

텍스트이(가) 표시된 사진

자동 생성된 설명



New SSH key를 클릭한다.

텍스트이(가) 표시된 사진

자동 생성된 설명

키 부분에 아까 복사한 공용키를 붙여 넣는다. Title은 원하는 대로 지으면 된다. Add SSH key를 누르면 내 ssh키의 공용키가 내 깃허브 계정에 등록된다.

이제 마지막 단계가 남았다. 터미널로 가자.

git remote set-url origin [git@github.com:username/repo.git](mailto:git@github.com:username/repo.git)

위에서 “username”을 본인의 깃허브 유저네임으로, “repo”는 본인의 원격 저장소 이름으로 바꿔주면 된다.

위 명령어를 입력하고 엔터를 치면 이제 내 로컬 저장소가 내 원격저장소로 연결된다. 여기까지 따라왔다면

git add .

git commit -m “”

git push

등의 명령어를 사용하는데 문제 없을 것이다. 당연하지만 해당 명령어를 실행할 현재 디렉토리는 내 로컬 레포지터리여야 한다.

돌이켜보면 간단하고 당연한 일도 처음에는 힘들다. 누구에게나 처음 접하는 순간은 있기 마련이다. 깃과 깃허브에 대해 아무것도 모르던 때 필자는 https 비밀번호 인증방식부터 ssh키를 깃에 등록하기까지 개념을 알아보고 적용하는 데만 다섯 시간 가량 걸렸던 것으로 기억한다. 여기까지 잘 따라왔다면 당신은 나보다 몇 시간이나 더 단축한 것이다. 포스팅이 도움이 되었으면 한다.