Instrukcja podpisywania git commitów za pomocą kluczy GPG

1. Po co podpisywanie commitów?

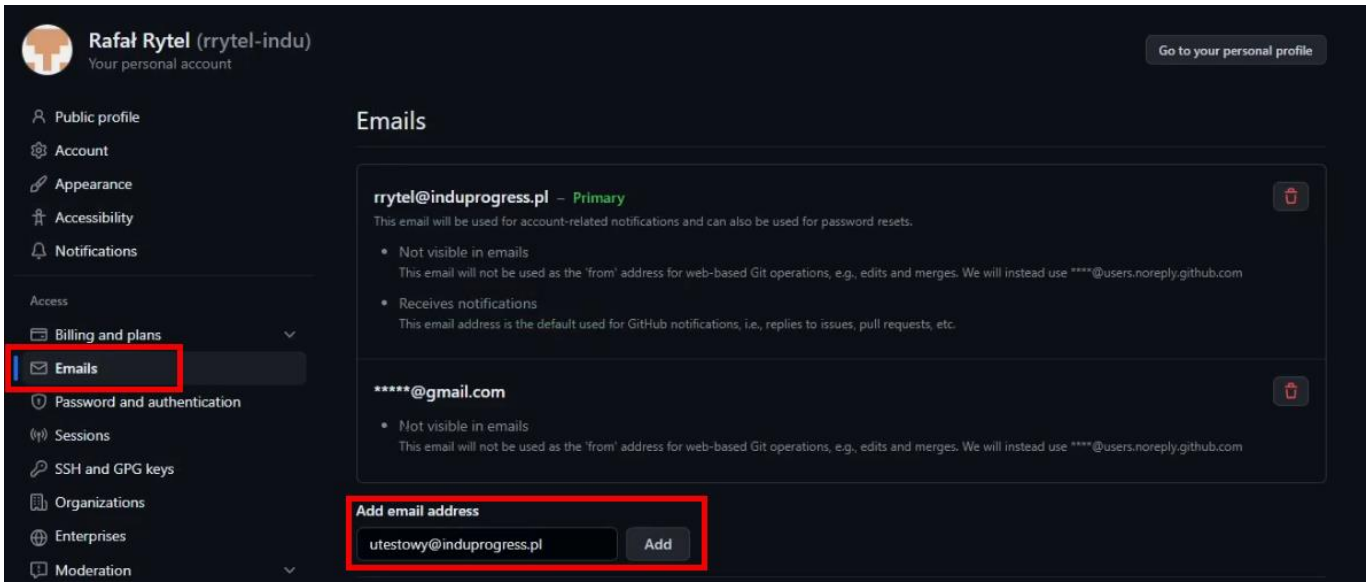
Repozytoria git domyślnie nie zawierają żadnych mechanizmów weryfikacji użytkownika, podobnie jest na Github. Wystarczy wpisać polecenie lub zmodyfikować jeden plik konfiguracyjny i można udawać innego użytkownika nawet jeśli nie ma dostępu do danego repozytorium.

Jest jednak mechanizm za pomocą którego można potwierdzić że dany commit pochodzi od właściwej osoby. Jest to podpisywanie commita kluczami GPG. Można też swój klucz publiczny (który jest wymagany do takiego potwierdzenia) umieścić na serwerze Githuba, wtedy wszystkie commity opisane jako będące od tego użytkownika są sprawdzane przesłanym kluczem.

2. Adres email na github.com

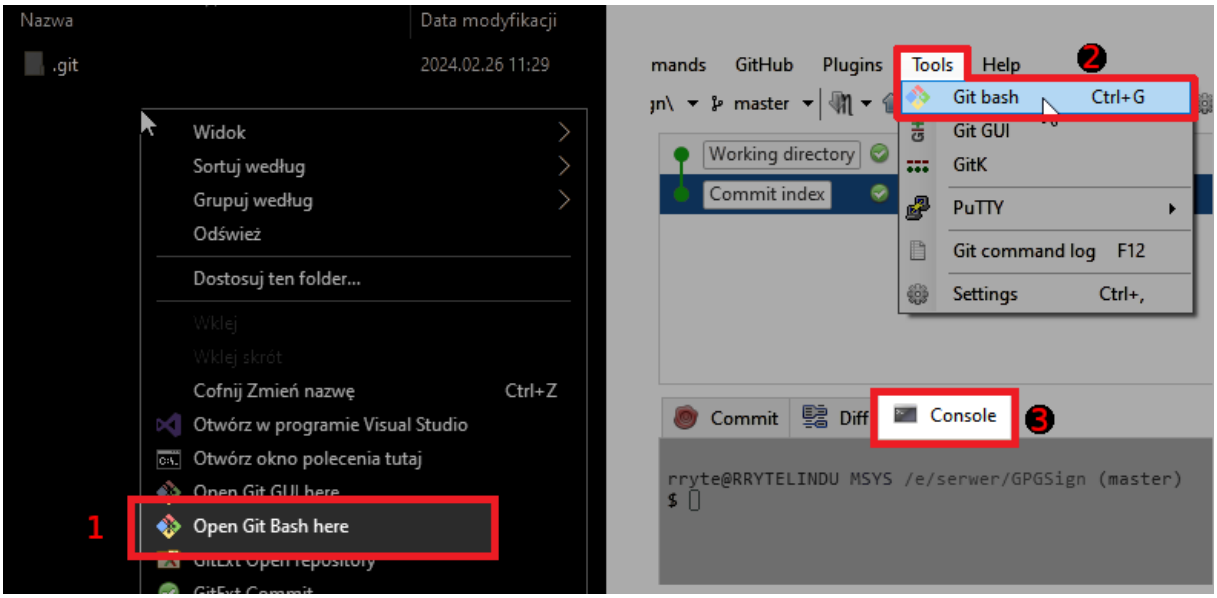
Ponieważ adres email jest widoczny po podpisaniu commita, musimy mieć przypisany do naszego konta na github nasz służbowy adres. Jeśli konto zostało założone z użyciem innego adresu mailowego, nadal możemy przypisać do niego dodatkowy adres email.

Dokonujemy tego w Settings/Emails.



3. Konsola git i wersja gpg

Poniższe polecenia należy wpisywać w konsoli git, można ją otworzyć na kilka sposobów.



1. W dowolnym katalogu klikamy prawym przyciskiem myszy i wybieramy Open Git Bash here.

2. W GitExtensions wybieramy Tools/Git bash

3. W GitExtensions klikamy zakładkę Console

Aby sprawdzić posiadaną wersję gpg (powinna się zainstalować domyślnie razem z gitem) wpisujemy w konsoli `gpg --version`

```
$ gpg --version
gpg (GnuPG) 2.4.5-unknown
libgcrypt 1.9.4-unknown
(...)
```

Na chwile pisania tej instrukcji razem z git version 2.47.1.windows.1 jest domyślnie zainstalowana wersja gpg 2.4.5.

W przypadku braku gpg lub chęci aktualizacji można pobrać najnowszą wersję ze strony:

https://www.gnupg.org/ftp/gcrypt/binary/

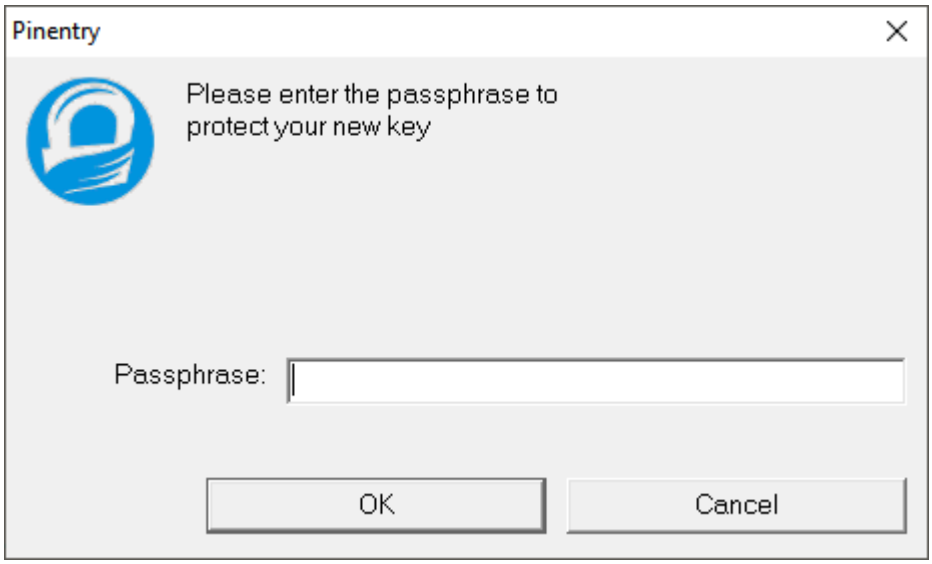
4. Generowanie pary kluczy gpg

Działanie gpg opiera się na stworzeniu pary kluczy: prywatnego (którego nie udostępniamy nikomu) oraz publicznego. Dane zaszyfrowane jednym z tych kluczy można odszyfrować jedynie za pomocą drugiego z pary (podpisanie commita jest za pomocą klucza prywatnego więc do potwierdzenia używamy klucza publicznego).

Parę kluczy generujemy poleceniem:

```
$ gpg --full-generate-key
gpg (GnuPG) 2.2.41-unknown; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Please select what kind of key you want:
   (1) RSA and RSA (default)
   (...)
Your selection? [wybieramy „RSA and RSA” i klikamy enter]
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096 [zalecane wpisanie 4096]
[im wieksza długość klucza tym trudniejszy do złamania]
Requested keysize is 4096 bits
Please specify how long the key should be valid.
         0 = key does not expire
   (...)
Key is valid for? (0) [Klikamy enter – klucz nie ma “daty ważności”]
Key does not expire at all
Is this correct? (y/N) Y

GnuPG needs to construct a user ID to identify your key.

Real name: Imie Nazwisko [wpisujemy swoje imie i nazwisko]
Email address: inazwisko@induprogress.pl [wpisujemy adres email]
Comment: [można dodać komentarz]
You selected this USER-ID:
    "Imie Nazwisko <inazwisko@induprogress.pl>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? O
[wpisujemy O aby zatwierdzić]
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
```

W tym momencie pokaże się okno do wpisania hasła którym zabezpieczony będzie nasz klucz prywatny. Gdy klucz prywatny jest zabezpieczony hasłem to ograniczamy możliwość użycia naszego klucza gdy ktoś inny wejdzie w jego posiadanie (kradzież komputera lub nieautoryzowany dostęp podczas choćby chwilowego odejścia od komputera).




```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: revocation certificate stored as '/c/Users/rryte/.gnupg/openpgp-
revocs.d/983FC7D2F6F78CD8708CF155BC638756C7AEA133.rev'
public and secret key created and signed.

pub   rsa4096 2024-02-26 [SC]
      983FC7D2F6F78CD8708CF155BC638756C7AEA133
uid                      Imie Nazwisko <inazwisko@induprogress.pl>
sub   rsa4096 2024-02-26 [E]
```

Para kluczy została wygenerowana.

Będziemy proszeni o podanie hasła gdy będziemy chcieli użyć klucza prywatnego (np. podczas podpisywania commita). Aby nie trzeba było za każdym razem wpisywać hasła można użyć agenta gpg. Aby sprawdzić czy agent gpg jest uruchomiony można wpisać polecenie:

```
$ gpg-agent
gpg-agent[5288]: gpg-agent running and available
```

Powyższy komunikat informuje nas, że agent działa. Jeśli doinstalujemy gpg, agent jest uruchamiany domyślnie. GPG zainstalowane razem z git nie uruchamia agenta gpg, ale możemy go uruchomić samodzielnie poleceniem:

```
gpgconf.exe --launch gpg-agent
```

Po tym poleceniu agent pozostanie aktywny do momentu ponownego uruchomienia systemu. Aby nie trzeba było za każdym razem go włączać, można dodać do Harmonogramu zadań systemu wpis uruchamiający przy włączeniu systemu lub po zalogowaniu użytkownika polecenie:

```
C:\Program Files\Git\usr\bin\gpgconf.exe --launch gpg-agent
```

Domyślnie agent gpg pamięta hasło przez 10 minut od jego ostatniego wpisania. Aby nie trzeba było co chwila wpisywać hasła można zwiększyć ten czas. W tym celu należy dodać lub zmienić wpisy w pliku konfiguracyjnym gpg-agent.conf. Możemy sprawdzić w jakich katalogach gpg szuka plików konfiguracyjnych poleceniem:

```
$ gpgconf.exe --list-dirs
 (...)
homedir:/c/Users/*****/.gnupg
```

Jednym z miejsc jest katalog domowy użytkownika. Możemy więc w pliku (jeśli nie istnieje to tworzymy) C:\Users\<UserName>\.gnupg\gpg-agent.conf dopisać:

```
default-cache-ttl 86400
max-cache-ttl 604800
```

Czasy podane w sekundach (86400 = 24h, 604800 = 7dni)

default-cache-ttl ile czasu od ostatniego użycia ma pamiętać hasło.

max-cache-ttl ile max czasu ma pamiętać hasło.

Czyli hasło musimy podać nie rzadziej niż co 7 dni, ale pomiędzy użyciami klucza nie może minąć więcej niż 24h (jak wpiszesz hasło w poniedziałek, to jeśli nie użyjesz klucza we wtorek, w środę będziesz musiał wpisać znowu, pomimo, że nie minęło 7 dni).

Po modyfikacji pliku w konsoli git wpisujemy:

```
gpgconf.exe --reload gpg-agent
```

Bez wywołania tego polecenia, zmiany czasu cache nastąpią po ponownym uruchomieniu agenta gpg.

5. Skrypt konfiguracyjny

Dla łatwej konfiguracji został przygotowany skrypt częściowo automatyzujący niektóre operacje.

W konsoli wpisujemy:

```
$ ./GPG_git.bat inazwisko
skrypt integrujący z gitem
Użycie  : E:\serwer\GPG\GPG_git.bat nazwaUżytkownika KeyID
Przykład: E:\serwer\GPG\GPG_git.bat jkowalski ABCDEFGHIJKLMNOP

Skrypt exportuje pliki kluczy: prywatny oraz publiczny do plików
Konfiguruje git aby domyślnie podpisywał wszystkie commity i tagi podanym kluczem
GPG

W przypadku braku któregokolwiek parametru wyświetla listę dostępnych kluczy.

Lista dostępnych kluczy:
sec   rsa4096/ABCDEFGHIJKLMNOP 2024-02-26 [SC]
      4ISDSYHZ7FNPC1FFUBWD61BRPSZOHBXE6MQ85HF1
uid                 [ultimate] Imie Nazwisko <inazwisko@induprogress.pl>
ssb   rsa4096/5R86364I23IB5S9D 2024-02-26 [E]
```

Polecenie pokazało przykład użycia oraz wyświetliło listę kluczy pasujących do opisu inazwisko (wywołanie bez parametrów pokaże listę wszystkich kluczy prywatnych na danym komputerze).

Wywołujemy polecenie ponownie podając w parametrach przedrostek maila oraz KeyID (na powyższym listingu jest to pogrubiony fragment `ABCDEFGHIJKLMNOP`)

```
$ ./GPG_git.bat inazwisko ABCDEFGHIJKLMNOP
Script name: GPG_git.bat
Użytkownik : inazwisko
KeyID      : ABCDEFGHIJKLMNOP

export kluczy
Ustawianie konfiguracji git (user.signingkey ABCDEFGHIJKLMNOP)
W notatniku otwarto plik którego treść należy wkleić w
https://github.com/settings/gpg/new
```

Używając skryptu można pominąć rozdziały "6.Eksportowanie kluczy i backup" oraz "7.Integracja z git – podpisywanie commitów i tagów". Skrypt wykonał niezbędne operacje za nas (oprócz backupu klucza prywatnego w bezpieczne miejsce), jednak warto przeczytać te rozdziały, aby zrozumieć zasadę działania. Możemy przejść do "Integracja z GitHub"

6. Eksportowanie kluczy i backup

Aby zobaczyć listę kluczy prywatnych na naszym komputerze możemy wpisać polecenie (dodanie danych jak adres email zawęzi wyniki tylko do tych pasujących do wzorca, jest to przydatne gdy mamy kilka par kluczy). Pogrubioną czcionką jest wypisany KeyID, przyda się później.

```
$ gpg --list-secret-keys --keyid-format=long inazwisko@induprogress.pl
sec   rsa4096/ABCDEFGHIJKLMNOP 2024-02-26 [SC]
      4ISDSYHZ7FNPC1FFUBWD61BRPSZOHBXE6MQ85HF1
uid                 [ultimate] Imie Nazwisko <inazwisko@induprogress.pl>
ssb   rsa4096/5R86364I23IB5S9D 2024-02-26 [E]
```

Aby wygenerować klucze do plików należy użyć poleceń (zamiast samej nazwy użyszkodnika można podać pełny adres email lub KeyID lub fingerprint – przydatne kiedy mamy kilka kluczy pod podobnymi nazwami):

```
$ gpg --armor --export-secret-key -o inazwisko_privateKey.asc inazwisko
$ gpg --armor --export -o inazwisko_publicKey.asc inazwisko
```

Podczas eksportu klucza prywatnego zostaniemy poproszeni o hasło do klucza.

Chcąc zabezpieczyć klucze przed utratą dobrze jest zrobić sobie kopię zapasową poza komputerem. Dobrze jest umieścić pliki (szczególnie klucz prywatny) w archiwum szyfrowanym 7z (ma tą zaletę w porównaniu do archiwum zip, że ma możliwość zaszyfrowania nazw znajdujących się wewnątrz plików. Możemy ukryć pliki z kluczami w szyfrowanym archiwum „zdjęcia śmiesznych kotków.7z”).

7. Integracja z git – podpisywanie commitów i tagów

Podczas integracji z git wykorzystamy KeyID o którym wspominałem wcześniej (na poniższym przykładzie zamieniamy `ABCDEFGHIJKLMNOP` na nasz KeyID):

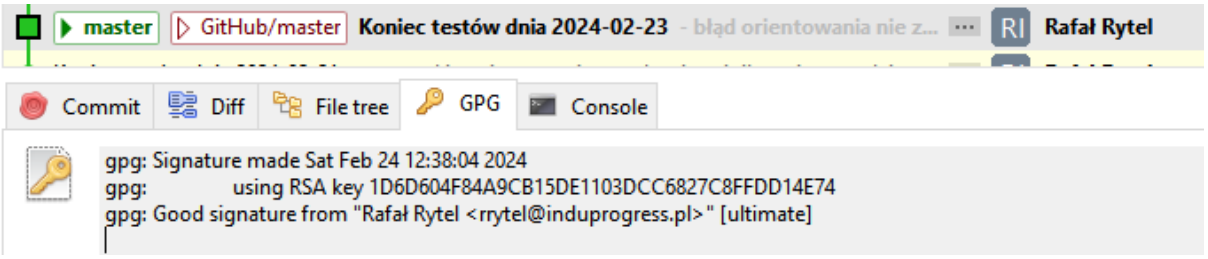
```
git config --global user.signingkey ABCDEFGHIJKLMNOP
git config --global commit.gpgsign true
git config --global tag.gpgsign true
```

Pomijając parametr `--global` sprawiamy że zmiana dotyczy tylko repozytorium lokalnego w którego folderze wykonujemy polecenie np. jeśli chcemy w jednym repozytorium używać innego klucza to mając je aktywne, wpisujemy w konsoli:

```
git config user.signingkey **********
```

i wtedy to jedno repozytorium będzie podpisywane innym kluczem niż inne.

Od tej pory nasze wszystkie commity i tagi w repozytoriach tworzone na naszym komputerze są domyślnie podpisywane kluczem prywatnym i istnieje możliwość sprawdzenia wiarygodności commita (potwierdzenia, że to my go utworzyliśmy)



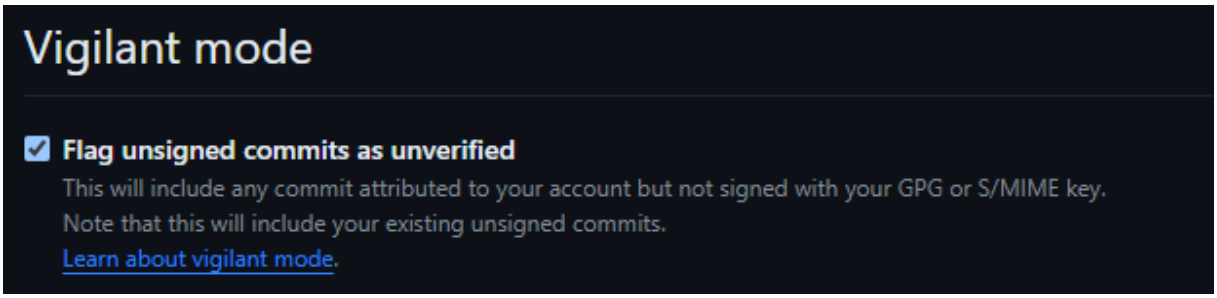
8. Integracja z GitHub

Będąc zalogowanym na swoim osobistym koncie Githuba otwieramy stronę https://github.com/settings/keys lub wybieramy „Settings/SSH and GPG keys”.

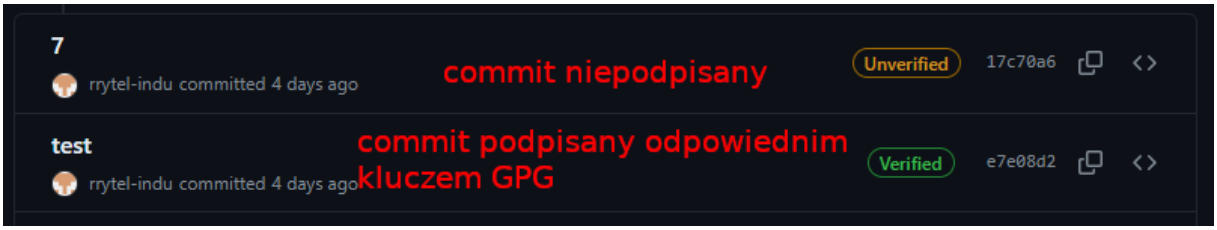
Następnie klikamy „New GPG key”. W polu Key wklejamy zawartość pliku z kluczem publicznym (inazwisko_publicKey.asc wygenerowany wcześniej) i klikamy „Add GPG key”.

Można mieć przypisanych do konta kilka kluczy (np. oddzielne dla różnych adresów email jeśli korzystamy z jednego konta do celów służbowych i prywatnych, lub nawet kilka kluczy w zależności od urządzenia na którym tworzymy commita).

Dodatkowo można zaznaczyć opcję:



Od tej pory wszystkie nasze commity będą sprawdzane czy są podpisane kluczem prywatnym pasującym do wgranego klucza publicznego. Commity poprawnie zweryfikowane będą oznaczone „Verified”, commity niepodpisane lub podpisane niewłaściwym kluczem oznaczane są jako „Unverified”.

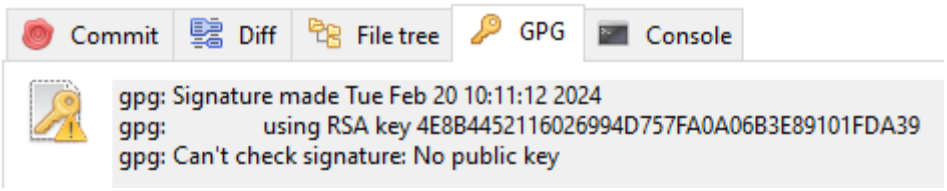


Niestety wszystkie poprzednie commity będą od tej pory oznaczone jako niezweryfikowane (są przecież niepodpisane). Teoretycznie jest możliwość aby podpisać poprzednie commity, ale powoduje to zmianę we wszystkich commitach następujących po modyfikowanym (przy okazji unieważniając wcześniej dokonane podpisy), więc nie jest to stosowane.

Stosowanym czasami półśrodkiem jest po utworzeniu klucza dodanie do repozytorium pustego commita z opisem, że od tej pory stosujemy podpis.

9. Weryfikacja commita lokalnie

Na swoim komputerze będziemy widzieli jedynie, że commit pochodzący od innej osoby jest podpisany kluczem bez dokładnych informacji o nim.



Jeśli chcemy mieć te informacje zawsze widoczne możemy zaimportować u siebie klucze publiczne osób z którymi pracujemy (Jest to czynność opcjonalna - jeśli tego nie zrobimy nie wpłynie to na możliwość zweryfikowania commitów na githubie). Klucz importujemy poleceniem:

```
gpg --import nazwaPlikuZkluczemPublicznym.asc
```

listę posiadanych kluczy możemy sprawdzić poleceniem

```
gpg --list-keys wzorzec
```

Parametr wzorzec jest opcjonalny, służy do zawężenia wyników tylko do kluczy pasujących do wzorca (można wpisać nazwisko, adres mailowy lub jego część) np.:

```
gpg --list-keys @induprogress.pl
```

wypisze tylko klucze osób z naszej firmy.

Literatura dodatkowa:

https://docs.github.com/en/authentication/managing-commit-signature-verification

https://www.gnupg.org/documentation/manuals/gnupg/

https://www.gnupg.org/gph/en/manual/book1.html