

# Devoir 3

INFO4305

Alec Jones  
A00216262

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Objectif du TP</b>	<b>2</b>
<b>3</b>	<b>Déroulement du TP</b>	<b>3</b>
3.1	Partie 1 . . . . .	3
3.2	Partie 2 . . . . .	3
3.2.1	Partie a . . . . .	3
3.2.2	Partie b . . . . .	4
3.3	Partie 3 . . . . .	4
3.4	Partie 4 . . . . .	4
3.5	Partie 5 . . . . .	5
3.6	Partie 6 . . . . .	6
3.7	Partie 7 . . . . .	6
3.8	Partie 8 . . . . .	6
3.9	Partie 9 . . . . .	7
3.9.1	Partie a . . . . .	7
3.9.2	Partie b - Communication avec un camarade de classe . . . . .	7
3.10	Partie 10 . . . . .	9
3.11	Partie 11 . . . . .	10
3.12	Partie 12 . . . . .	11
3.13	Partie 13 . . . . .	11
<b>4</b>	<b>Observation, interprétation et conclusion</b>	<b>13</b>
<b>5</b>	<b>Sources</b>	<b>13</b>

# 1 Introduction

La cryptographie joue un rôle fondamental dans la sécurisation des communications numériques. Dans ce document, nous explorons brièvement comment générer des clés, chiffrer des messages et vérifier des signatures électroniques, afin de garantir l'intégrité et la confidentialité des échanges.

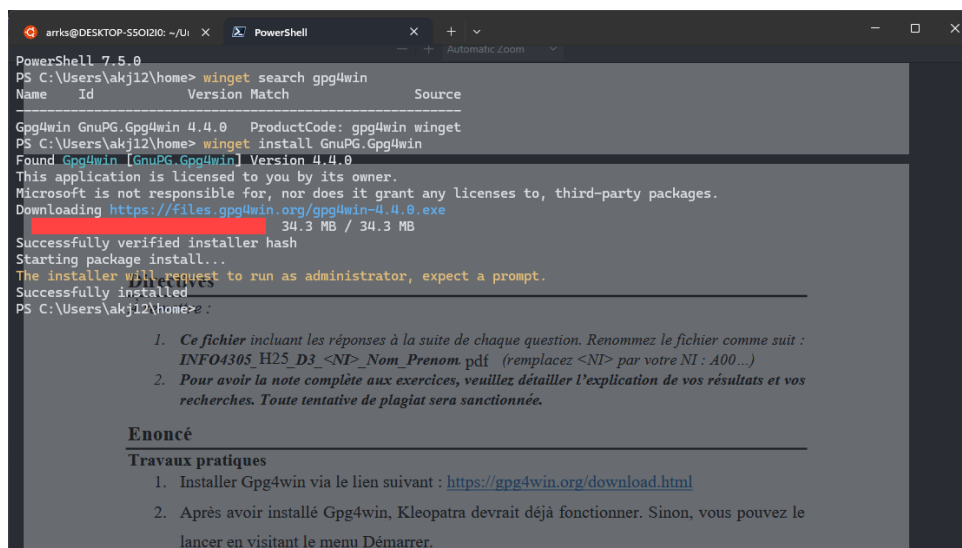
## 2 Objectif du TP

L'objectif de ce TP est de se familiariser avec la cryptographie à clé publique, en particulier avec l'utilisation de GPG (GNU Privacy Guard) pour la gestion des clés et le chiffrement des messages. Nous allons également aborder la création de certificats à l'aide d'OpenSSL et la signature de commits Git avec notre clé GPG.

## 3 D roulement du TP

### 3.1 Partie 1

Premi rement on doit d'abord installer Gpg4win, j'ai install  le logiciel   l'aide du cadre de paquets WinGet (voir la figure 1).



```
PowerShell 7.5.0
PS C:\Users\akj12\home> winget search gpg4win
Name      Id          Version Match          Source
-----
Gpg4win   GnuPG.Gpg4win 4.4.0   ProductCode: gpg4win winget
PS C:\Users\akj12\home> winget install GnuPG.Gpg4win
Found Gpg4win [GnuPG.Gpg4win] Version 4.4.0
This application is licensed to you by its owner.
Microsoft is not responsible for, nor does it grant any licenses to, third-party packages.
Downloading https://files.gpg4win.org/gpg4win-4.4.0.exe 34.3 MB / 34.3 MB
Successfully verified installer hash
Starting package install...
The installer will request to run as administrator, expect a prompt.
Successfully installed
PS C:\Users\akj12\home>
```

**Travaux pratiques**

1. Ce fichier incluant les r ponses   la suite de chaque question. Renommez le fichier comme suit : **INFO4305\_H25\_D3\_<NI>\_Nom\_Prenom.pdf** (remplacez <NI> par votre NI : A00...)
2. Pour avoir la note compl te aux exercices, veuillez d tailler l'explication de vos r sultats et vos recherches. Toute tentative de plagiat sera sanctionn e.

**Enonc **

**Travaux pratiques**

1. Installer Gpg4win via le lien suivant : <https://gpg4win.org/download.html>
2. Apr s avoir install  Gpg4win, Kleopatra devrait d j  fonctionner. Sinon, vous pouvez le lancer en visitant le menu D marrer.

Figure 1: Installation de Gpg4win

### 3.2 Partie 2

#### 3.2.1 Partie a

Pour cr er une paire de cl s   l'aide de l'interface graphique, on doit ouvrir le logiciel Kleopatra, ensuite on s lectionne l'option new key pair (voir la figure 2).

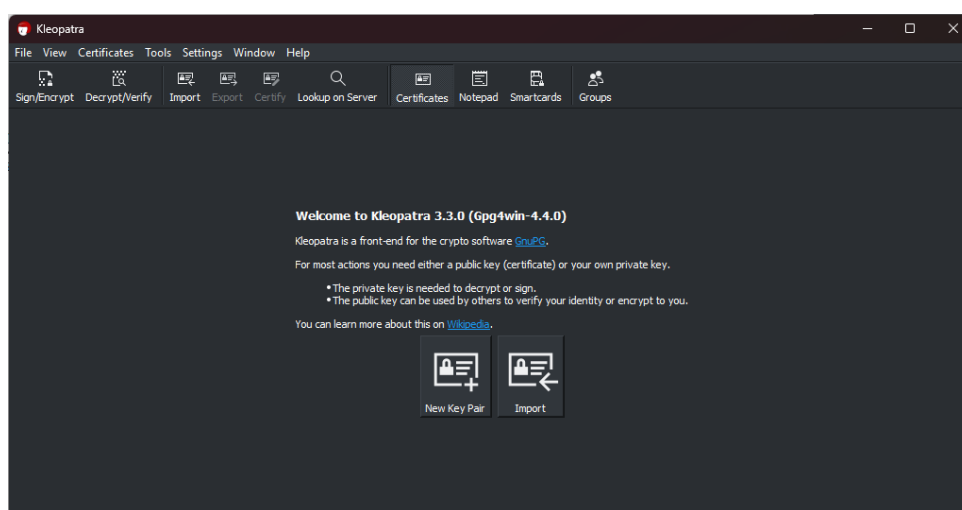


Figure 2: Menu initial Kleopatra

Ensuite, on doit s lectionner options avanc es et puis rsa2048. On doit aussi remplir notre nom et notre courriel pour le certificat (voir 3).

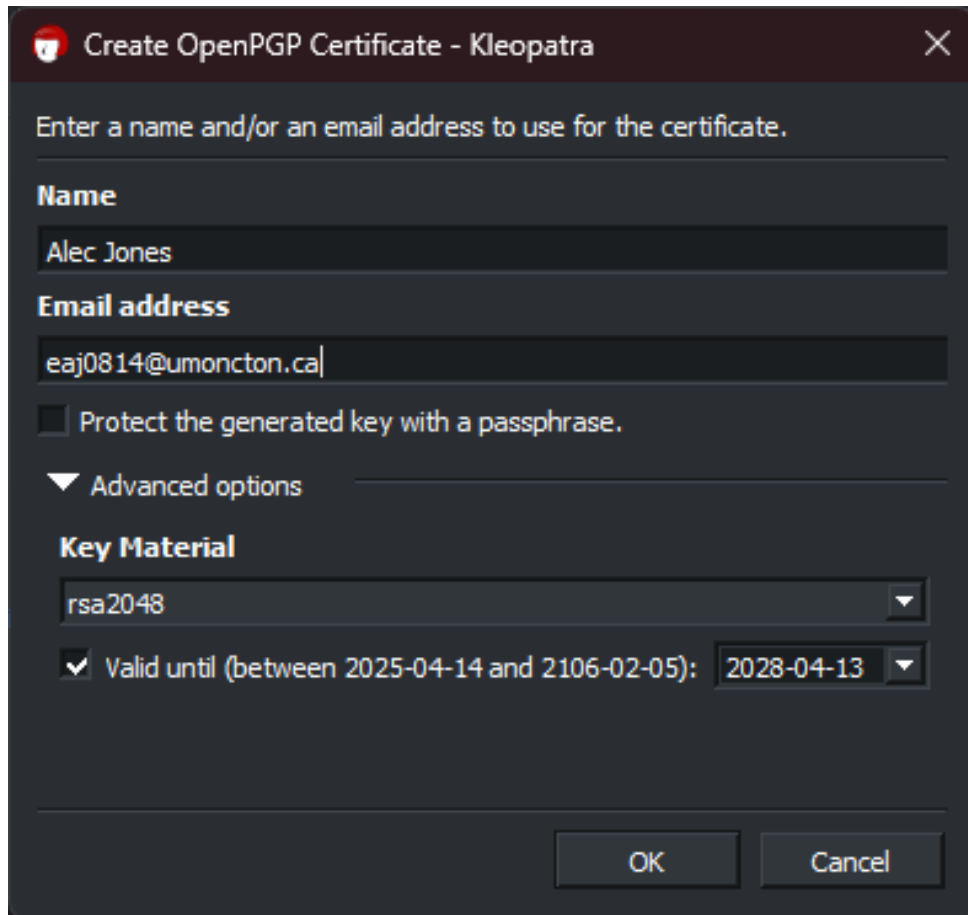


Figure 3: Création de clés dans Kleopatra

### 3.2.2 Partie b

Pour créer une paire de clés en ligne de commande, vous pouvez utiliser la commande suivante dans un terminal :

```
gpg --full-generate-key
```

Cette commande lance un assistant interactif vous permettant de choisir le type de clé, la longueur (par exemple, rsa2048 ou rsa4096) et de renseigner les informations nécessaires (nom, adresse électronique, etc.). Une fois terminée, votre paire de clés sera générée et stockée dans votre trousseau GPG (voir figure 4).

## 3.3 Partie 3

Pour lister notre trousseau de clés, on peut utiliser la commande suivante:

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

En exécutant la commande, on aperçoit les deux clés créées précédemment (voir figure 5).

## 3.4 Partie 4

Pour exporter les clés publiques, on utilise la commande:



## 3.6 Partie 6

Si on voulait ensuite déchiffrer ce texte, on utiliserait la commande suivant :

```
gpg --output doc --decrypt doc.gpg
```

## 3.7 Partie 7

Pour signer un document et laisser le texte en clair, on utilise la commande suivante (voir la figure 7 pour exemple de résultat):

```
gpg --clearsign document.txt
```

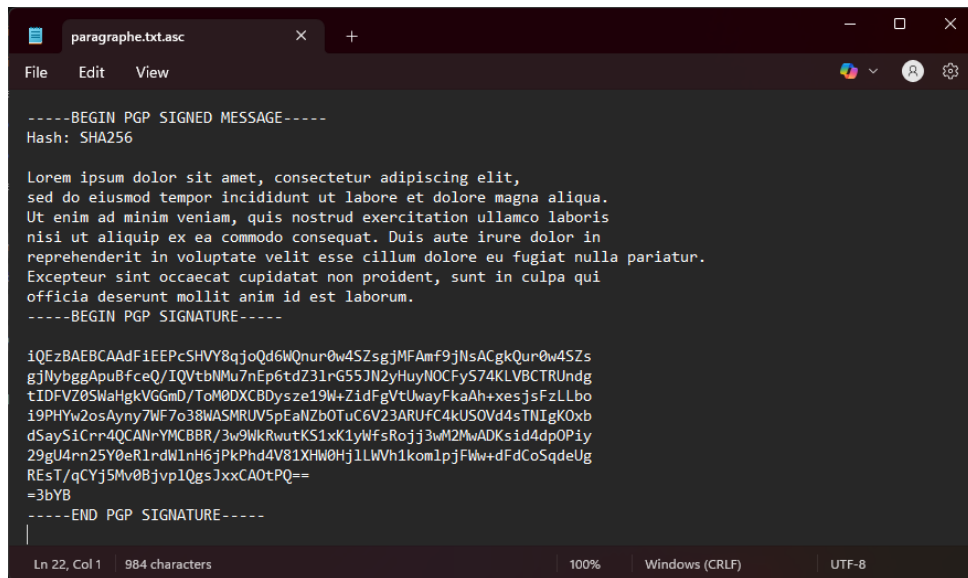


Figure 7: Document signé

## 3.8 Partie 8

Pour vérifier la signature, on utilise la commande suivante (voir figure 8):

```
gpg --verify document.txt.asc
```

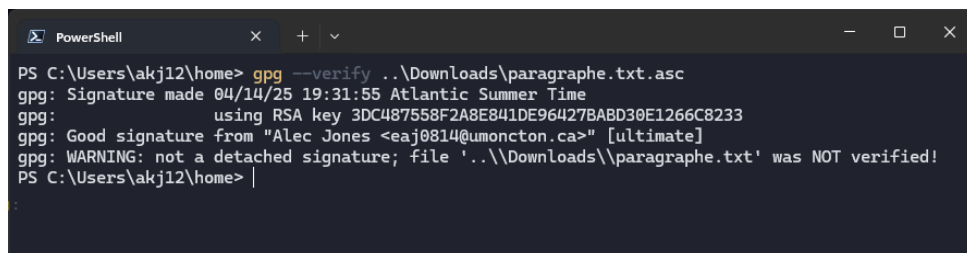
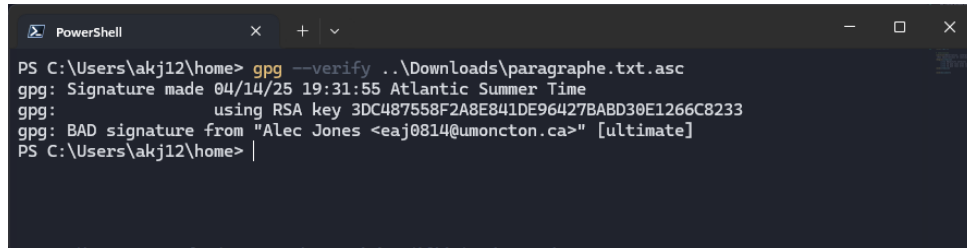


Figure 8: Vérification de la signature

## 3.9 Partie 9

### 3.9.1 Partie a

L'utilité de la signature dans ce contexte est de garantir l'intégrité du document et d'assurer que le document n'a pas été modifié depuis sa signature. En vérifiant la signature, on peut s'assurer que le document provient bien de la personne qui l'a signé et qu'il n'a pas été altéré. (voir figure 9 pour un exemple de document modifié).



```
PS C:\Users\akj12\home> gpg --verify ..\Downloads\paragraphe.txt.asc
gpg: Signature made 04/14/25 19:31:55 Atlantic Summer Time
gpg: using RSA key 3DC487558F2A8E841DE96427BABD30E1266C8233
gpg: BAD signature from "Alec Jones <eaj0814@umoncton.ca>" [ultimate]
PS C:\Users\akj12\home>
```

Figure 9: Vérification de la signature sur un document modifié, le premier mot a été enlevé

### 3.9.2 Partie b - Communication avec un camarade de classe

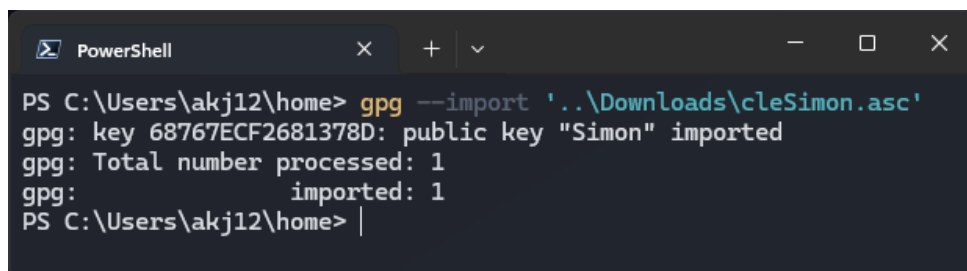
I Exporter votre clé publique. L'envoyer à un camarade de classe. J'ai réutilisé la commande suivante:

```
gpg --armor --output maclé.asc --export UserID
```

J'ai ensuite envoyé le fichier maclé.asc à mon camarade de classe par courriel.

II Récupérer la clé publique de votre camarade et l'importer dans votre base de clés. Pour ce faire j'ai utilisé la commande suivante (voir figure 10):

```
gpg --import maclé.asc
```



```
PS C:\Users\akj12\home> gpg --import '..\Downloads\cleSimon.asc'
gpg: key 68767ECF2681378D: public key "Simon" imported
gpg: Total number processed: 1
gpg: imported: 1
PS C:\Users\akj12\home>
```

Figure 10: Importation de la clé publique de mon camarade

III Visualiser votre base de clés. Pour ce faire j'ai utilisé la commande suivante:

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



```
PowerShell
PS C:\Users\akj12\home> gpg --list-keys
[keyboard]
-----
pub   rsa3072 2025-04-07 [SC]
      2E6580031C87AAC5218B524768767ECF2681378D
uid           [ unknown] Simon
sub   rsa3072 2025-04-07 [E]

pub   rsa2048 2025-04-13 [SC]
      3DC487558F2A8E841DE96427BABD30E1266C8233
uid           [ultimate] Alec Jones <ej0814@umoncton.ca>
sub   rsa2048 2025-04-13 [E]

pub   rsa2048 2025-04-13 [SC]
      77458201B3F3319E01585A7CCAA6B8AE5E694E5A
uid           [ultimate] Alec Jones <ej0814@umoncton.ca>
sub   rsa2048 2025-04-13 [E]

PS C:\Users\akj12\home> |
```

Figure 11: Liste des clés après importation de la clé publique de mon camarade

On remarque que la clé de mon camarade a bien été importée (voir figure 11).

- IV Chiffrer un message à destination d'un camarade et lui envoyer. Pour ce faire j'ai utilisé la commande suivante:

```
gpg -er UserID paragraphe.txt
```

J'ai ensuite envoyé le fichier paragraphe.txt.gpg à mon camarade de classe par Discord.

- V Déchiffrer un message reçu d'un camarade. Pour ce faire j'ai utilisé la commande suivante (voir figure 12):

```
gpg --decrypt doc.gpg
```

```
PowerShell
PS C:\Users\akj12\home> gpg --decrypt ..\Downloads\message.txt.gpg
gpg: encrypted with rsa2048 key, ID 973881E1A2666240, created 2025-04-13
      "Alec Jones <ej0814@umoncton.ca>"
oublie pas de remettre ton projet pour le cours de base de données avancée demain
PS C:\Users\akj12\home> |
```

Figure 12: Déchiffrement du message de mon camarade

- VI Calculer la signature électronique d'un message et l'expédier à un camarade. Pour ce faire j'ai utilisé la commande suivante:

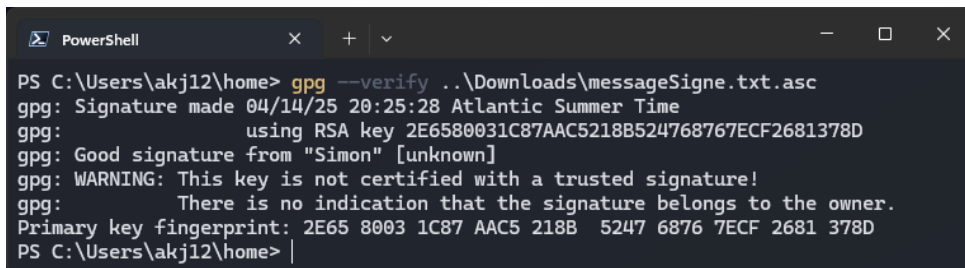
```
gpg --clearsign paragraphe.txt
```

J'ai ensuite envoyé le fichier `paragraphe.txt.asc` à mon camarade de classe par Discord.

- VII Récupérer un message signé d'un camarade et vérifier sa provenance/intégrité. Après avoir reçu le message signé de mon camarade, j'ai utilisé la commande suivante pour vérifier la signature:

```
gpg --verify paragraphe.txt.asc
```

On remarque que le message a bien été signé par mon camarade (voir figure 13).



```
PS C:\Users\akj12\home> gpg --verify ..\Downloads\messageSigne.txt.asc
gpg: Signature made 04/14/25 20:25:28 Atlantic Summer Time
gpg: using RSA key 2E6580031C87AAC5218B524768767ECF2681378D
gpg: Good signature from "Simon" [unknown]
gpg: WARNING: This key is not certified with a trusted signature!
gpg: There is no indication that the signature belongs to the owner.
Primary key fingerprint: 2E65 8003 1C87 AAC5 218B 5247 6876 7ECF 2681 378D
PS C:\Users\akj12\home> |
```

Figure 13: Vérification de la signature du message de mon camarade

### 3.10 Partie 10

Pour créer un certificat à l'aide de openssl, on utilisera la commande suivante:

```
openssl req -newkey rsa:2048 -keyout domain.key -out domain.csr
```

Décortiquons cette commande (voir figure 14 pour la sortie):

- `openssl req` : indique que nous voulons créer une demande de certificat.
- `-newkey rsa:2048` : crée une nouvelle clé RSA de 2048 bits.
- `-keyout domain.key` : Spécifie le fichier de sortie pour la clé privée.
- `-out domain.csr` : Spécifie le fichier de sortie pour la demande de signature de certificat (CSR).
- `-days 365` : Définit la durée de validité du certificat à 365 jours.
- `-sha256` : Utilise l'algorithme de hachage SHA-256 pour le certificat.

Cette commande génère deux fichiers :

- `domain.key` : La clé privée.
- `domain.csr` : La demande de signature de certificat (CSR).

Normalement, ce serait à une autorité de certification (CA) de signer le certificat, mais pour les besoins de ce TP, nous allons le signer nous-mêmes.

```

arrks@DESKTOP-550I2I0: ~/test$ openssl req -newkey rsa:2048 -keyout domain.key -out domain.csr
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CA
State or Province Name (full name) [Some-State]:New-Brunswick
Locality Name (eg, city) []:Moncton
Organization Name (eg, company) [Internet Widgits Pty Ltd]:Umoncton
Organizational Unit Name (eg, section) []:Info
Common Name (e.g. server FQDN or YOUR name) []:Alec Jones
Email Address []:eaj0814@umoncton.ca

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
arrks@DESKTOP-550I2I0:~/test$

```

Figure 14: Création d'un certificat avec OpenSSL

### 3.11 Partie 11

Pour signer le certificat avec la clé privée, on utilise la commande suivante:

```
openssl x509 -signkey domain.key -in domain.csr -req -days 365 -out domain.crt
```

Décortiquons cette commande (voir figure 15 pour la sortie):

- `openssl x509` : Indique que nous voulons travailler avec des certificats X.509.
- `-signkey domain.key` : Spécifie la clé privée à utiliser pour signer le certificat.
- `-in domain.csr` : Spécifie le fichier d'entrée (CSR) à signer.
- `-req` : indique que nous travaillons avec une demande de signature de certificat.
- `-days 365` : Définit la durée de validité du certificat à 365 jours.
- `-out domain.crt` : Spécifie le fichier de sortie pour le certificat signé.

Cette commande génère un fichier `domain.crt` qui contient le certificat signé. Ce certificat peut être utilisé pour établir des connexions sécurisées (SSL/TLS) sur un serveur web.

```

arrks@DESKTOP-550I2I0:~/test$ openssl x509 -signkey domain.key -in domain.csr -req -days 365 -out domain.crt
Enter pass phrase for domain.key:
Certificate request self-signature ok
subject=C = CA, ST = New-Brunswick, L = Moncton, O = Umoncton, OU = Info, CN = Alec Jones, emailAddress = eaj0814@umoncton.ca
arrks@DESKTOP-550I2I0:~/test$ ls
domain.crt  domain.csr  domain.key
arrks@DESKTOP-550I2I0:~/test$

```

Figure 15: Signature d'un certificat avec OpenSSL

## 3.12 Partie 12

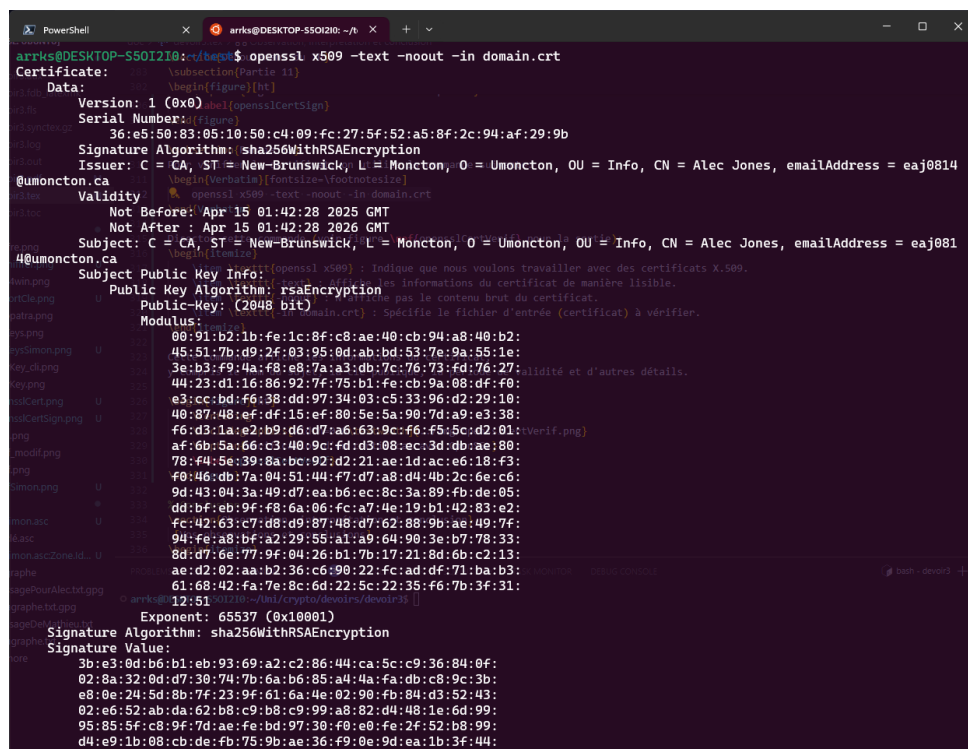
Pour vérifier le certificat, on utilise la commande suivante:

```
openssl x509 -texte -noout -in domain.crt
```

Décortiquons cette commande (voir figure 16 pour la sortie):

- `openssl x509` : indique que nous voulons travailler avec des certificats X.509.
- `-texte` : Affiche les informations du certificat de manière lisible.
- `-noout` : N'affiche pas le contenu brut du certificat.
- `-in domain.crt` : spécifie le fichier d'entrée (certificat) à vérifier.

Cette commande affiche les informations du certificat, y compris le nom du sujet, la clé publique, la période de validité et d'autres détails.



```
PowerShell
arrks@DESKTOP-S50I2I0: ~\temp$ openssl x509 -text -noout -in domain.crt
Certificate:
    \subsection{Partie 11}
    Data:
        \begin{figure}[ht]
        \begin{tikzpicture}
        \draw (0,0) node[anchor=north west] {\texttt{openssl x509}};
        \draw (0,1) node[anchor=north west] {\texttt{-text}};
        \draw (0,2) node[anchor=north west] {\texttt{-noout}};
        \draw (0,3) node[anchor=north west] {\texttt{-in domain.crt}};
        \end{tikzpicture}
        \end{figure}
    \end{tikzpicture}
    Version: 1 (0x0)
    Serial Number: 36:e5:50:83:05:10:50:c4:09:fc:27:5f:52:a5:8f:2c:94:af:29:9b
    Signature Algorithm: sha256WithRSAEncryption
    Issuer: C = CA, ST = New-Brunswick, L = Moncton, O = Umoncton, OU = Info, CN = Alec Jones, emailAddress = eaj0814@umoncton.ca
    Validity
        Not Before: Apr 15 01:42:28 2025 GMT
        Not After : Apr 15 01:42:28 2026 GMT
    Subject: C = CA, ST = New-Brunswick, L = Moncton, O = Umoncton, OU = Info, CN = Alec Jones, emailAddress = eaj0814@umoncton.ca
    Subject Public Key Info:
        Public Key Algorithm: rsaEncryption
        Public-Key: (2048 bit)
        Modulus:
            00:91:b2:1b:fe:1c:8f:c8:ae:40:cb:94:a8:40:b2:
            45:51:7b:d9:2f:03:95:0d:ab:bd:53:7e:9a:55:1e:
            3e:b3:f9:4a:f8:e8:7a:a3:db:7c:76:73:fd:76:27:
            44:23:d1:16:86:92:7f:75:b1:fe:cb:9a:08:df:f0:
            e3:cc:bd:f6:38:dd:97:34:03:c5:33:96:d2:29:10:
            40:87:48:ef:df:15:ef:80:5e:5a:90:7d:a9:e3:38:
            f6:d3:1a:e2:b9:d6:d7:a6:63:9c:f6:5f:5c:d2:01:
            af:6b:5a:66:c3:40:9c:fd:d3:08:ec:3d:db:ae:80:
            78:f4:5e:39:8a:bc:92:d2:21:ae:1d:ac:e6:18:f3:
            f0:46:db:7a:04:51:44:f7:d7:a8:d4:4b:2c:6e:c6:
            9d:43:04:3a:49:d7:ea:b6:ec:8c:3a:89:fb:de:05:
            dd:bf:eb:9f:f8:6a:06:fc:a7:4e:19:b1:42:83:e2:
            fc:42:63:c7:d8:d9:87:48:d7:62:88:9b:ae:49:7f:
            94:fe:a8:bf:a2:29:55:a1:a9:64:90:3e:b7:78:33:
            8d:d7:6e:77:9f:04:26:b1:7b:17:21:8d:6b:c2:13:
            ae:d2:02:aa:b2:36:c6:90:22:fc:ad:df:71:ba:b3:
            61:68:42:fa:7e:8c:6d:22:5c:22:35:f6:7b:3f:31:
            12:51:90:10:~\tmp\crypto\devoirs\devoir35 []
        Exponent: 65537 (0x10001)
        Signature Algorithm: sha256WithRSAEncryption
        Signature Value:
            3b:e3:0d:b6:b1:eb:93:69:a2:c2:86:44:ca:5c:c9:36:84:0f:
            02:8a:32:0d:d7:30:74:7b:6a:b6:85:a4:4a:fa:db:c8:9c:3b:
            e8:0e:24:5d:8b:7f:23:9f:61:6a:4e:02:90:fb:84:d3:52:43:
            02:e6:52:ab:da:62:b8:c9:b8:c9:99:a8:82:d4:48:1e:6d:99:
            95:85:3f:c8:9f:7d:ae:fe:bd:97:30:f0:e0:fe:2f:52:b8:99:
            d4:e9:1b:08:cb:de:fb:75:9b:ae:36:f9:0e:9d:ea:1b:3f:44:
```

Figure 16: Vérification d'un certificat avec OpenSSL

## 3.13 Partie 13

Git est un système de contrôle de version distribué qui permet de suivre les modifications apportées à des fichiers et de collaborer avec d'autres personnes. On est aussi capable de signer nos commits avec notre clé privée GPG. Pour ce faire, on doit d'abord configurer git pour utiliser notre clé GPG.

Premièrement, on doit lister nos clés GPG pour trouver l'ID de notre clé publique. Ensuite, on doit configurer git pour utiliser cette clé avec les commandes suivantes:

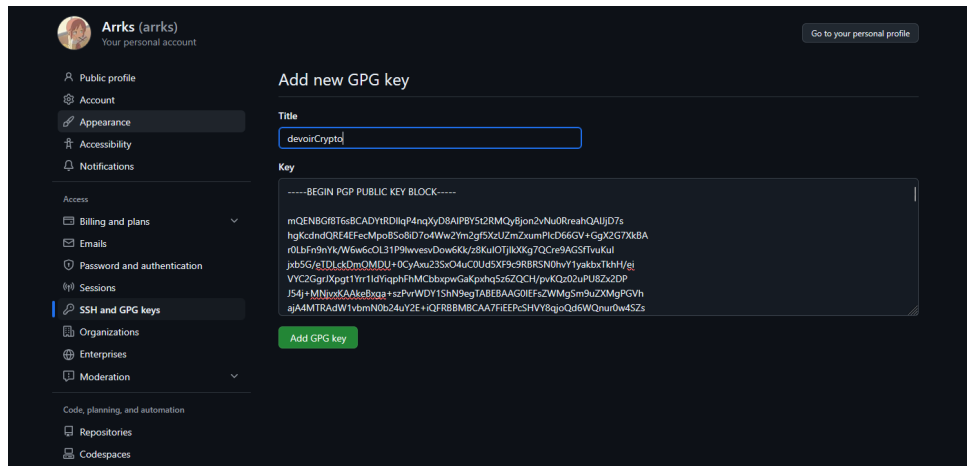


Figure 17: Ajout de la clé publique à GitHub

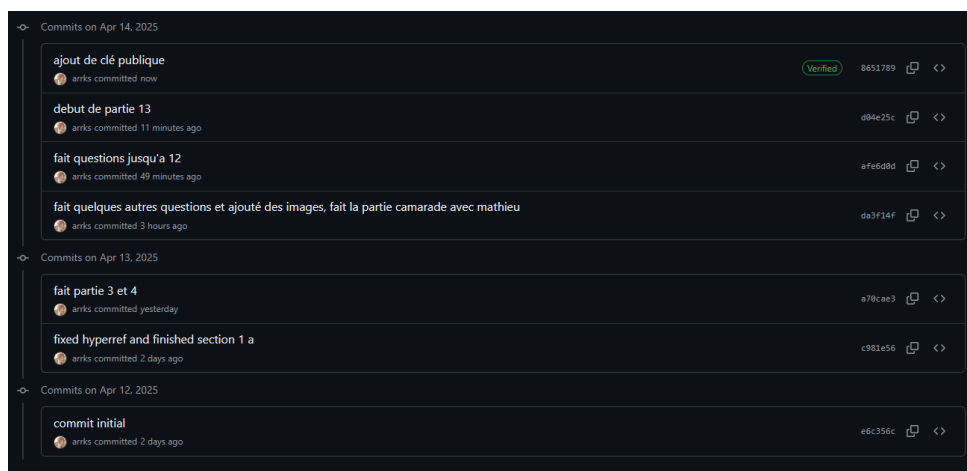


Figure 18: Vérification de la signature du commit sur GitHub

```
git config --global user.signingkey <ID de la clé>
git config --global commit.gpgsign true
```

Ensuite, on peut signer nos commits avec la commande suivante:

```
git commit -S -m "Message de commit"
```

Le -S indique à git de signer le commit avec notre clé GPG. On peut vérifier la signature du commit avec la commande suivante:

```
git log --show-signature
```

On devra aussi s'assurer que notre clé publique est disponible sur GitHub pour que les autres pouvant vérifier la signature de nos commits. Pour ajouter notre clé publique à GitHub, on doit se rendre dans les paramètres de notre compte, aller dans la section "SSH and GPG keys" et cliquer sur "New GPG key" (voir figure 17).

Après avoir effectué un push à GitHub, on peut vérifier que notre commit est bien signé en se rendant sur la page des commits sur GitHub (voir figure 18).

## 4 Observation, interprétation et conclusion

Dans ce TP, nous avons exploré les concepts fondamentaux de la cryptographie à clé publique, y compris la génération de clés, le chiffrement et le déchiffrement de messages, la signature électronique et la vérification de signatures. Nous avons également appris à utiliser GPG pour gérer nos clés et signer nos commits Git, ainsi qu'à créer et signer des certificats avec OpenSSL. Ces compétences sont essentielles pour garantir la sécurité et l'intégrité des communications numériques, en particulier dans le contexte de la collaboration en ligne et du développement de logiciels.

## 5 Sources

- GNU Privacy Guard (GPG): <https://gnupg.org/>
- OpenSSL: <https://www.openssl.org/>
- Documentation Git: <https://git-scm.com/doc>
- Aide pour certificats: <https://www.baeldung.com/openssl-self-signed-cert>