Informacje

- **Kontakt:** katarzyna.mazur@umcs.pl
- **Konsultacje:** pokój 412 na 4 piętrze, przed konsultacjami proszę o wiadomość mailową
- **Zasady zaliczenia:**
- **Materiały, aktualności, zmiany terminów zajęć:** `https://kampus.umcs.pl/course/view.php?id=15080`

Teoria

PGP (Pretty Good Privacy Całkiem Niezła Prywatność) jest kryptosystemem (tzn. systemem szyfrująco-deszyfrującym) autorstwa Phila Zimmermanna wykorzystującym ideę klucza publicznego. Podstawowym zastosowaniem PGP jest szyfrowanie poczty elektronicznej, transmitowanej przez kanały nie dające gwarancji poufności ani integralności poczty. Przez poufność rozumiemy tu niemożność podglądania zawartości listów przez osoby trzecie; przez integralność niemożność wprowadzania przez takie osoby modyfikacji do treści listu. PGP pozwala nie tylko szyfrować listy, aby uniemożliwić ich podglądanie, ale także sygnować (podpisywać) listy zaszyfrowane lub niezaszyfrowane w sposób umożliwiający adresatowi (adresatom) stwierdzenie, czy list pochodzi rzeczywiście od nadawcy, oraz czy jego treść nie była po podpisaniu modyfikowana przez osoby trzecie. Szczególnie istotny z punktu widzenia użytkownika poczty elektronicznej jest fakt, że techniki szyfrowania oparte o metodę klucza publicznego nie wymagają wcześniejszego przekazania klucza szyfrowania/deszyfrowania kanałem bezpiecznym (tzn. gwarantującym poufność). Dzięki temu, używając PGP, mogą ze sobą korespondować osoby, dla których poczta elektroniczna (kanał niepoufny) jest jedyną formą kontaktu. Weryfikacja kluczy opiera się o sieć zaufania (web of trust).

Za pomocą PGP można wygenerować parę kluczy (prywatny/publiczny), wyeksporotwać posiadane klucze do pliku, zaimportować klucz z pliku, uzyskać informacje o posiadaczu klucza (***key fingerprint***), podpisać klucz innej osoby. Odcisku tego użyć można do weryfikacji klucza innym kanałem, niekoniecznie bezpiecznym z punktu widzenia kryptografii, ale dającym nam pewność, że po drugiej stronie jest osoba, której się spodziewamy. Jeśli znamy właściciela klucza osobiście, może to być np. sprawdzenie ódcisku klucza"przez telefon. W innych przypadkach pozostaje albo osobiste spotkanie z właścicielem klucza, albo zaufanie do sygnatur już się w tym kluczu znajdujących. Kiedy już weszliśmy w posiadanie klucza publicznego innej osoby, aby móc z niego korzystać należy dołączyć go do swojego kółka z kluczami (***keyring***). Kółko z kluczami to nic innego, jak plik zawierający wiele kluczy w odpowiednim formacie zazwyczaj będzie to to samo kółko, na którym znajduje się klucz publiczny właściciela.

Sieć zaufania (ang. web of trust) zdecentralizowana metoda uwierzytelniania osób, w której nie ma hierarchicznej struktury organizacji uwierzytelniających, a zaufanie do poszczególnych certyfikatów jest sumą podpisów złożonych przez innych uczestników sieci. Każdy uczestnik sieci podpisuje klucze osób, które osobiście zweryfikował. Podpis stanowi poświadczenie, że osoba podpisująca jest przekonana o autentyczności klucza oraz tym, że faktycznie należy on do osoby, która deklaruje że jest jego właścicielem. Jeśli ktoś z uczestników sieci chce sprawdzić, czy pewien klucz rzeczywiście należy do danej osoby, próbuje ułożyć łańcuch zaufania w postaci: "Ja zweryfikowałem, że X to X i podpisałem jego

klucz KX, X zweryfikował, że Y to Y i podpisał jego klucz KY, itd., Z zweryfikował, że sprawdzany klucz rzeczywiście należy do tej osoby."

Zadania

Przykładowe maile, których można użyć w zadaniach: `student1.bsk@wp.pl`, `student2.bsk@wp.pl` *hasło:* `testowehaslo2021`.

4.1 Wykorzystując oprogramowanie `gpg`, wygeneruj zestaw kluczy (publiczny-prywatny) dla dowolnego użytkownika przy pomocy algorytmów RSA i DSA. (Pamiętaj, że algorytmu RSA możemy używać do szyfrowania/podpisywania, natomiast DSA do podpisywania.)

4.2 Wykorzystując oprogramowanie `gpg`, wygeneruj zestaw kluczy (publiczny-prywatny) dla **użytkownika BSK** (użyj przykładowych maili) przy pomocy algorytmu DSA. (Niech będzie to klucz bez określonego terminu ważności.)

4.3 Zmień datę wygaśnięcia klucza utworzonego w zad **4.2** na za 3 miesiące od dziś.

4.4 Wyświetl listę posiadanych kluczy, zarówno publicznych jak i prywatnych (czyli swój keyring, inaczej nazywany repozytorium kluczy programu `gpg`).

4.5 Usuń klucz użytkownika BSK (utworzony w zadaniu **4.2**) ze swojego repozytorium kluczy.

4.6 Wyeksportuj swój klucz publiczny do pliku w postaci tekstowej. Podejrzyj informacje o kluczu.

4.7 Wyeksportuj swój klucz publiczny do pliku w postaci binarnej. Podejrzyj informacje o kluczu.

4.8 Z serwera kluczy `hkp://keys.openpgp.org` zaimportuj klucz o ID `4DFA270A`. Sprawdź ponownie swój keyring. Podpisz zaimportowany klucz. (*Poprzez podpisanie klucza innej osoby naszym kluczem zatwierdzamy jego autentyczność.*)

4.9 Wygeneruj losowy plik o wielkości 1MB korzystając z pliku `/dev/urandom`. Za pomocą `gpg` zaszyfruj plik przy pomocy szyfrowania symetrycznego i algorytmu AES-256. Wynik zapisz do pliku w postaci tekstowej. (*Oprogramowanie* `gpg` *umożliwia wykonanie szyfrowania symetrycznego, jak również asymetrycznego. W przypadku szyfrowania symetrycznego klucz szyfrujący/deszyfrujący jest generowany z podanego hasła.*)

4.10 Za pomocą `gpg` zaszyfrowano pewien plik przy pomocy algorytmu CAMELLIA128. Wynik szyfrowania zapisano do pliku `enc4.10.txt`. Hasło wykorzystane podczas szyfrowania to `test#2021`. Odszyfruj plik `enc4.10.txt`, wynik deszyfrowania zapisz do pliku `dec4.10.txt`.

Oprogramowanie `gpg` umożliwia wykonanie szyfrowania symetrycznego, jak również asymetrycznego. W przypadku szyfrowania symetrycznego klucz szyfrujący/deszyfrujący jest generowany z podanego hasła.)

4.11 Wygeneruj 2 pary kluczy: dla użytkownika Alice (`alice.bsk@wp.pl`) i Bob (`bob.bsk@wp.pl`). Zakładając, że Bob chce wysłać zaszyfrowaną wiadomość do Alice, przygotuj plik `hello.txt` z przykładową wiadomością, zaszyfruj go (jako Bob), a następnie odszyfruj (jako Alice). Zaszyfrowany tekst powinien być zakodowany Base64.

Oprogramowanie `gpg` umożliwia wykonanie szyfrowania symetrycznego, jak również asymetrycznego. W przypadku szyfrowania asymetrycznego wykorzystywana jest para kluczy. Aby dokonać szyfrowania wiadomości, należy zaimportować lub posiadać klucz publiczny osoby, której chcemy przesłać zaszyfrowany plik. Wówczas jedynie ta osoba, przy pomocy swojego klucza prywatnego będzie w stanie odczytać wiadomość.

4.12 Ze strony kursu pobierz 2 pliki: `mallory.pub` (klucz publiczny użytkownika Mallory) oraz pliki `msg.sig` (podpisany przez niego plik) i `msg.txt`. Zweryfikuj podpis.

4.13 Użytkownicy systemów linuksowych mają możliwość pobrania oprogramowania z repozytoriów przygotowanych dla danej dystrybucji systemu. Niekiedy jednak dodatkowe oprogramowanie można pobrać jedynie ze strony WWW. W takim przypadku, jaka jest pewność, że plik znajdujący się na stronie, został na niej w rzeczywistości umieszczony przez dewelopera aplikacji, a nie hakera? Niektórzy programiści podpisują swoje oprogramowanie przy pomocy rozwiązań PGP (takich jak np. GPG). Dzięki temu, jako użytkownicy, mamy możliwość weryfikacji integralności oprogramowania. Proces weryfikacji jest prosty, należy:

(a) Pobrać klucz publiczny autora oprogramowania
(b) Sprawdzić fingerprint klucza
(c) Zaimportować klucz do własnego keyringa
(d) Pobrać sygnaturę dla ściąganego oprogramowania
(e) Użyć klucza publicznego do zweryfikowania podpisu

Pobierz oprogramowanie VeraCrypt i zweryfikuj jego integralność.

4.14 Pobierz najnowszą wersję serwera Apache i zweryfikuj integralność pobranego pliku. Potrzebny serwer kluczy to `pgpkeys.mit.edu`.

4.15 Podpisz klucz publiczny Apache dwoma kluczami z własnego keyringu.

Bardzo istotna jest pewność, że klucz publiczny, który właśnie w ten czy inny sposób pozyskaliśmy, należy naprawdę do osoby, do której wydaje się należeć. Zapewnieniu tego służą certyfikaty kluczy. Certyfikowanie (podpisywanie) czyjegoś klucza to nic innego, jak sygnowanie go swoim własnym - ma to na celu potwierdzenie jego autentyczności.

Jeśli użytkownik A otrzyma klucz publiczny użytkownika B bezpośrednio od niego, to zapewne jest to naprawdę klucz użytkownika B. Ale jeśli otrzymuje go od użytkownika C, któremu niekoniecznie ufa, i podejrzewa, że w rzeczywistości klucz ten może być sfałszowany przez C? Cóż, jeśli klucz ten jest certyfikowany przez pewnego innego użytkownika D (któremu A ufa, i którego klucz publiczny już ma), i sygnatura się zgadza, to A zyskuje pewność, że klucz przekazany przez C jest zgodny z oryginałem. Oczywiście A zakłada w tym momencie, że D w chwili certyfikowania tego klucza miał 100% pewności, że klucz ten należy do B - albo dlatego, że otrzymał go od B bezpośrednio, albo dlatego, że klucz był certyfikowany przez kolejną osobę, do której D miał zaufanie. Należy z tego wysnuć jeden wniosek: NIGDY nie certyfikuj cudzego klucza, jeśli nie masz pewności, że nie jest on fałszywy. W przeciwnym razie osoby, które następnie ten klucz od Ciebie otrzymają, będą sądzić, że taką pewność miałeś i używać (być może fałszywego) klucza z powodu zaufania do Ciebie.

Linki

- `https://home.agh.edu.pl/~szymon/artykuly/pgp_opis.html`
- `https://crypto.stackexchange.com/questions/2585/whycantdsabeusedforencryptions`
- `https://gnupg.org/gph/en/manual.pdf`
- `https://www.cs.stonybrook.edu/sites/default/files/PGP70IntroToCrypto.pdf`
- `https://nordlocker.com/blog/rsavsdsa/`
- `https://superuser.com/questions/655246/are-gnupg-1-and-gnupg-2-compatible-with-each-other`

Odpowiedzi

4.1 Instalacja: `sudo apt-get install gnupg`

Generowanie kluczy:
`gpg --gen-key`
`gpg --full-generate-key`

Sprawdzenie zbioru kluczy:
`gpg --list-keys`

4.2 Generowanie kluczy:
`gpg --full-generate-key`

4.3 Sprawdzenie kluczy:

```
gpg --list-keys

/home/kmazur/.gnupg/pubring.kbx

pub   rsa3072 20211003 [SC] [wygasa: 20231003]
      06B7F4920EA14F614EDC9683D69A2F183F8E282D
uid    [   absolutne   ] Katarzyna Mazur <katarzyna.mazur@umcs.pl>
sub   rsa3072 20211003 [E] [wygasa: 20231003]

pub   rsa3072 20211003 [SC]
      0E66D0C1509D737E689924FEE82CBDFD72E5B533
uid    [   absolutne   ] Katarzyna Mazur <kasiula.mazur@gmail.com>
sub   rsa3072 20211003 [E]
```

Edycja klucza (polecenia po kolei):

```
gpg --edit-key 06B7F4920EA14F614EDC9683D69A2F183F8E282D
key 0
expire
3m
```

4.4 Wyświetlenie kluczy:
```
gpg --list-keys
gpg --list-secret-keys
```

4.5 Wyświetlenie kluczy:
```
gpg --list-keys
gpg --list-secret-keys
```

Wyświetlenie kluczy w innym formacie:
```
gpg --list-keys --keyid-format SHORT
gpg --list-secret-keys --keyid-format SHORT
```

Usunięcie kluczy:
```
gpg --delete-key 5DC93E58
gpg --delete-secret-key 5DC93E58
```

4.6
```
gpg --output kmazur.key --armour --export katarzyna.mazur@umcs.pl
gpg --show-keys kmazur.key
gpg --show-keys --keyid-format SHORT kmazur.key
```

4.7
```
gpg --output kmazurbin.key --export katarzyna.mazur@umcs.pl
gpg --show-keys kmazur.key
gpg --show-keys --keyid-format SHORT kmazur.key
```

4.8 Klucz wysłano na serwer za pomocą:
```
gpg --keyserver hkp://keys.openpgp.org --send-keys 4DFA270A
```

```
gpg --keyserver hkp://keys.openpgp.org --recv 4DFA270A
gpg --list-keys --keyid-format SHORT
```

4.9 Sprawdzenie listy dostępnych algorytmów:
```
gpg --version
```

Losowy plik:
```
head -c 1048576 < /dev/urandom | base64 > plain.txt
```

Szyfrowanie symetryczne AES-256:
`gpg --symmetric --armor --cipher-algo AES256 --output enc.txt plain.txt`

4.10 Plik utworzono przy pomocy polecenia:
`gpg --symmetric --armor --cipher-algo CAMELLIA128 --output enc4.10.txt ex4.10.txt`

Odpowiedź (deszyfrowanie):
`gpg --decrypt --output dec4.10.txt enc4.10.txt`

4.11 Generowanie kluczy:
`gpg --gen-key`
`gpg --full-generate-key`

Sprawdzenie ID klucza Alice:

```
(kmazur@kali)-[~/Gitz/bsk_2021_umcs/Lab4/all-files]
$ gpg --list-keys --keyid-format SHORT
/home/kmazur/.gnupg/pubring.kbx
-------------------------------
pub   rsa3072/B9F2A4E2 2021-10-04 [SC]
      3A35F58B9879DADA646BAAF44FDA17ECB9F2A4E2
uid  [   absolutne   ] Bob UMCS <bob.bsk@wp.pl>
sub   rsa3072/00E29EFC 2021-10-04 [E]

pub   rsa3072/6BC50406 2021-10-04 [SC]
      275A440229261D065F3688B8F3DDB1296BC50406
uid  [   absolutne   ] Alice UMCS <alice.bsk@wp.pl>
sub   rsa3072/1A35C3AA 2021-10-04 [E]
```

Szyfrowanie wiadomości do Alice (kluczem publicznym Alice):
`gpg --encrypt --recipient 6BC50406 --armour --output hello.enc hello.txt`

Odszyfrowanie wiadomości przez Alice (kluczem prywatnym Alice):
`gpg --decrypt --output hello.dec hello.enc`

4.12 Eksport klucza publicznego do pliku:
`gpg --output mallory.pub --armour --export mallory.bsk@wp.pl`
Eksport klucza prywatnego do pliku:
`ppg --export-secret-keys --armor mallory.bsk@wp.pl > mallory.priv`

4.13 `wget https://www.idrix.fr/VeraCrypt/VeraCrypt_PGP_public_key.asc`
`gpg --show-keys VeraCrypt_PGP_public_key.asc`
`gpg --import VeraCrypt_PGP_public_key.asc`

```
wget https://launchpad.net/veracrypt/trunk/1.24-update7/+download/veracrypt-1.24-Update7-Ubuntu-21.04-amd64.deb
wget https://launchpad.net/veracrypt/trunk/1.24-update7/+download/veracrypt-1.24-Update7-Ubuntu-21.04-amd64.deb.sig
gpg --verify veracrypt-1.24-Update7-Ubuntu-21.04-amd64.deb.sig veracrypt-1.24-Update7-Ubuntu-21.04-amd64.deb
```

4.14
```
gpg --verify httpd-2.4.49.tar.bz2.asc httpd-2.4.49.tar.bz2
gpg --keyserver pgpkeys.mit.edu --recv-key 26F51EF9A82F4ACB43F1903ED377C9E7D1944C66
gpg --verify httpd-2.4.49.tar.bz2.asc httpd-2.4.49.tar.bz2
```

4.15
```
gpg --check-sigs
gpg --default-key CB08B727B9DFE15BAFFCEDC2DF0C1370EAAD9477 --sign-key 26F51EF9A82F4ACB43F1903ED377C9E7D1944C66
```

Listingi

Usuwanie klucza:

```
(kmazur@kali)[~]
$ gpg --list-keys --keyid-format SHORT
gpg: sprawdzanie bazy zaufania
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: poziom: 0 poprawnych:   2 podpisanych:   0 zaufanie: 0,0q,0n,0m,0f,2u
gpg: następne sprawdzanie bazy odbędzie się 20220101
/home/kmazur/.gnupg/pubring.kbx

pub   rsa3072/3F8E282D 20211003 [SC] [wygasa: 20220101]
      06B7F4920EA14F614EDC9683D69A2F183F8E282D
uid  [   absolutne   ] Katarzyna Mazur <katarzyna.mazur@umcs.pl>
sub   rsa3072/41443AE2 20211003 [E] [wygasa: 20231003]

pub   rsa3072/14B89878 20211004 [SC]
      D2FC26E28139CAA800FBD2B263D3DBF914B89878
uid  [   absolutne   ] Student BSK <bsk@wp.pl>
sub   rsa3072/83A9F833 20211004 [E]

(kmazur@kali)[~]
$ gpg --delete-secret-key 14B89878
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

sec  rsa3072/63D3DBF914B89878 20211004 Student BSK <bsk@wp.pl>

Usunąć ten klucz ze zbioru? (t/N) t
To jest klucz tajny!  czy na pewno go usunąć? (t/N) t

(kmazur@kali)[~]
$ gpg --delete-key 14B89878
gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

pub  rsa3072/63D3DBF914B89878 20211004 Student BSK <bsk@wp.pl>

Usunąć ten klucz ze zbioru? (t/N) t

(kmazur@kali)[~]
$ gpg --list-keys --keyid-format SHORT
gpg: sprawdzanie bazy zaufania
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: poziom: 0 poprawnych:   1 podpisanych:   0 zaufanie: 0,0q,0n,0m,0f,1u
gpg: następne sprawdzanie bazy odbędzie się 20220101
/home/kmazur/.gnupg/pubring.kbx

pub   rsa3072/3F8E282D 20211003 [SC] [wygasa: 20220101]
      06B7F4920EA14F614EDC9683D69A2F183F8E282D
uid  [   absolutne   ] Katarzyna Mazur <katarzyna.mazur@umcs.pl>
sub   rsa3072/41443AE2 20211003 [E] [wygasa: 20231003]

(kmazur@kali)[~]
```

Edycja klucza:

```
gpg --edit-key 06B7F4920EA14F614EDC9683D69A2F183F8E282D

gpg (GnuPG) 2.2.27; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Dostępny jest klucz tajny.

sec  rsa3072/D69A2F183F8E282D
     utworzono: 20211003  wygasa: 20231003  użycie: SC
     zaufanie: absolutne     poprawność: absolutne
ssb  rsa3072/2FF49B1D41443AE2
     utworzono: 20211003  wygasa: 20231003  użycie: E
[   absolutne   ] (1). Katarzyna Mazur <katarzyna.mazur@umcs.pl>

gpg> key 0

sec  rsa3072/D69A2F183F8E282D
     utworzono: 20211003  wygasa: 20231003  użycie: SC
     zaufanie: absolutne     poprawność: absolutne
ssb  rsa3072/2FF49B1D41443AE2
     utworzono: 20211003  wygasa: 20231003  użycie: E
[   absolutne   ] (1). Katarzyna Mazur <katarzyna.mazur@umcs.pl>

gpg> expire
Zmiana daty ważności głównego klucza.
Okres ważności klucza.
         0 = klucz nie ma określonego terminu ważności
      <n>  = termin ważności klucza upływa za n dni
      <n>w = termin ważności klucza upływa za n tygodni
      <n>m = termin ważności klucza upływa za n miesięcy
      <n>y = termin ważności klucza upływa za n lat
Okres ważności klucza? (0) 3m
Klucz traci ważność sob, 1 sty 2022, 11:23:27 CET
Czy wszystko się zgadza (t/N)? t

sec  rsa3072/D69A2F183F8E282D
     utworzono: 20211003  wygasa: 20220101  użycie: SC
     zaufanie: absolutne     poprawność: absolutne
ssb  rsa3072/2FF49B1D41443AE2
     utworzono: 20211003  wygasa: 20231003  użycie: E
[   absolutne   ] (1). Katarzyna Mazur <katarzyna.mazur@umcs.pl>

gpg> q
Zapisać zmiany? (t/N) t
```

Weryfikacja integralności oprogramowania:

```
(kmazur@kali)[~/Gitz/bsk_2021_umcs/Lab4/all-files]
$ gpg --show-keys VeraCrypt_PGP_public_key.asc
pub   rsa4096 2018-09-11 [SC]
      5069A233D55A0EEB174A5FC3821ACD02680D16DE
uid                      VeraCrypt Team (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr>
sub   rsa4096 2018-09-11 [E]
sub   rsa4096 2018-09-11 [A]

(kmazur@kali)[~/Gitz/bsk_2021_umcs/Lab4/all-files]
gpg --import VeraCrypt_PGP_public_key.asc
gpg: klucz 821ACD02680D16DE: 1 podpis nie został sprawdzony z powodu braku klucza
gpg: klucz 821ACD02680D16DE: klucz publiczny ,,VeraCrypt Team
     (2018 - Supersedes Key ID=0x54DDD393) <veracrypt@idrix.fr>'' wczytano do zbioru
gpg: Ogółem przetworzonych kluczy: 1
gpg:          dołączono do zbioru: 1
gpg: brak absolutnie zaufanych kluczy

(kmazur@kali)[~/Gitz/bsk_2021_umcs/Lab4/all-files]$
```