



BIRMINGHAM CITY
University

CMP5329 Logbook

DRAFT VERSION

Lewis Higgins - Student ID 22133848

January - March, 2024

Contents

Introduction	2
1 Lab 1 - OpenSSL	3
1.1 Version checking and ciphers	3
1.2 Symmetric encryption	4
1.2.1 DES symmetric encryption	4
1.2.2 AES256 symmetric encryption and decryption	5
1.3 Asymmetric encryption	6
1.3.1 Generating an RSA private key	6
1.3.2 Storing DES3 & passphrase encrypted RSA keys in a file	7
1.3.3 Getting a public key from the private key	8
1.3.4 Obtaining a message/file digest	9
1.3.5 Signing a digest	10
2 Lab 2 - Usage of GPG	12
2.1 Creating test users	12
2.1.1 Elevating the terminal	12
2.1.2 Creating Bob and Alice	12
2.2 Exchanging encrypted files over an insecure channel	15
2.2.1 Generating public/private key-pairs	15
2.2.2 Exporting public keys	16
2.2.3 Importing and signing public keys	16
2.2.4 Encrypting and decrypting data	18
5 Lab 5 - Discretionary Access Control	20
5.1 Creating test users and groups	20
5.1.1 Creating groups	20
5.1.2 Adding users to groups	20
5.2 Using chmod and chgrp to assign permissions	23
5.2.1 Mnemonic and octal chmod	23
5.2.2 Restricting directory access	23
5.2.3 Using chown	26
6 Lab 6 - Password Cracking	28
Conclusion	29

Introduction

This logbook documents the work completed and knowledge gained across the CMP5329 labs, showcasing the use of a wide variety of security techniques and access control methods on a Linux OS. This logbook specifically covers the following labs:

- Lab 1, covering OpenSSL.
- Lab 2, covering simple usage of GPG.
- Lab 5, covering the use of Linux Discretionary Access Control commands.
- Lab 6, covering password cracking.

As per module specifications, screenshots taken in each lab include the date and time at which they were taken.

Example note

Additional notes, such as minor issues encountered or omitted screenshots due to work having been done in earlier labs, are documented using these orange notes.

Example important note

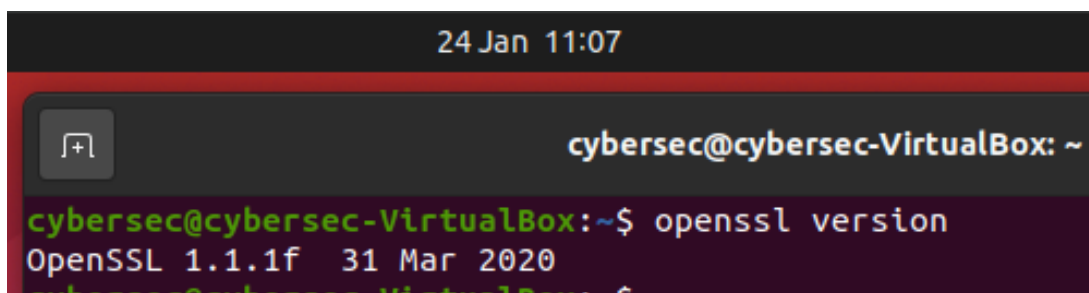
Critical issues that required special workarounds are documented using these red notes.

Lab 1 - OpenSSL

This lab was an introduction to the usage of a Linux (Ubuntu distro) VM on a host machine, and the usage of OpenSSL to encrypt and decrypt data using the (outdated) DES algorithm and AES256 symmetric encryption algorithm. Additionally, this lab looked at RSA private keys used in asymmetric encryption, and how to generate and gather public and private keys, alongside message digests.

1.1 Version checking and ciphers

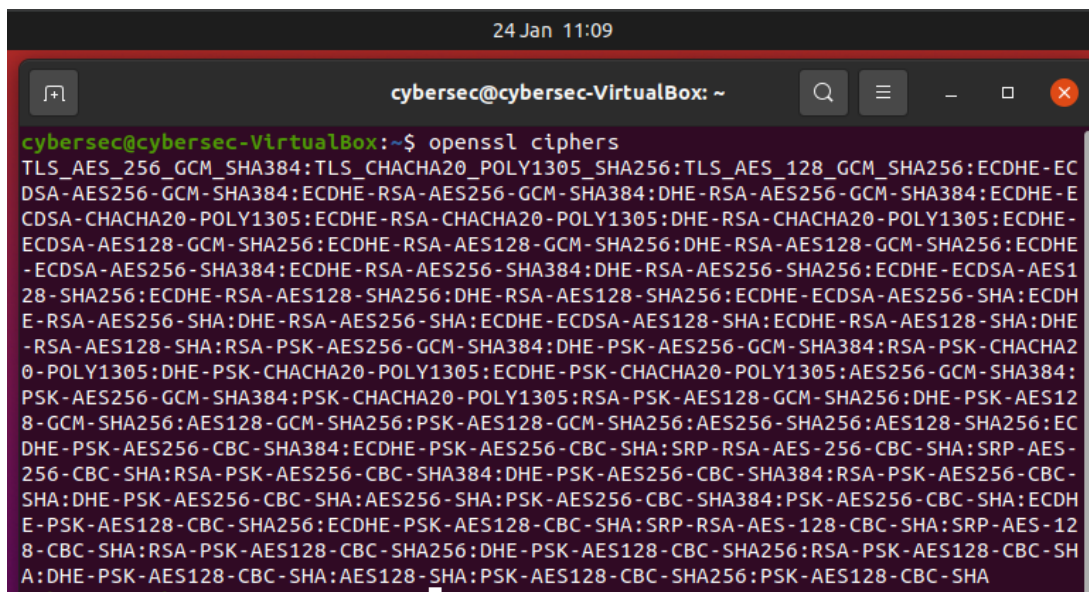
To check the installed version of OpenSSL, the command "openssl version" can be used. The provided virtual machine from [the CMP5329 Moodle page](#) uses OpenSSL version 1.1.1f, dated 31st March 2020.¹



```
24 Jan 11:07
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl version
OpenSSL 1.1.1f 31 Mar 2020
```

Figure 1.1: Getting the OpenSSL version

The list of OpenSSL ciphers can be viewed by executing the "openssl ciphers" command.



```
24 Jan 11:09
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl ciphers
TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:ECDHE-EC
DSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-E
CDSA-CHACHA20-POLY1305:ECDHE-RSA-CHACHA20-POLY1305:DHE-RSA-CHACHA20-POLY1305:ECDHE-
ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES128-GCM-SHA256:ECDHE
-ECDSA-AES256-SHA384:ECDHE-RSA-AES256-SHA384:DHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES1
28-SHA256:ECDHE-RSA-AES128-SHA256:DHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:EC DH
E-RSA-AES256-SHA:DHE-RSA-AES256-SHA:EC DH E-ECDSA-AES128-SHA:EC DH E-RSA-AES128-SHA:DHE
-RSA-AES128-SHA:RSA-PSK-AES256-GCM-SHA384:DHE-PSK-AES256-GCM-SHA384:RSA-PSK-CHACHA2
0-POLY1305:DHE-PSK-CHACHA20-POLY1305:EC DH E-PSK-CHACHA20-POLY1305:AES256-GCM-SHA384:
PSK-AES256-GCM-SHA384:PSK-CHACHA20-POLY1305:RSA-PSK-AES128-GCM-SHA256:DHE-PSK-AES12
8-GCM-SHA256:AES128-GCM-SHA256:PSK-AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:EC
DHE-PSK-AES256-CBC-SHA384:EC DH E-PSK-AES256-CBC-SHA:SRP-RSA-AES-256-CBC-SHA:SRP-AES-
256-CBC-SHA:RSA-PSK-AES256-CBC-SHA384:DHE-PSK-AES256-CBC-SHA384:RSA-PSK-AES256-CBC-
SHA:DHE-PSK-AES256-CBC-SHA:AES256-SHA:PSK-AES256-CBC-SHA384:PSK-AES256-CBC-SHA:EC DH
E-PSK-AES128-CBC-SHA256:EC DH E-PSK-AES128-CBC-SHA:SRP-RSA-AES-128-CBC-SHA:SRP-AES-12
8-CBC-SHA:RSA-PSK-AES128-CBC-SHA256:DHE-PSK-AES128-CBC-SHA256:RSA-PSK-AES128-CBC-SH
A:DHE-PSK-AES128-CBC-SHA:AES128-SHA:PSK-AES128-CBC-SHA256:PSK-AES128-CBC-SHA
```

Figure 1.2: Getting the OpenSSL ciphers

¹This is a heavily outdated version of OpenSSL, however I have continued to use it due to it being part of the module-provided resources.

1.2 Symmetric encryption

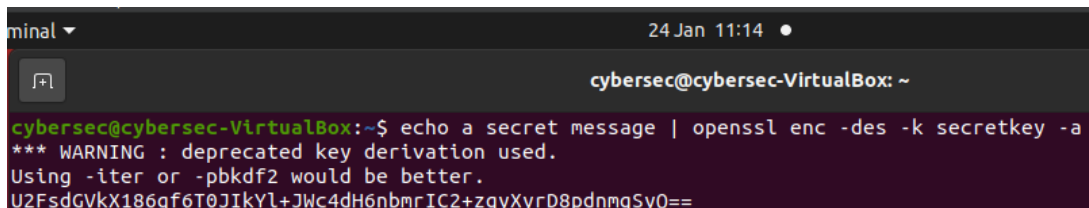
Symmetric cryptography refers to the process of transferring data that has been encrypted by a single key. Both the sender and receiver of this data use the same key to encrypt and decrypt the data. Symmetric encryption does not allow for non-repudiation, as it cannot necessarily be proven that data came from one sender and not someone else with the key, as there is no attached signature to prove the sender's identity.

1.2.1 DES symmetric encryption

OpenSSL can be used to encrypt plaintext into unreadable text known as ciphertext. Many algorithms exist to generate ciphertext, but for this example, the Data Encryption Standard (DES) symmetric encryption algorithm will be used. This is performed using the command

"openssl enc -des -k secretkey -a".² This command can be broken down into:

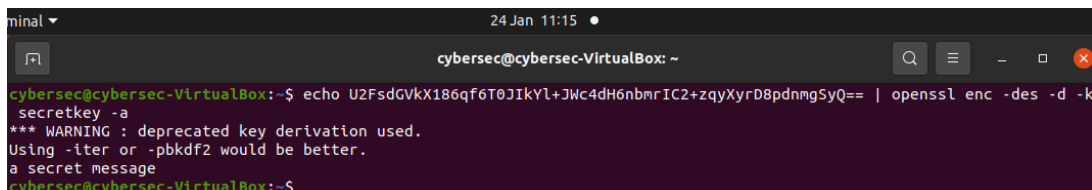
- enc - Encode
- -des - Using the DES algorithm
- -k secretkey - Using the key "secretkey"
- -a - Writes the ciphertext in base64/ASCII, allowing it to be properly displayed. If this flag isn't used, the text will not be legible, showing question mark symbols instead.



```
minal 24 Jan 11:14
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ echo a secret message | openssl enc -des -k secretkey -a
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
U2FsdGVkX186qf6T0JIKYL+JWc4dH6nbmrIC2+zqyXyrD8pdnmgSyQ==
```

Figure 1.3: Converting "a secret message" to ciphertext using DES with key "secretkey".

This ciphertext can then be decoded if you know the key it was encoded with.



```
minal 24 Jan 11:15
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ echo U2FsdGVkX186qf6T0JIKYL+JWc4dH6nbmrIC2+zqyXyrD8pdnmgSyQ== | openssl enc -des -d -k secretkey -a
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
a secret message
cybersec@cybersec-VirtualBox:~$
```

Figure 1.4: Decoding the ciphertext back to its original form using the key "secretkey".

This command is the same as in Figure 1.3, with two alterations:

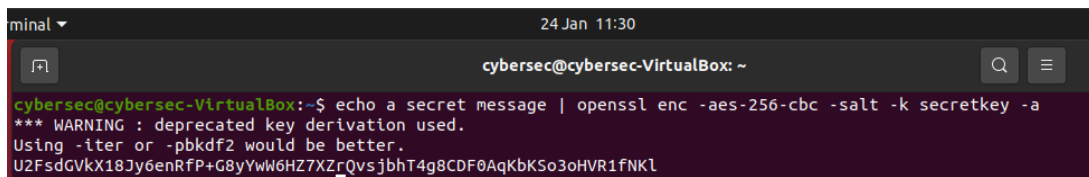
- The ciphertext is passed to OpenSSL rather than the plaintext, as we typically wouldn't know the original plaintext in this scenario, which is what we are aiming to achieve by decrypting it.
- -d - Indicates that the text should be decrypted rather than encrypted.

²"echo a secret message | " passes the phrase "a secret message" as the text to be encrypted.

1.2.2 AES256 symmetric encryption and decryption

The DES algorithm is considered weak to today's standards due to how simple it is to brute-force using today's processing power. Because of this, newer algorithms were developed, with one of these being AES. AES256 refers to the key size, which is 256 bits for this variant, though you can use key sizes of 128 or 192 bits as well, which would be weaker.

I researched how to use this algorithm in OpenSSL, eventually finding [this help page](#) (Heinlein, 2016) which provided details on encrypting text using the AES-256-cbc cipher.



```
minal 24 Jan 11:30
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ echo a secret message | openssl enc -aes-256-cbc -salt -k secretkey -a
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.
U2FsdGVkX18Jy6enRfP+G8yYwW6HZ7XZrQvsjbhT4g8CDF0AqKbKSo3oHVR1fNKl
```

Figure 1.5: Encoding the plaintext with AES-256-cbc using the key "secretkey".

In this command, the -des flag is instead replaced by -aes-256-cbc, indicating the cipher to use, and the optional -salt flag was added, which salts the text to provide different ciphertext. Salting is the process of adding random data to the end of the text prior to encoding it, which will change the resulting ciphertext, making it harder to decrypt and increasing the strength of the encryption.

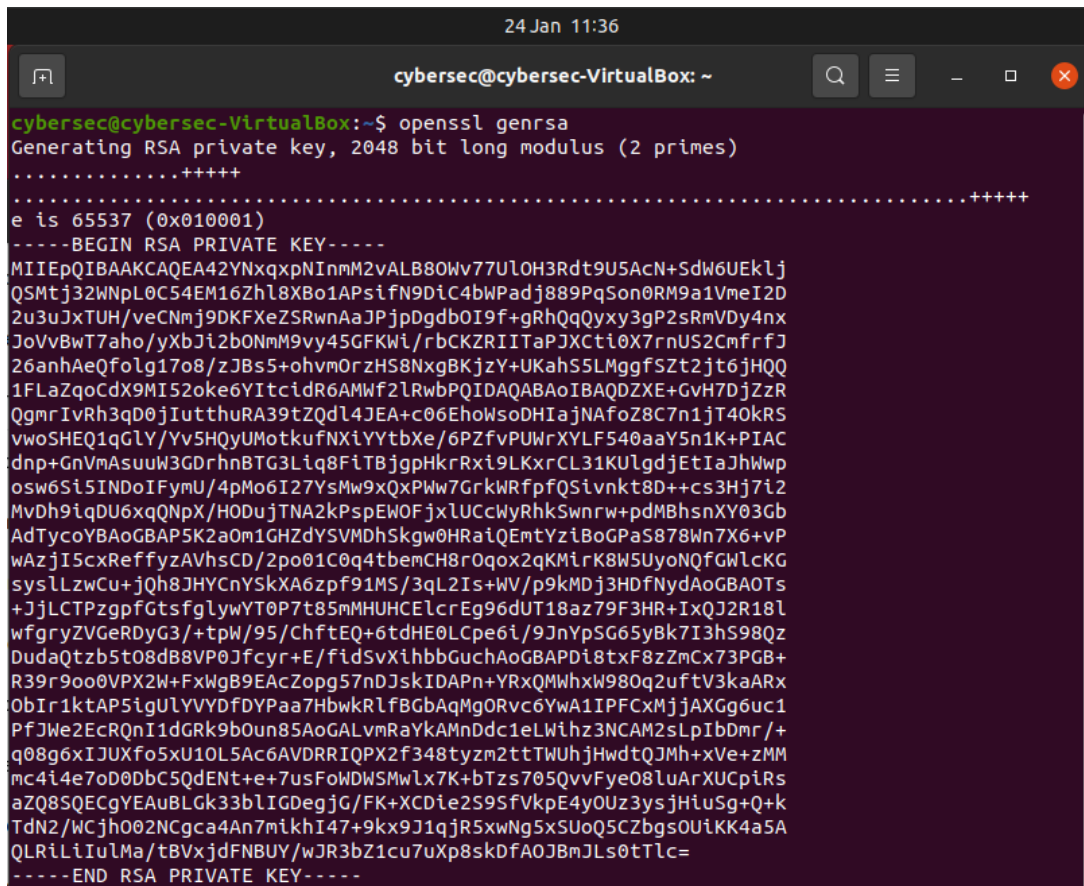
1.3 Asymmetric encryption

Asymmetric cryptography is the practice of using two keys when transmitting data: a public key used to encrypt data, and a private key used to decrypt it. For example: John is sending Alex an encrypted message. The message is encrypted using Alex's public key and sent to Alex. When Alex receives the message, it is decrypted using Alex's private key. Alex can prove that John sent this message because of the digital signature attached, which is generated from John's private key, and then verified using his public key. John does not ever know what Alex's private key is, nor does Alex know John's private key.

Data transferred this way has a digital signature attached, which allows for non-repudiation, as it cannot be denied that the data originated from the device with said signature. The signature is validated using the sender's private key.

1.3.1 Generating an RSA private key

OpenSSL can be used to generate these keys by using the "openssl genrsa" command.



```

24 Jan 11:36
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl genrsa
Generating RSA private key, 2048 bit long modulus (2 primes)
.....+++++
e is 65537 (0x010001)
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAA42YNxqxpNInmM2vAlB80Wv77U1OH3Rdt9U5AcN+SdW6UEklj
QSMtj32WNpL0C54EM16Zhl8XBo1APsifN9DiC4bwPadj889PqSon0RM9a1VmeI2D
2u3uJxTUH/veCNmj9DKFXeZSRwnAaJPjpDgdbOI9f+gRhQqQxy3gP2sRmVDy4nx
JoVvBwT7aho/yXbJi2bONmM9vy45GFKWi/rbCKZRITaPJXCTi0X7rnUS2CmfrfJ
26anhAeQfo1g17o8/zJBs5+ohvmOrzHS8NxbKjzY+UKahS5LMggf5Zt2jt6jHQQ
1FLaZqoCdX9MI52oke6YItcIdR6AMWf2lRwbPQIDAQABAOIBAQDZXE+GvH7DjZzR
QgmrIvRh3Qd0jIutthuRA39tZQdl4JEA+c06EhoWsoDHIAjNafoZ8C7n1jt40kRS
vwoSHEQ1qGLY/Yv5HQyUMotkufNXiYytbXe/6PZfVPUWrXYLF540aaY5n1K+PIAC
dnp+GnVmAsuuW3GDrhnBTG3Liq8FiTBjgphKrRxI9LKxrCL31KULgdjEtIaJhWwp
osw6Si5INDoIFymU/4pMo6I27YsMw9xQxPw7GrkWRfpfQSiVnkt8D++cs3Hj7i2
MvDh9iQDU6xqQNPx/HODujTNA2kPspEwOFjxLUCCwYRhKSwrnrw+pdMBhsnXY03Gb
AdTycoYBAoGBAP5K2aOm1GHZdYSVMDhSkGw0HRaiQEmtYziBoGPas87Wn7X6+vP
wAzjIScxReffyzAVhsCD/2po01C0q4tbemCH8r0qox2qKMirK8W5UyoNQfGWLcKG
sysLLzwCu+jQh8JHYCnYSkXA6zpf91MS/3qL2Is+WV/p9kMDj3HDfNydaoGBAOTs
+JjLCTPzgpfgtsfgyWYT0P7t85mMHUHCeLcrEg96dUT18az79F3HR+IxQJ2R18L
wfgryZVGeRdyG3/+tpw/95/ChftEQ+6tdHE0LCpe6i/9JnYpSG65yBk7I3hS98QZ
DudaQtzb5t08dB8VP0Jfcyr+E/fldsvXiHbbGuchAoGBAPDi8txF8zZmCx73PGB+
R39r9oo0VPX2W+FxWgB9EAcZopg57nDJsKIDAPn+YRxQMWhxW980q2uftV3kaARx
ObIr1ktAP5iGUlVYDFDYpaa7HbwkRlFBGbaQMgORvc6YwA1IPFCxMjjAXGg6uc1
PfJWe2EcrQnI1dGRk9b0un85AoGALvmRaYkAMnDdc1eLWihz3NCAM2sLpIbDmr/+
q08g6xIJUXFo5xU10L5Ac6AVDRRIQPX2f348tyzm2ttTWuhjHwdtQJmH+xVe+zMM
mc4i4e7oD0DbC5QdENT+e+7usFoWDSMWlx7K+bTzs705QvvFye08luArXUCpiRs
azQ8S5EQCgYEAuBLGk33bLIGDegjG/FK+XCDie2S9SfVkpE4yOUz3ysjHtuSg+Q+k
TdN2/Wcjho02NCgca4An7mikhI47+9kx9J1qjR5xwNg5xSUoQ5CZbgsOUiKK4a5A
QLRiLiIuLma/tBVxjdfNBuY/wJR3bZ1cu7uXp8skDfAOJBMJLs0tTlc=
-----END RSA PRIVATE KEY-----

```

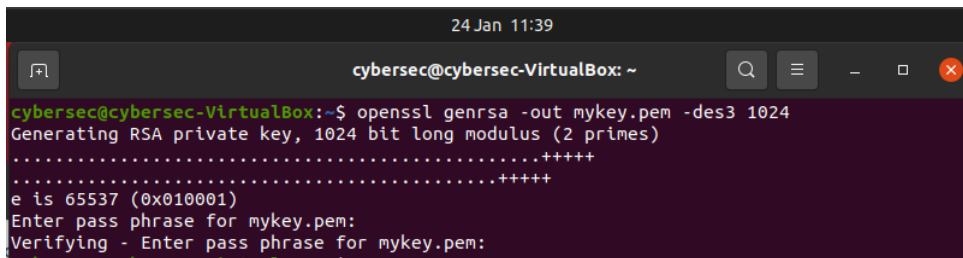
Figure 1.6: Generating a 2048-bit private key using genrsa.

This works to generate a private key, but this command merely outputs it to the console, and does not actually store it anywhere on the system.

1.3.2 Storing DES3 & passphrase encrypted RSA keys in a file

The key generated can then be encrypted using an encryption algorithm, which would be DES3 for this example, and a passphrase. Upon doing this, the key can be stored into a .pem file, also known as a Privacy Enhanced Mail file, which is a file format 'to provide the creation and validation of digital signatures, and in addition the encryption and decryption of signed data, based on asymmetric and symmetric cryptography.' (Kolletzki, 1996, p. 1894)

In this example, a 1024-bit key is created using DES3 and the passphrase ³ "secretkey".



```

24 Jan 11:39
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl genrsa -out mykey.pem -des3 1024
Generating RSA private key, 1024 bit long modulus (2 primes)
.....+++++
.....+++++
e is 65537 (0x010001)
Enter pass phrase for mykey.pem:
Verifying - Enter pass phrase for mykey.pem:

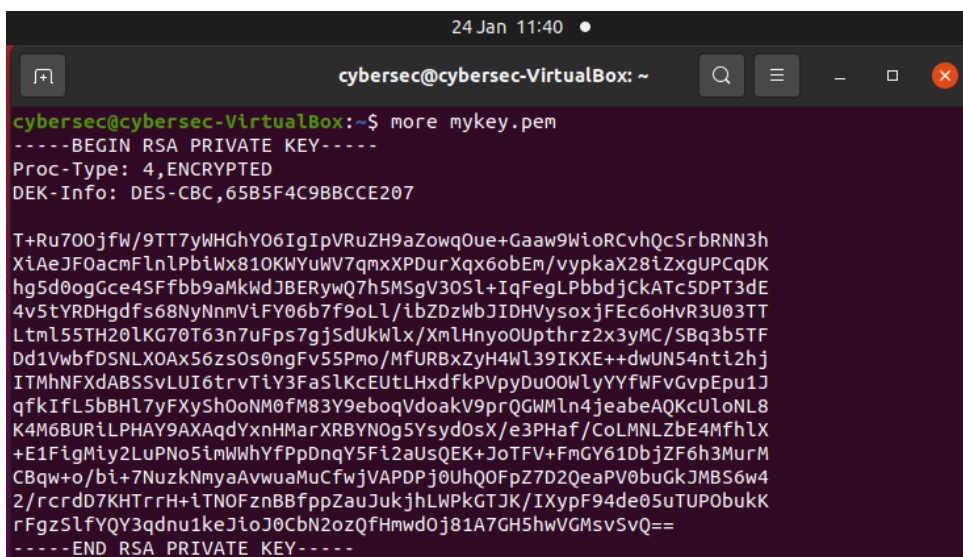
```

Figure 1.7: Generating and storing a 1024-bit private key using genrsa, DES3 and the passphrase "secretkey".

To break down this command:

- genrsa - Generate an RSA private key.
- -out mykey.pem - Save the generated key to the file "mykey.pem" in the current directory.
- -des3 - Using the DES3 encryption method.
- 1024 - The key will be 1024 bits.

We can then view this key by entering the command "more mykey.pem".



```

24 Jan 11:40
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ more mykey.pem
-----BEGIN RSA PRIVATE KEY-----
Proc-Type: 4, ENCRYPTED
DEK-Info: DES-CBC,65B5F4C9BBCCE207

T+Ru700jfw/9TT7yWHGhY06IgiPVRuZH9aZowqOue+Gaaw9WioRCvhQcSrbrRNN3h
XiAeJFOacmFlnlPbiWx810KWYUwV7qmxXPDurXqx6obEm/vypkaX28iZxgUPCqDK
hg5d0ogGce4SFfbb9aMkwdJBERywQ7h5MSgV30Sl+IqFegLPbbdjckATc5DPT3dE
4v5tYRDHgdFs68NyNnmViFY06b7f9oLL/ibZDzWbJIDHvysoxjFEc6oHvR3U03TT
Ltml5STH20lKG70T63n7uFps7gjSduKwLx/XmIHnyo0Upthr z2x3yMC/SBq3b5TF
Dd1VwbFD5NLX0Ax56zs0s0ngFv55Pmo/MFURBxZyH4Wl39IKXE++dwUN54nti2hj
ITMhNFXdABSSvLUI6trvtiY3FaSLKcEutLHxdfkPvPyDu00WlyYYfWfVgVpEpu1J
qfkiFL5bBHl7yFXySh0oNM0fM83Y9eboqVdoakV9prQGWMLn4jeabeAQKcULoNL8
K4M6BURiLPHAY9AXAqdYxnHMarXRBYN0g5Ysyd0sX/e3PHaf/CoLMNLZbE4MfhLX
+E1FiGmIy2LuPno5imWWhYfPpDnqY5Fi2aUsQEK+JoTFV+FmGY61DbjZF6h3MurM
CBQw+o/bi+7NuzkNmyaAvwuaMuCfwjVAPDPj0UHQFpZ7D2QeaPV0buGkJMBS6w4
2/rcrd07KHTrrH+iTNOFznBBfppZauJukjhLWPkGTJK/IXypF94de05uTUP0buk
rFgzSLfYQY3qdnui1keJioJ0CbN2ozQfHmwd0j81A7GH5hwVGMsvSvQ==
-----END RSA PRIVATE KEY-----

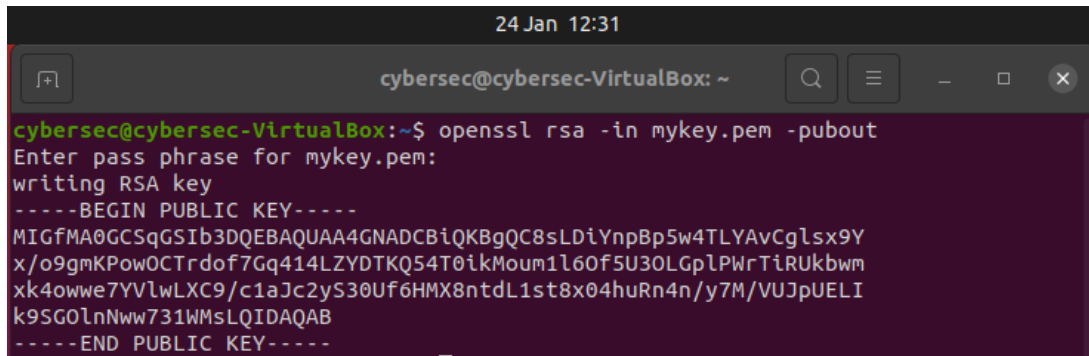
```

Figure 1.8: The key stored in mykey.pem.

³The passphrase is not visible when typed, and therefore does not show in the image.

1.3.3 Getting a public key from the private key

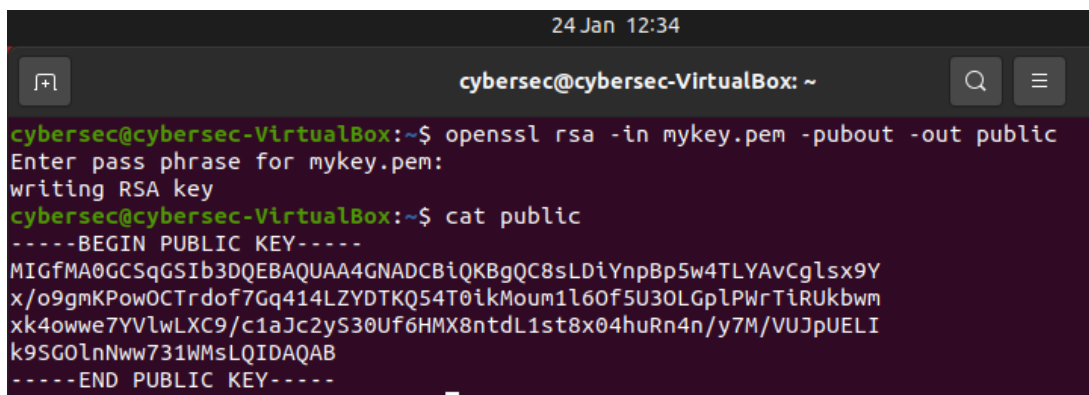
The private key stored into "mykey.pem" by the previous command can be accessed again to generate a public key, which would be used when data is encrypted. To do so, the passphrase we set earlier (secretkey) must be entered.



```
24 Jan 12:31
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl rsa -in mykey.pem -pubout
Enter pass phrase for mykey.pem:
writing RSA key
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8sLDiYnpBp5w4TLyAvCglsx9Y
x/o9gmKPowOCTrdof7Gq414LZYDTKQ54T0ikMoum1l60f5U30LGplPwrTiRukbwm
xk4owwe7YVlwLXC9/c1aJc2yS30Uf6HMX8ntdL1st8x04huRn4n/y7M/VUJpUELI
k9SG0lnNww731WmsLQIDAQAB
-----END PUBLIC KEY-----
```

Figure 1.9: The public key generated from mykey.pem.

Breaking down this command, "-in mykey.pem" uses mykey.pem as the input for the command, and "-pubout" outputs the public key generated from the private key. However, this command only outputs said key to the console, so an altered version is necessary to save the key to a file.



```
24 Jan 12:34
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl rsa -in mykey.pem -pubout -out public
Enter pass phrase for mykey.pem:
writing RSA key
cybersec@cybersec-VirtualBox:~$ cat public
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC8sLDiYnpBp5w4TLyAvCglsx9Y
x/o9gmKPowOCTrdof7Gq414LZYDTKQ54T0ikMoum1l60f5U30LGplPwrTiRukbwm
xk4owwe7YVlwLXC9/c1aJc2yS30Uf6HMX8ntdL1st8x04huRn4n/y7M/VUJpUELI
k9SG0lnNww731WmsLQIDAQAB
-----END PUBLIC KEY-----
```

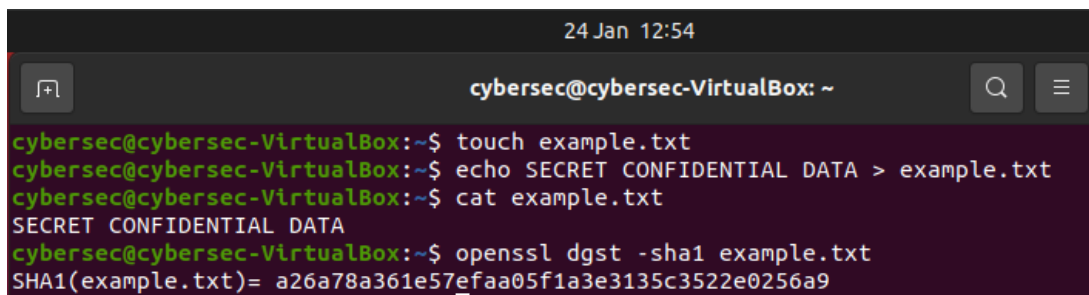
Figure 1.10: Storing the public key in a file.

The additional argument "-out public" is set in this command, meaning the public key will be written to a file called "public". This file can then be read using the cat command, revealing the public key seen in Figure 1.9.

1.3.4 Obtaining a message/file digest

When data is transmitted, it could become corrupted or potentially even intercepted and modified before it reaches its intended recipient. To mitigate the risks from this, files can have "digests", which are the result of hashing their contents. If the file is modified whatsoever, even by a single byte, the digest would be different, meaning the file has been corrupted or tampered with. These are also the basis of digital signatures; by encrypting a digest with your private key, you cannot deny that you were the source of the file.

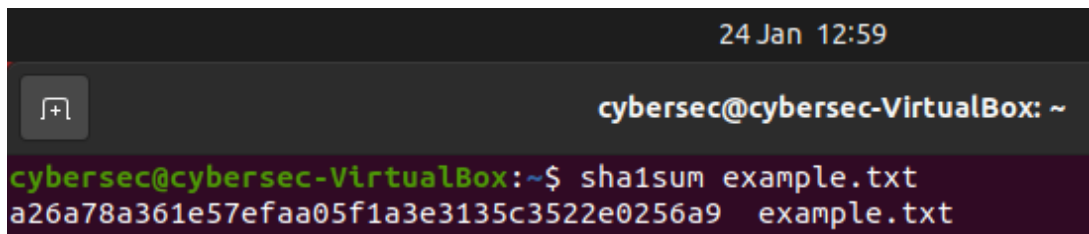
OpenSSL can generate digests using its "dgst" command.



```
24 Jan 12:54
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ touch example.txt
cybersec@cybersec-VirtualBox:~$ echo SECRET CONFIDENTIAL DATA > example.txt
cybersec@cybersec-VirtualBox:~$ cat example.txt
SECRET CONFIDENTIAL DATA
cybersec@cybersec-VirtualBox:~$ openssl dgst -sha1 example.txt
SHA1(example.txt)= a26a78a361e57efaa05f1a3e3135c3522e0256a9
```

Figure 1.11: Creating a file, then getting the SHA1 digest of it.

This can also be verified by using a non-OpenSSL command, `sha1sum`, which returns the same digest as the OpenSSL `dgst`.



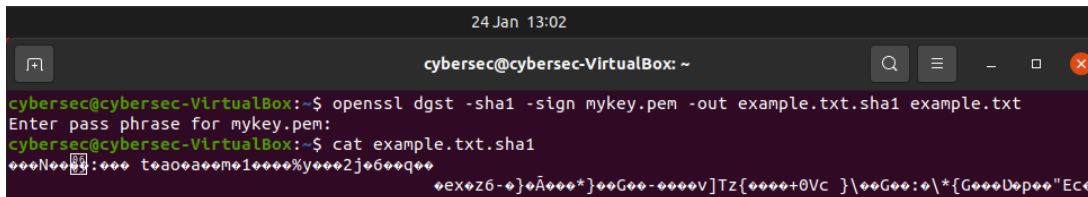
```
24 Jan 12:59
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ sha1sum example.txt
a26a78a361e57efaa05f1a3e3135c3522e0256a9 example.txt
```

Figure 1.12: Verifying the digest.

1.3.5 Signing a digest

Non-repudiation is an important topic in cybersecurity - verifying the original sender of a file is an imperative task in the event of any issues that may arise such as malware being appended to the file, in either the prosecution or defense of the alleged sender. Signing a message digest using your private key definitively proves the device that data was sent from, meaning that it cannot be denied that the file was sent, nor who it was sent by.

The previously used "example.txt" can again be used here to generate a digest encrypted using the "mykey.pem" private key established earlier, which signs the digest.

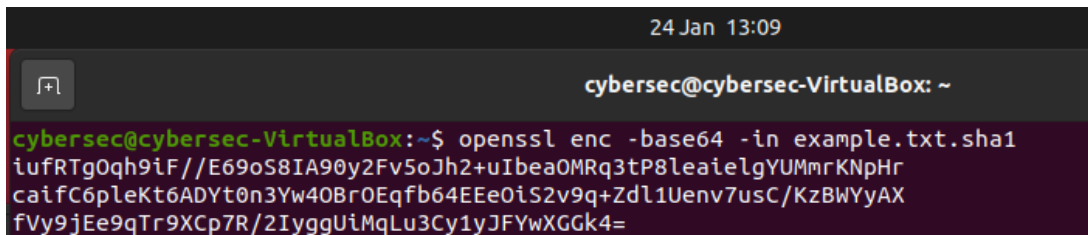


```

24 Jan 13:02
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl dgst -sha1 -sign mykey.pem -out example.txt.sha1 example.txt
Enter pass phrase for mykey.pem:
cybersec@cybersec-VirtualBox:~$ cat example.txt.sha1
-----BEGIN-----
tAo0a0m01000%y0002j0600q00
0ex0z6-0}0Ã000*}00G00-0000v]Tz{0000+0Vc }00G00:0{*{G000D0p00"E0c0
-----END-----
  
```

Figure 1.13: Writing a signed digest to a file.

Note that when we try to read this file, it is completely illegible, as it is not in a compatible format. To counteract this, it can be converted to Base64 using OpenSSL's "enc" command, passing the generated file as the argument.



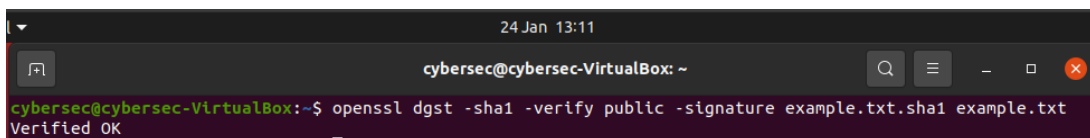
```

24 Jan 13:09
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl enc -base64 -in example.txt.sha1
iufRTg0qh9iF//E69oS8IA90y2Fv5oJh2+uIbeaOMRq3tP8leaieIgyUMmrKNpHr
caifC6pleKt6ADYt0n3Yw40Br0Eqfb64EEe0iS2v9q+Zd1lUenv7usC/KzBWYyAX
fVy9jEe9qTr9XCp7R/2IyggUiMqLu3Cy1yJFYwXGgk4=
  
```

Figure 1.14: Encoding the signed digest to Base64.

This doesn't have any use other than allowing us to see the key in a Base64 format - the key functions even if it can't be conventionally read.

Now that we have the signed digest, it can be verified using the public key, which confirms the authenticity of the data in example.txt.

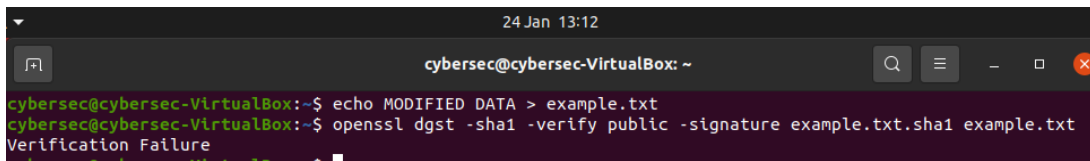


```

24 Jan 13:11
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ openssl dgst -sha1 -verify public -signature example.txt.sha1 example.txt
Verified OK
  
```

Figure 1.15: Verifying the signature of example.txt.

This returns "Verified OK" as intended, as the file has not been modified. If the file does get modified through either corruption or a threat actor, the digest would not be the same, which can be verified by modifying the file ourselves and then verifying the digest of the file once again.



```
24 Jan 13:12
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ echo MODIFIED DATA > example.txt
cybersec@cybersec-VirtualBox:~$ openssl dgst -sha1 -verify public -signature example.txt.sha1 example.txt
Verification Failure
```

Figure 1.16: Failing to verify the signature of example.txt, as it has been modified.

This returns "Verification Failure", as the digest would now be different due to example.txt now containing different data than it did when the SHA1 digest was created.

Lab 2 - Usage of GPG

Important note

In this lab, an incompatibility meant that instead of using the base GPG program, the alternative **GnuPG1** was used. Therefore, commands use the phrase "gpg1" instead of "gpg".

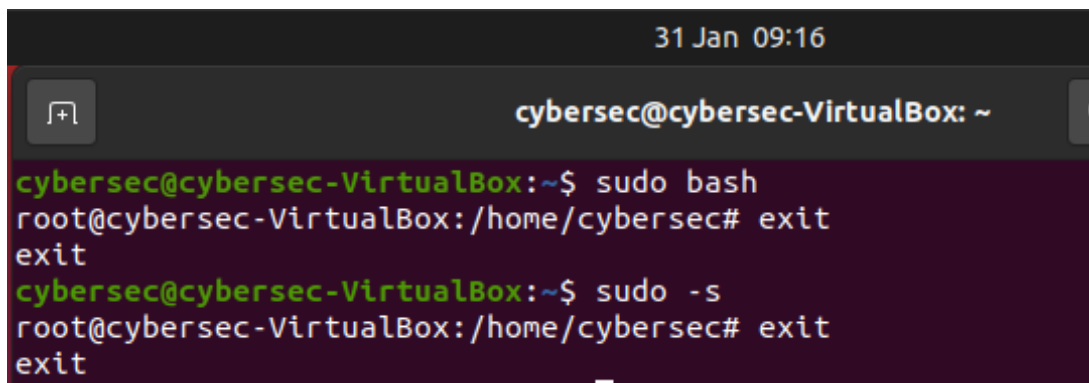
This lab expanded on the concepts of asymmetric encryption through the use of GPG/GnuPG (GNU Privacy Guard) to produce, sign and verify public and private keys.

2.1 Creating test users

For this lab, two test users were created and used to execute the necessary commands.

2.1.1 Elevating the terminal

To add users to the system, administrative privileges are required. To gain the necessary privileges, the command "sudo -s" or "sudo bash" can be entered (both commands are functionally identical) which will change the terminal to be at root level.

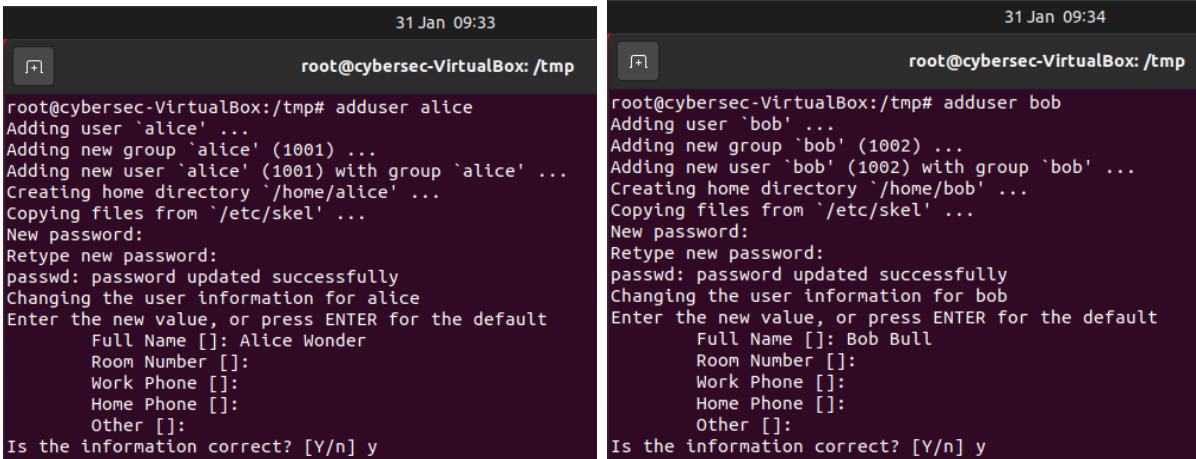
A screenshot of a terminal window titled "31 Jan 09:16" and "cybersec@cybersec-VirtualBox: ~". The terminal shows the user "cybersec" at the prompt "cybersec@cybersec-VirtualBox:~\$". They enter "sudo bash", which prompts them to become "root" at the prompt "root@cybersec-VirtualBox:/home/cybersec#". They then enter "exit" to return to the user prompt. This process is repeated with "sudo -s" and "exit".

```
31 Jan 09:16
cybersec@cybersec-VirtualBox: ~
cybersec@cybersec-VirtualBox:~$ sudo bash
root@cybersec-VirtualBox:/home/cybersec# exit
exit
cybersec@cybersec-VirtualBox:~$ sudo -s
root@cybersec-VirtualBox:/home/cybersec# exit
exit
```

Figure 2.1: Elevating the terminal.

2.1.2 Creating Bob and Alice

With the elevated privileges gained from being a superuser, it is now possible to add users to the system using "adduser" followed by the given username. A password for the user will then be necessary, followed by optional information such as phone numbers, which are left blank for this lab.



The image shows two terminal windows side-by-side. The left window, titled 'root@cybersec-VirtualBox: /tmp' with a timestamp of '31 Jan 09:33', shows the command 'adduser alice' being executed. It prompts for a password, confirms the user information (Full Name: Alice Wonder, Room Number, Work Phone, Home Phone, Other), and asks if the information is correct, with 'y' being entered. The right window, titled 'root@cybersec-VirtualBox: /tmp' with a timestamp of '31 Jan 09:34', shows the command 'adduser bob' being executed. It prompts for a password, confirms the user information (Full Name: Bob Bull, Room Number, Work Phone, Home Phone, Other), and asks if the information is correct, with 'y' being entered.

Figure 2.2: Creating users 'bob' and 'alice'.

For ease of access, multiple terminal tabs can be open at a time, so I elected to use one for the superuser root, and one each for Bob and Alice.

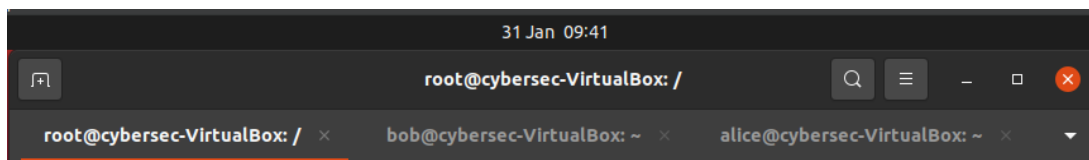
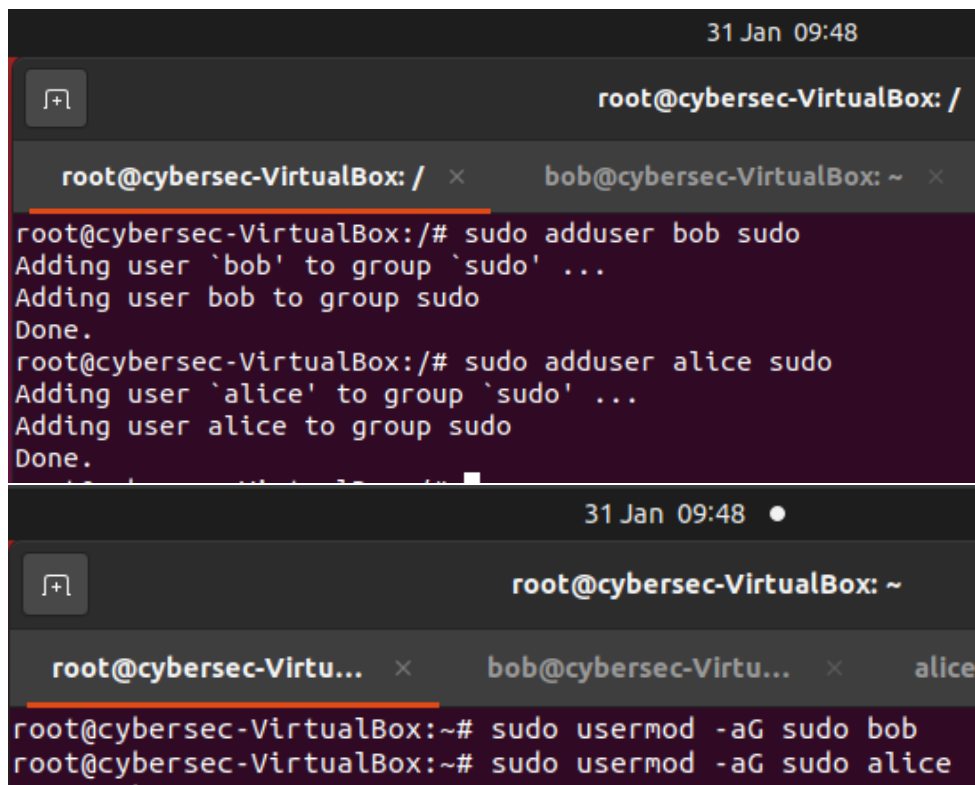


Figure 2.3: Multiple terminal tabs.

I also added these new users to the "sudo" group, allowing them to also use the sudo command to execute commands with elevated permissions.



```
31 Jan 09:48
root@cybersec-VirtualBox: /

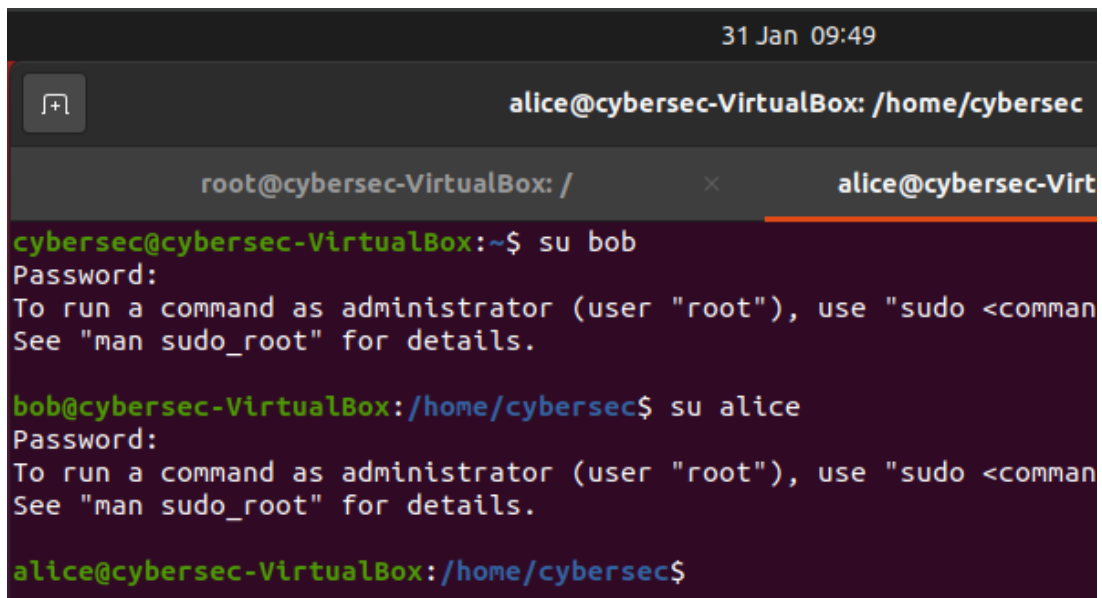
root@cybersec-VirtualBox: / x bob@cybersec-VirtualBox: ~ x
root@cybersec-VirtualBox:/# sudo adduser bob sudo
Adding user `bob' to group `sudo' ...
Adding user bob to group sudo
Done.
root@cybersec-VirtualBox:/# sudo adduser alice sudo
Adding user `alice' to group `sudo' ...
Adding user alice to group sudo
Done.

31 Jan 09:48
root@cybersec-VirtualBox: ~

root@cybersec-Virtu... x bob@cybersec-Virtu... x alice
root@cybersec-VirtualBox:~# sudo usermod -aG sudo bob
root@cybersec-VirtualBox:~# sudo usermod -aG sudo alice
```

Figure 2.4: Adding bob and alice to sudo.

It is possible to switch the active terminal user using the command "su" followed by the account to switch to, and then the password of the given account.



```
31 Jan 09:49
alice@cybersec-VirtualBox: /home/cybersec

root@cybersec-VirtualBox: / x alice@cybersec-Virt
cybersec@cybersec-VirtualBox:~$ su bob
Password:
To run a command as administrator (user "root"), use "sudo <command>"
See "man sudo_root" for details.

bob@cybersec-VirtualBox:/home/cybersec$ su alice
Password:
To run a command as administrator (user "root"), use "sudo <command>"
See "man sudo_root" for details.

alice@cybersec-VirtualBox:/home/cybersec$
```

Figure 2.5: Switching the active terminal user. Note the prompt about running commands as an administrator, which signifies that they were successfully added to the sudo group.

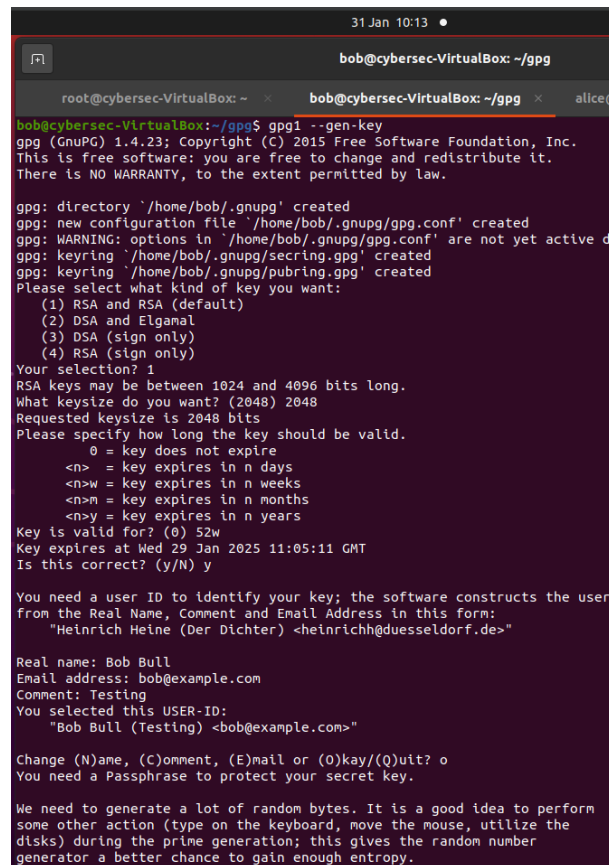
2.2 Exchanging encrypted files over an insecure channel

For this section, assume that all commands have been executed on **both** the Bob and Alice user accounts unless stated otherwise.

On standard Linux distributions, the `/tmp` directory is a public directory accessible to all users. For this reason, it is therefore insecure, as every user on the system can read the files placed there.¹ To transfer files across insecure channels such as `/tmp/`, they should first be encrypted so that they can only be read and/or used by their intended recipient. Therefore, GNU Privacy Guard (GPG hereafter) can be used to generate and store public and private asymmetric keys.

2.2.1 Generating public/private key-pairs

To generate a private key, the command `"gpg1 --gen-key"` can be used.



```

31 Jan 10:13 •
bob@cybersec-VirtualBox: ~/gpg
root@cybersec-VirtualBox: ~
bob@cybersec-VirtualBox: ~/gpg
alice@
bob@cybersec-VirtualBox:~/gpg$ gpg1 --gen-key
gpg (GnuPG) 1.4.23; Copyright (C) 2015 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.

gpg: directory '/home/bob/.gnupg' created
gpg: new configuration file '/home/bob/.gnupg/gpg.conf' created
gpg: WARNING: options in '/home/bob/.gnupg/gpg.conf' are not yet active due
gpg: keyring '/home/bob/.gnupg/secring.gpg' created
gpg: keyring '/home/bob/.gnupg/pubring.gpg' created
Please select what kind of key you want:
(1) RSA and RSA (default)
(2) DSA and Elgamal
(3) DSA (sign only)
(4) RSA (sign only)
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (2048) 2048
Requested keysize is 2048 bits
Please specify how long the key should be valid.
0 = key does not expire
<n> = key expires in n days
<n>w = key expires in n weeks
<n>m = key expires in n months
<n>y = key expires in n years
Key is valid for? (0) 52w
Key expires at Wed 29 Jan 2025 11:05:11 GMT
Is this correct? (y/N) y

You need a user ID to identify your key; the software constructs the user
from the Real Name, Comment and Email Address in this form:
"Heinrich Heine (Der Dichter) <heinrichh@duesseldorf.de>"

Real name: Bob Bull
Email address: bob@example.com
Comment: Testing
You selected this USER-ID:
"Bob Bull (Testing) <bob@example.com>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? o
You need a Passphrase to protect your secret key.

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.

```

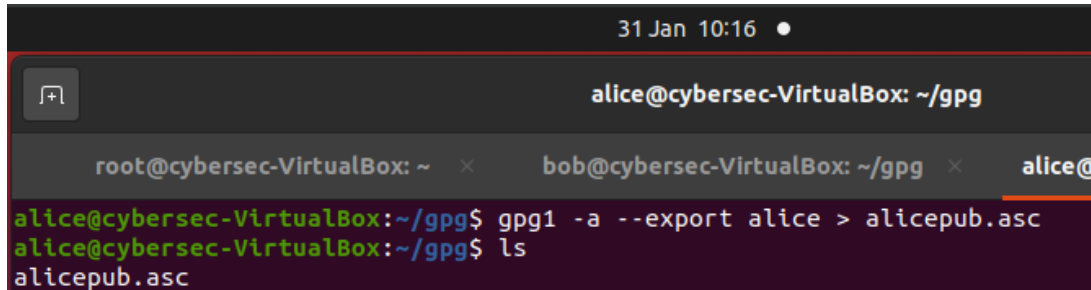
Figure 2.6: Generating a private key.

This will open a submenu where the user can select the kind of key they wish to generate, as well as the size and expiry date of the key. Once this is established, they must create a user ID if they didn't already have one, consisting of their full name, email address and an optional comment. While the key generates, the user is prompted to perform random inputs such as moving the mouse and typing on the keyboard to enhance the randomness of the generated key. A key was also generated for Alice.

¹However, they cannot update/change them without sudo permissions.

2.2.2 Exporting public keys

It is possible to export the public keys from the generated key-pairs using GPG's export command.



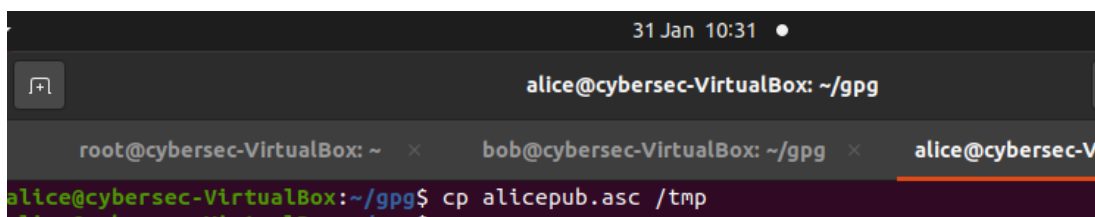
```
31 Jan 10:16 ●
alice@cybersec-VirtualBox: ~/gpg

root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@

alice@cybersec-VirtualBox:~/gpg$ gpg1 -a --export alice > alicepub.asc
alice@cybersec-VirtualBox:~/gpg$ ls
alicepub.asc
```

Figure 2.7: Exporting Alice's public key.

This exports the public key in ASCII format (due to the use of the -a flag) to the file "alicepub.asc". This can be seen by using "ls" to show the files in the directory.² Because this is Alice's **public** key, we are comfortable sharing this to the public /tmp/ directory where all users can see it.



```
31 Jan 10:31 ●
alice@cybersec-VirtualBox: ~/gpg

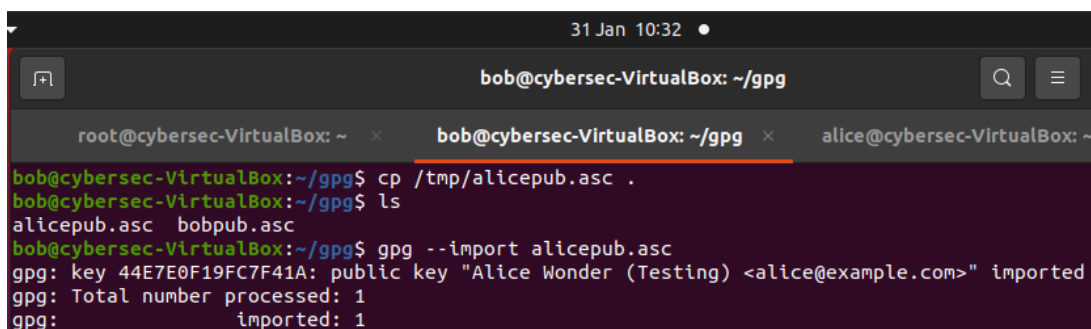
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybersec-V

alice@cybersec-VirtualBox:~/gpg$ cp alicepub.asc /tmp
```

Figure 2.8: Copying Alice's public key to /tmp.

2.2.3 Importing and signing public keys

Now that Alice's public key is in /tmp, Bob can copy this to his own directory and import it using GPG.



```
31 Jan 10:32 ●
bob@cybersec-VirtualBox: ~/gpg

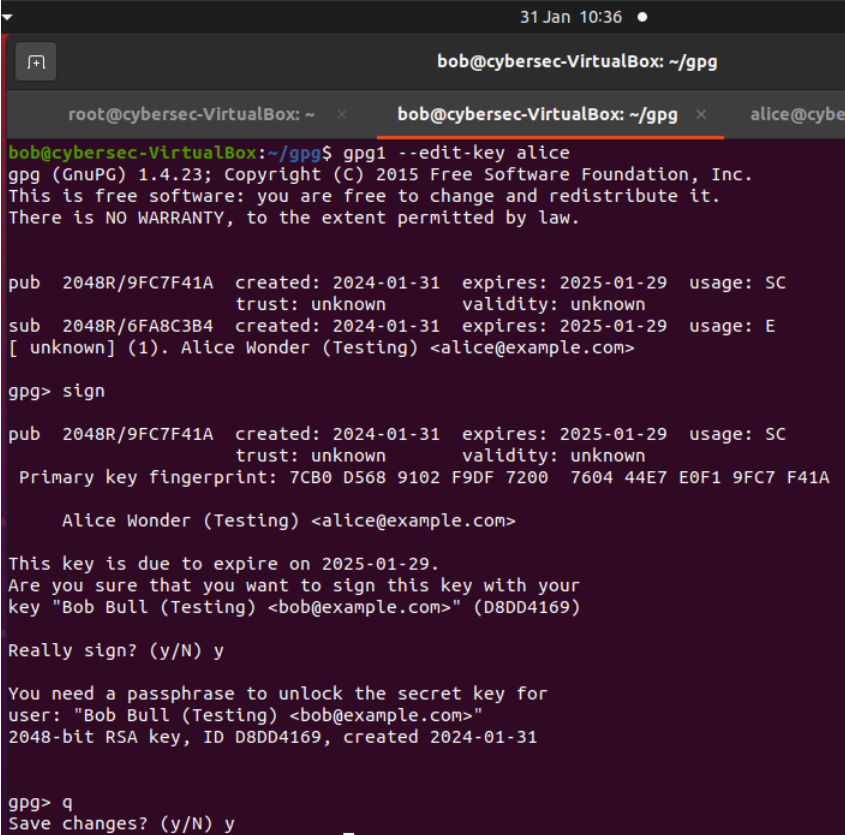
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybersec-VirtualBox: ~/gpg

bob@cybersec-VirtualBox:~/gpg$ cp /tmp/alicepub.asc .
bob@cybersec-VirtualBox:~/gpg$ ls
alicepub.asc  bobpub.asc
bob@cybersec-VirtualBox:~/gpg$ gpg --import alicepub.asc
gpg: key 44E7E0F19FC7F41A: public key "Alice Wonder (Testing) <alice@example.com>" imported
gpg: Total number processed: 1
gpg: imported: 1
```

Figure 2.9: Importing Alice's public key as Bob.

²The file can be read using "cat alicepub.asc", but it is a 2048-bit key, so it would completely fill the terminal window.

Bob can then **sign** this key, verifying that he trusts that this key does belong to Alice. This is done by editing Alice's key as Bob, signing it, and then saving this.



```
31 Jan 10:36
bob@cybersec-VirtualBox: ~/gpg
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybe
bob@cybersec-VirtualBox:~/gpg$ gpg1 --edit-key alice
gpg (GnuPG) 1.4.23; Copyright (C) 2015 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 2048R/9FC7F41A created: 2024-01-31 expires: 2025-01-29 usage: SC
trust: unknown validity: unknown
sub 2048R/6FA8C3B4 created: 2024-01-31 expires: 2025-01-29 usage: E
[ unknown] (1). Alice Wonder (Testing) <alice@example.com>

gpg> sign

pub 2048R/9FC7F41A created: 2024-01-31 expires: 2025-01-29 usage: SC
trust: unknown validity: unknown
Primary key fingerprint: 7CB0 D568 9102 F9DF 7200 7604 44E7 E0F1 9FC7 F41A

Alice Wonder (Testing) <alice@example.com>

This key is due to expire on 2025-01-29.
Are you sure that you want to sign this key with your
key "Bob Bull (Testing) <bob@example.com>" (D8DD4169)

Really sign? (y/N) y

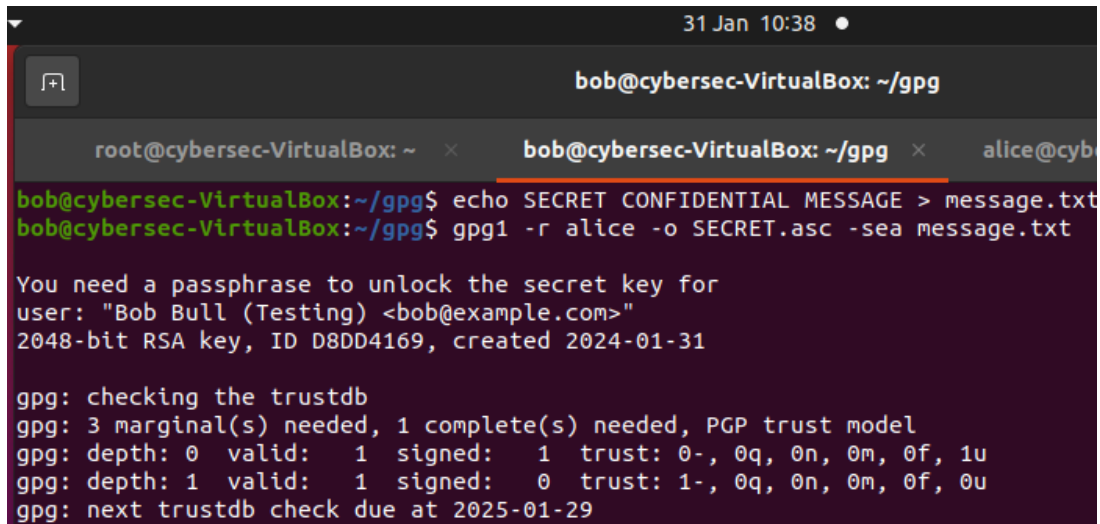
You need a passphrase to unlock the secret key for
user: "Bob Bull (Testing) <bob@example.com>"
2048-bit RSA key, ID D8DD4169, created 2024-01-31

gpg> q
Save changes? (y/N) y
```

Figure 2.10: Bob signing Alice's public key.

2.2.4 Encrypting and decrypting data

Now that Alice and Bob have their key-pairs generated, they can transfer asymmetrically encrypted data to each other. This was tested by making a file, encrypting it using Alice's public key, and copying it to the /tmp directory.



```
31 Jan 10:38
bob@cybersec-VirtualBox: ~/gpg
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybersec-VirtualBox: ~/gpg
bob@cybersec-VirtualBox:~/gpg$ echo SECRET CONFIDENTIAL MESSAGE > message.txt
bob@cybersec-VirtualBox:~/gpg$ gpg1 -r alice -o SECRET.asc -sea message.txt

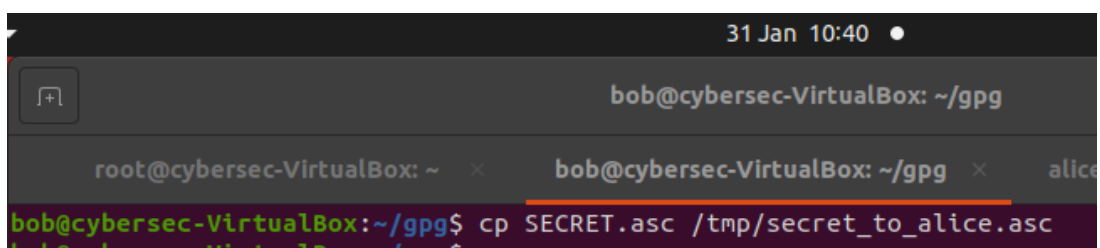
You need a passphrase to unlock the secret key for
user: "Bob Bull (Testing) <bob@example.com>"
2048-bit RSA key, ID D8DD4169, created 2024-01-31

gpg: checking the trustdb
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   1  signed:   1  trust: 0-, 0q, 0n, 0m, 0f, 1u
gpg: depth: 1  valid:   1  signed:   0  trust: 1-, 0q, 0n, 0m, 0f, 0u
gpg: next trustdb check due at 2025-01-29
```

Figure 2.11: Making a file and encrypting it using Alice's public key.

This command can be broken down to its components:

- -r alice - Sets Alice as the recipient of the file by using her public key.
- -o SECRET.asc - Outputs the encrypted data to SECRET.asc.
- -sea message.txt - Sign and encrypt the contents of message.txt in ASCII format.



```
31 Jan 10:40
bob@cybersec-VirtualBox: ~/gpg
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybersec-VirtualBox: ~/gpg
bob@cybersec-VirtualBox:~/gpg$ cp SECRET.asc /tmp/secret_to_alice.asc
```

Figure 2.12: Copying the encrypted file to /tmp with the name "secret_to_alice.asc".

Alice can then decrypt "secret_to_alice.asc" and output the results to "message.txt", where they can then be read in human-legible form.

```
31 Jan 10:45 ●
alice@cybersec-VirtualBox: ~/gpg
root@cybersec-VirtualBox: ~ x bob@cybersec-VirtualBox: ~/gpg x alice@cybersec-Vi
alice@cybersec-VirtualBox:~/gpg$ gpg1 -o message.txt -d /tmp/secret_to_alice.asc
You need a passphrase to unlock the secret key for
user: "Alice Wonder (Testing) <alice@example.com>"
2048-bit RSA key, ID 6FA8C3B4, created 2024-01-31 (main key ID 9FC7F41A)

gpg: encrypted with 2048-bit RSA key, ID 6FA8C3B4, created 2024-01-31
      "Alice Wonder (Testing) <alice@example.com>"
gpg: Signature made Wed 31 Jan 2024 10:38:43 GMT using RSA key ID D8DD4169
gpg: Good signature from "Bob Bull (Testing) <bob@example.com>"
alice@cybersec-VirtualBox:~/gpg$ ls
alicepub.asc bobpub.asc message.txt
alice@cybersec-VirtualBox:~/gpg$ cat message.txt
SECRET CONFIDENTIAL MESSAGE
```

Figure 2.13: Decrypting the encrypted message and reading it.

Lab 5 - Discretionary Access Control

This lab explored the use of Discretionary Access Control methods on a Linux system, which allows the owner of an object to assign the level of access that others will have to said object.

5.1 Creating test users and groups

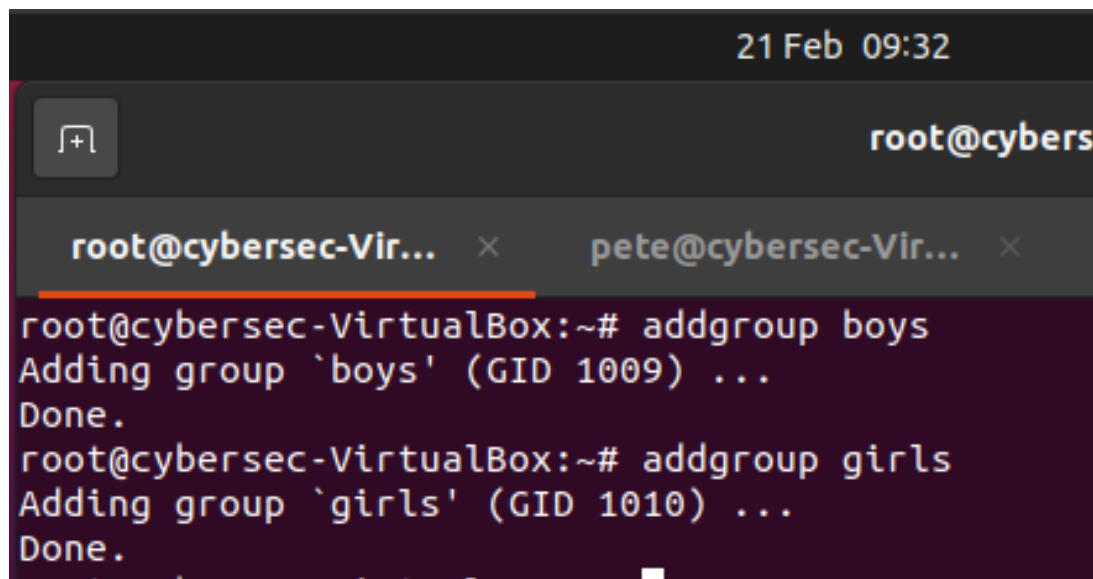
Note

Commands not pictured in this section were already showcased and explained in further detail in Lab 2, specifically in figures 2.1, 2.2 and 2.4 of section 2.1.

For this lab, three test users "Pete", "Ali" and "Mary" were added to the system. Pete and Ali were assigned to the "Boys" group, whereas Mary was assigned to the "Girls" group.

5.1.1 Creating groups

With sudo privileges, additional groups can be added to the system.

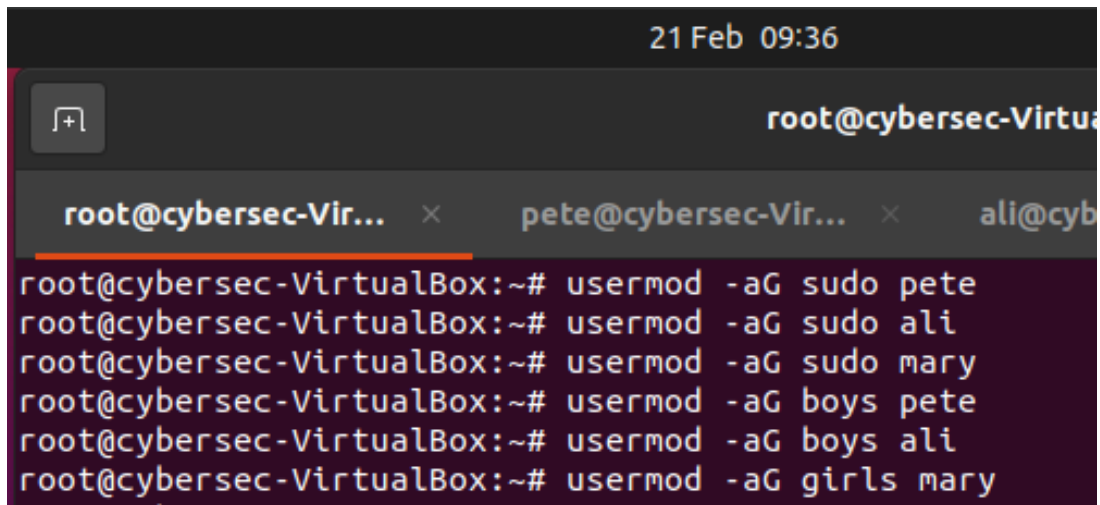
A screenshot of a terminal window. At the top right, it shows the date and time '21 Feb 09:32'. Below that, the prompt 'root@cybersec-Vir...' is visible. The terminal shows two commands being executed: 'addgroup boys' and 'addgroup girls'. The output for the first command is 'Adding group `boys` (GID 1009) ... Done.' and for the second is 'Adding group `girls` (GID 1010) ... Done.'.

```
root@cybersec-VirtualBox:~# addgroup boys
Adding group `boys` (GID 1009) ...
Done.
root@cybersec-VirtualBox:~# addgroup girls
Adding group `girls` (GID 1010) ...
Done.
```

Figure 5.1: Making the "boys" and "girls" groups.

5.1.2 Adding users to groups

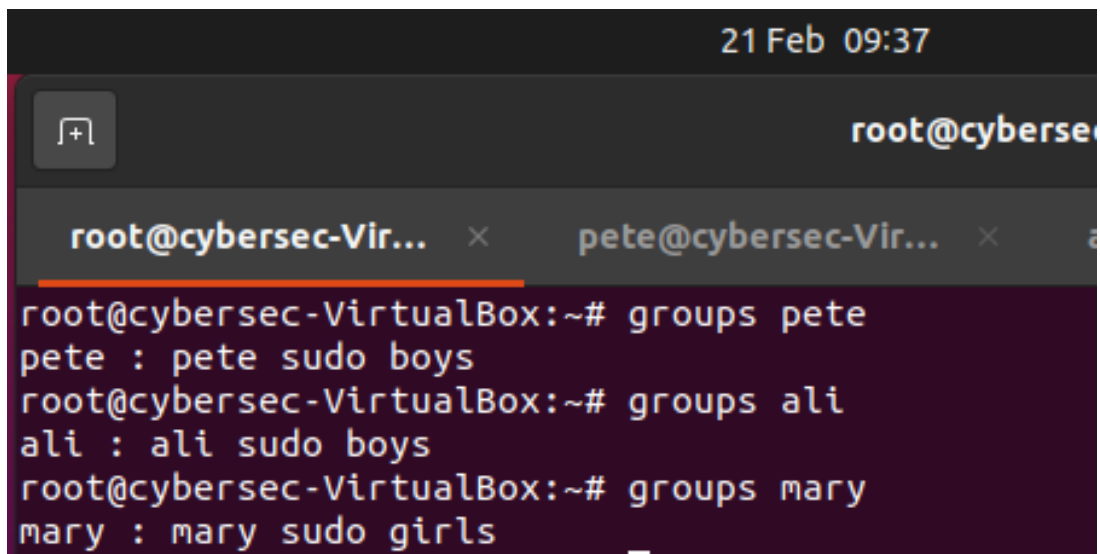
The new users were added to the groups mentioned above as well as the sudo group, enabling their user accounts to run commands with sudo privileges.



```
21 Feb 09:36
root@cybersec-Virtua
root@cybersec-Vir... x pete@cybersec-Vir... x ali@cyb
root@cybersec-VirtualBox:~# usermod -aG sudo pete
root@cybersec-VirtualBox:~# usermod -aG sudo ali
root@cybersec-VirtualBox:~# usermod -aG sudo mary
root@cybersec-VirtualBox:~# usermod -aG boys pete
root@cybersec-VirtualBox:~# usermod -aG boys ali
root@cybersec-VirtualBox:~# usermod -aG girls mary
```

Figure 5.2: Adding the users to sudo and their respective groups.

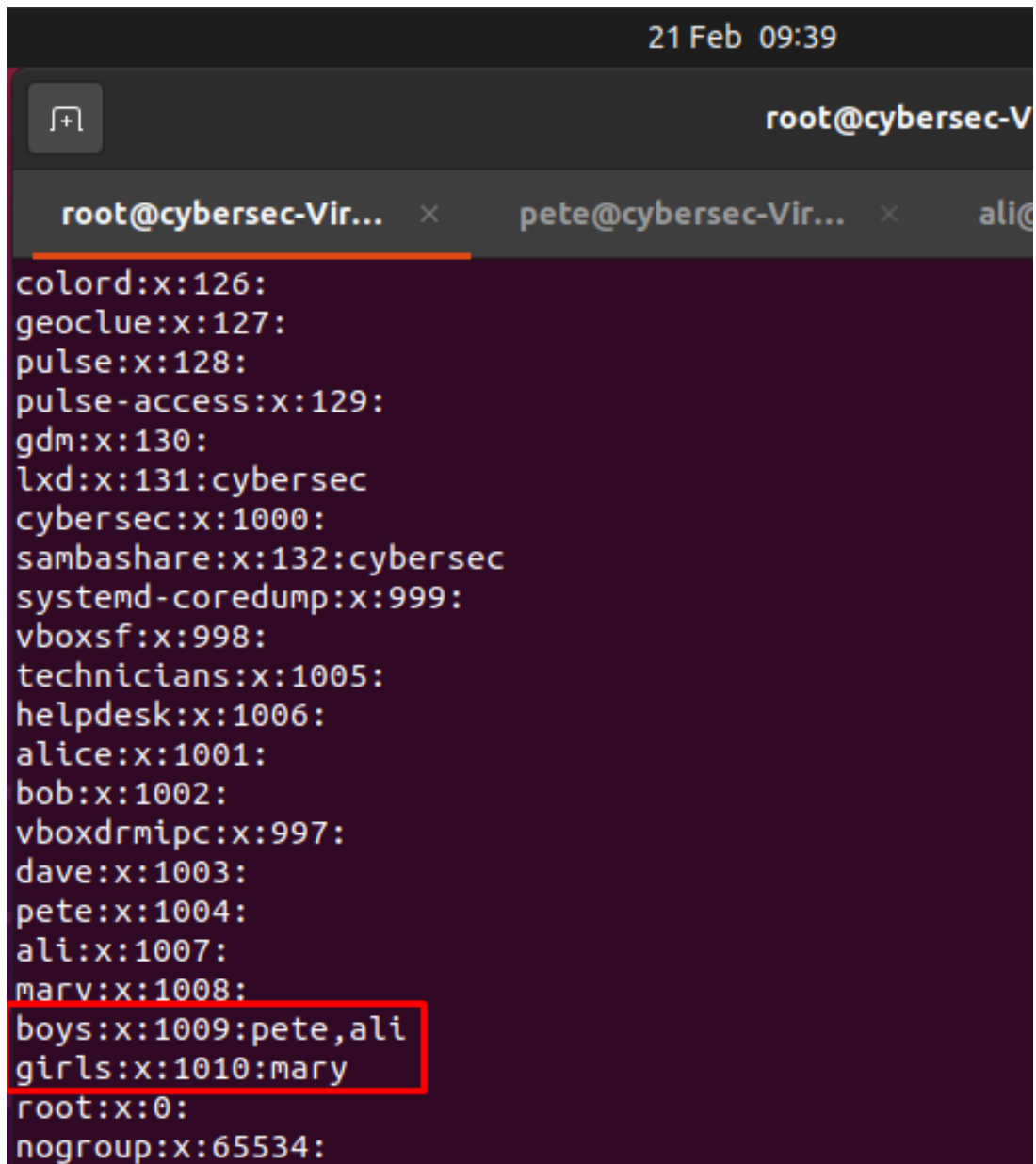
We can verify which groups a given user is in by using the "groups [username]" command.



```
21 Feb 09:37
root@cybersec-Virtua
root@cybersec-Vir... x pete@cybersec-Vir... x a
root@cybersec-VirtualBox:~# groups pete
pete : pete sudo boys
root@cybersec-VirtualBox:~# groups ali
ali : ali sudo boys
root@cybersec-VirtualBox:~# groups mary
mary : mary sudo girls
```

Figure 5.3: Verifying that the users were added to the groups.

This can also be checked by viewing all groups on the system via "getent groups".



```
21 Feb 09:39
root@cybersec-V...
root@cybersec-Vir... x pete@cybersec-Vir... x ali@
colord:x:126:
geoclue:x:127:
pulse:x:128:
pulse-access:x:129:
gdm:x:130:
lxd:x:131:cybersec
cybersec:x:1000:
sambashare:x:132:cybersec
systemd-coredump:x:999:
vboxsf:x:998:
technicians:x:1005:
helpdesk:x:1006:
alice:x:1001:
bob:x:1002:
vboxdrmipc:x:997:
dave:x:1003:
pete:x:1004:
ali:x:1007:
marv:x:1008:
boys:x:1009:pete,ali
girls:x:1010:mary
root:x:0:
nogroup:x:65534:
```

Figure 5.4: Seeing all groups (Boys and Girls are highlighted).

5.2 Using chmod and chgrp to assign permissions

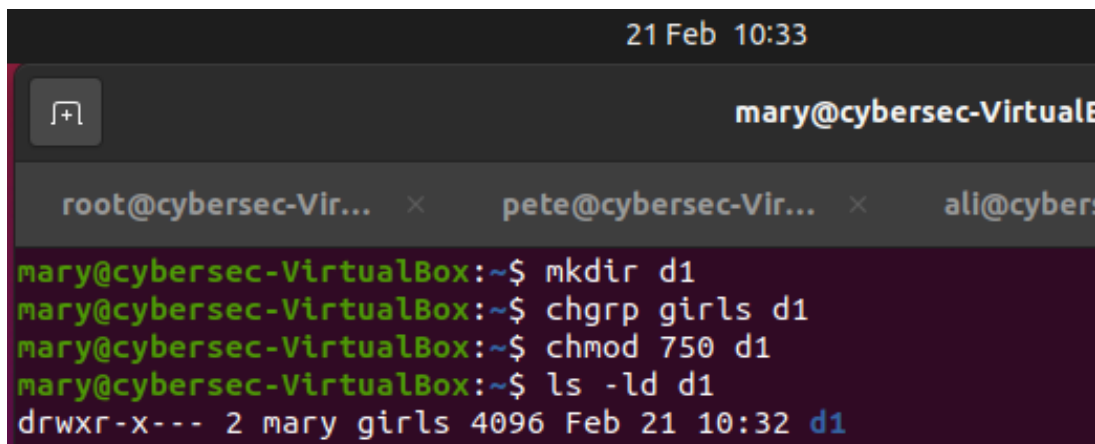
Chmod is a command that changes the permissions for a given file or directory. It can change permissions for reading, writing and executing files for the owner of the file, a group of users, other users¹ or all users.

5.2.1 Mnemonic and octal chmod

Chmod's arguments can be supplied as letters (mnemonic) or numbers (octal). Functionally, there is no difference between the two, as the permissions will be changed regardless. A mnemonic chmod command would be "chmod a rwx message.txt", which gives all users read, write and execute permissions for the file message.txt. An octal chmod command would be "chmod 777 message.txt", which gives the owner, group members and others read, write and execute permissions. The number 777 correlates to this because octal chmod operates by numeric sums. Reading permissions are the number 4, writing is the number 2 and executing is the number 1. Adding these together results in the number 7.

5.2.2 Restricting directory access

For the purposes of testing, a directory called D1 was added to user Mary's home. This directory was associated with the girls group via chgrp, and modified with an octal chmod command so that other users cannot access the directory whatsoever, but Mary herself and users of the girls group can read and execute from it.



```
21 Feb 10:33
mary@cybersec-VirtualBox
root@cybersec-Vir... x pete@cybersec-Vir... x ali@cybers
mary@cybersec-VirtualBox:~$ mkdir d1
mary@cybersec-VirtualBox:~$ chgrp girls d1
mary@cybersec-VirtualBox:~$ chmod 750 d1
mary@cybersec-VirtualBox:~$ ls -ld d1
drwxr-x--- 2 mary girls 4096 Feb 21 10:32 d1
```

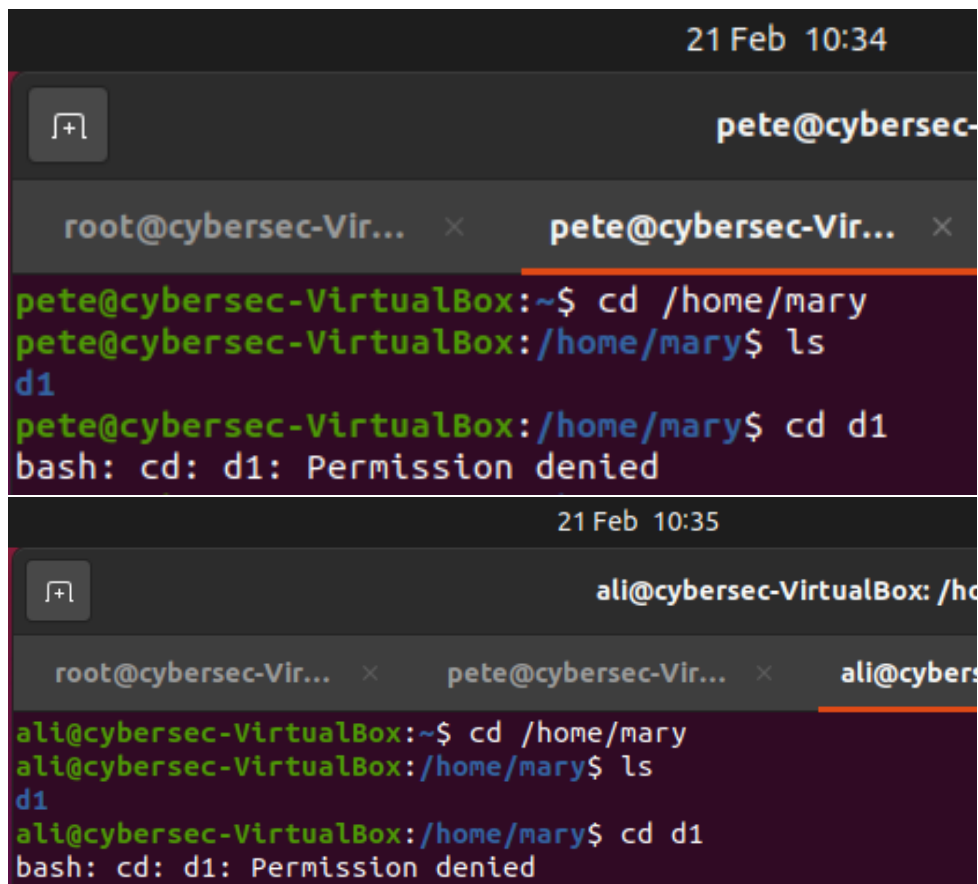
Figure 5.5: Creating D1 and modifying its permissions.

By using the command "ls -ld", the permissions of the directory are outputted. The returned message reveals that:

- The directory owner (Mary) has **R**ead, **W**rite and **eX**ecute permissions.
- Group members can **R**ead and **eX**ecute.
- Others can only **eX**ecute.

This can be tested using Ali and Pete's accounts, which are members of the boys group and not of the girls group, meaning they are considered as "other users".

¹Defined as users that aren't the owner or in an associated group with permissions.



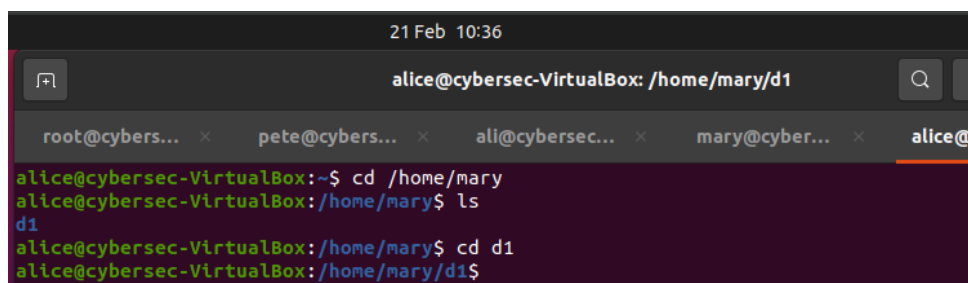
The figure consists of two terminal window screenshots. The top screenshot shows a terminal window titled 'pete@cybersec-...' with tabs for 'root@cybersec-Vir...' and 'pete@cybersec-Vir...'. The user 'pete' is at the prompt 'pete@cybersec-VirtualBox:~\$' and enters 'cd /home/mary'. The prompt changes to 'pete@cybersec-VirtualBox:/home/mary\$' and 'ls' is entered, showing 'd1'. Then 'cd d1' is entered, resulting in the error 'bash: cd: d1: Permission denied'. The bottom screenshot shows a terminal window titled 'ali@cybersec-VirtualBox: /h...' with tabs for 'root@cybersec-Vir...', 'pete@cybersec-Vir...', and 'ali@cybersec...'. The user 'ali' is at the prompt 'ali@cybersec-VirtualBox:~\$' and enters 'cd /home/mary'. The prompt changes to 'ali@cybersec-VirtualBox:/home/mary\$' and 'ls' is entered, showing 'd1'. Then 'cd d1' is entered, resulting in the error 'bash: cd: d1: Permission denied'.

Figure 5.6: Attempting to access D1 as Pete and Ali.

Note

An issue arose here that I didn't have another user in the girls group to test access with. To fix this, I went back and added the existing Alice account from Labs 1 and 2 to the girls group with *sudo usermod -aG girls alice*.

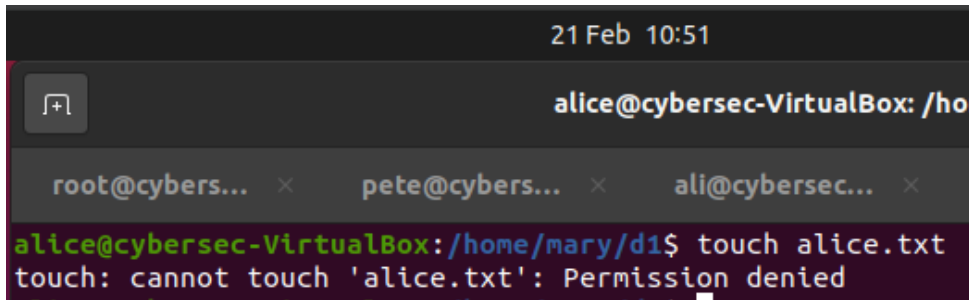
Using another account in the girls group, we can test if other girls can access the directory, which succeeds.



The screenshot shows a terminal window titled 'alice@cybersec-VirtualBox: /home/mary/d1' with tabs for 'root@cybers...', 'pete@cybers...', 'ali@cybersec...', 'mary@cyber...', and 'alice@...'. The user 'alice' is at the prompt 'alice@cybersec-VirtualBox:~\$' and enters 'cd /home/mary'. The prompt changes to 'alice@cybersec-VirtualBox:/home/mary\$' and 'ls' is entered, showing 'd1'. Then 'cd d1' is entered, and the prompt changes to 'alice@cybersec-VirtualBox:/home/mary/d1\$'.

Figure 5.7: Successfully accessing D1 as Alice.

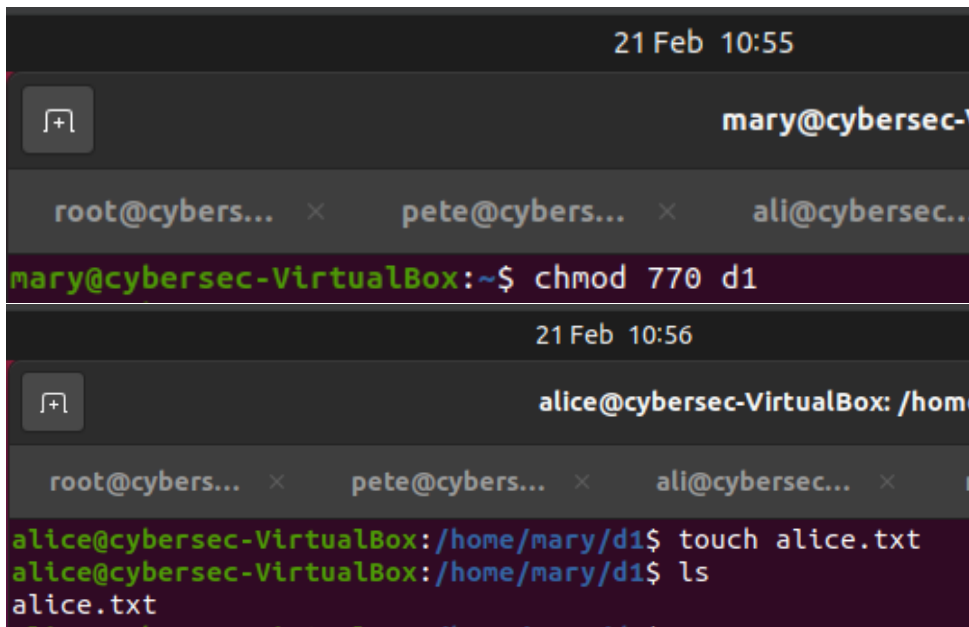
Because of the permissions set, Alice is permitted to read the d1 directory and execute files within it, but she cannot write any files of her own, which is expected behaviour.



```
21 Feb 10:51
alice@cybersec-VirtualBox: /home/mary/d1$ touch alice.txt
touch: cannot touch 'alice.txt': Permission denied
```

Figure 5.8: Failing to write to D1 as Alice.

To test this further, the permissions can then be modified² again to allow girls to write files, which will then allow Alice to make the file.



```
21 Feb 10:55
mary@cybersec-VirtualBox: ~$ chmod 770 d1

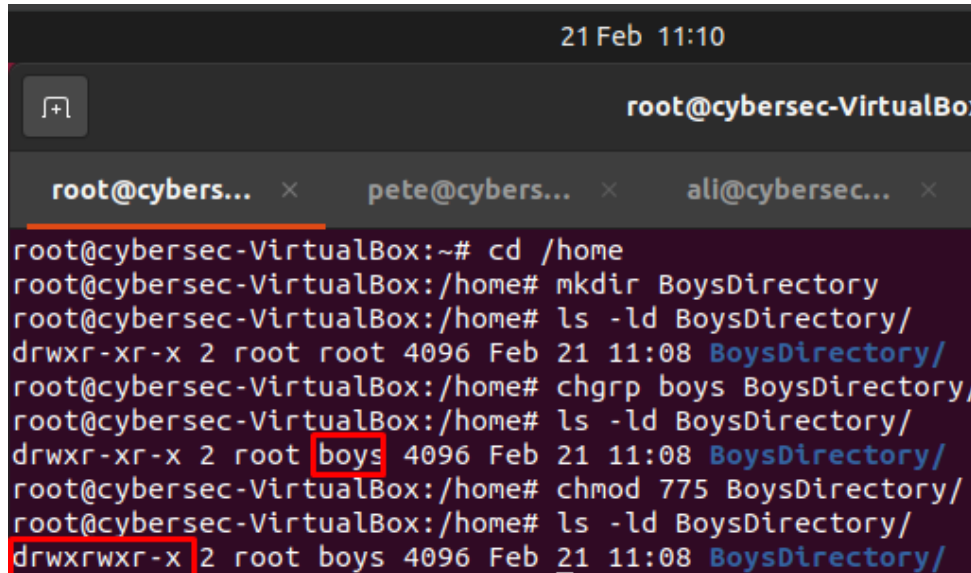
21 Feb 10:56
alice@cybersec-VirtualBox: /home/mary/d1$ touch alice.txt
alice@cybersec-VirtualBox: /home/mary/d1$ ls
alice.txt
```

Figure 5.9: Modifying the permissions of D1, allowing Alice to write the file.

²770 is "rwxrwx—", which correlates to the file owner and members of the group having read, write and execute permissions, but other users have none.

5.2.3 Using chown

Chown is a command similar to chgrp, which assigns a file or directory's ownership to a specific user. For this example, we will create a directory in the shared /home folder and assign group ownership to Boys via chgrp, and using chmod to allow all Boys all permissions, and all other users read & execute permissions.



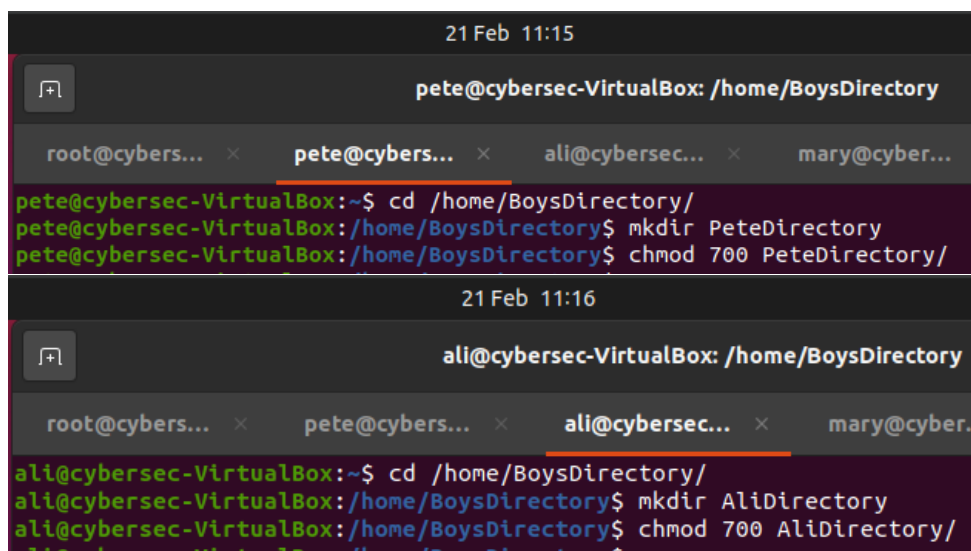
```

21 Feb 11:10
root@cybersec-VirtualBox:~# cd /home
root@cybersec-VirtualBox:/home# mkdir BoysDirectory
root@cybersec-VirtualBox:/home# ls -ld BoysDirectory/
drwxr-xr-x 2 root root 4096 Feb 21 11:08 BoysDirectory/
root@cybersec-VirtualBox:/home# chgrp boys BoysDirectory/
root@cybersec-VirtualBox:/home# ls -ld BoysDirectory/
drwxr-xr-x 2 root boys 4096 Feb 21 11:08 BoysDirectory/
root@cybersec-VirtualBox:/home# chmod 775 BoysDirectory/
root@cybersec-VirtualBox:/home# ls -ld BoysDirectory/
drwxrwxr-x 2 root boys 4096 Feb 21 11:08 BoysDirectory/

```

Figure 5.10: Making BoysDirectory and giving group ownership to Boys.

This new directory can be accessed by all users, but only written to by boys. We can now test chown by making two subdirectories within BoysDirectory, where one will be owned by Pete, and one by Ali. Both users also modify the permissions of their own directories to only be accessible by them.³



```

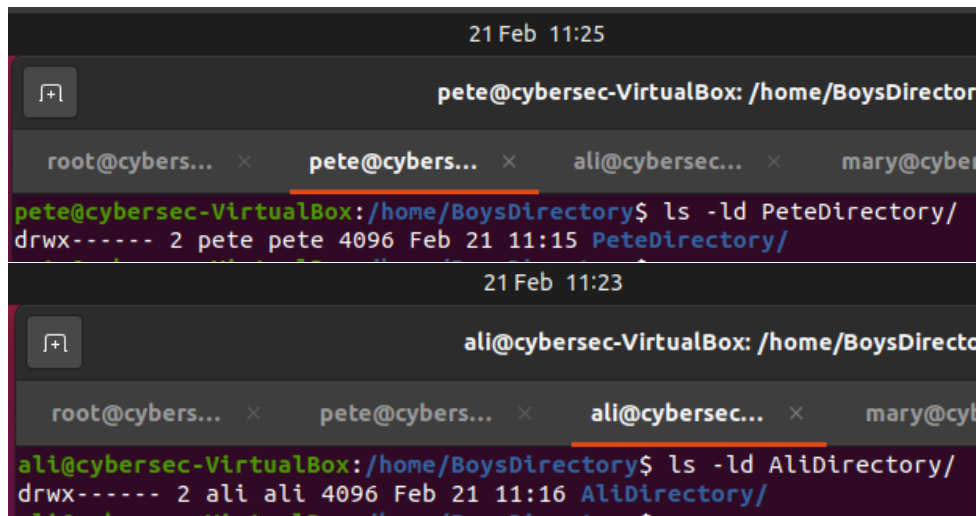
21 Feb 11:15
pete@cybersec-VirtualBox: /home/BoysDirectory
pete@cybersec-VirtualBox:~$ cd /home/BoysDirectory/
pete@cybersec-VirtualBox:/home/BoysDirectory$ mkdir PeteDirectory
pete@cybersec-VirtualBox:/home/BoysDirectory$ chmod 700 PeteDirectory/

21 Feb 11:16
ali@cybersec-VirtualBox: /home/BoysDirectory
ali@cybersec-VirtualBox:~$ cd /home/BoysDirectory/
ali@cybersec-VirtualBox:/home/BoysDirectory$ mkdir AliDirectory
ali@cybersec-VirtualBox:/home/BoysDirectory$ chmod 700 AliDirectory/

```

Figure 5.11: Creating and modifying each user's directories and access permissions.

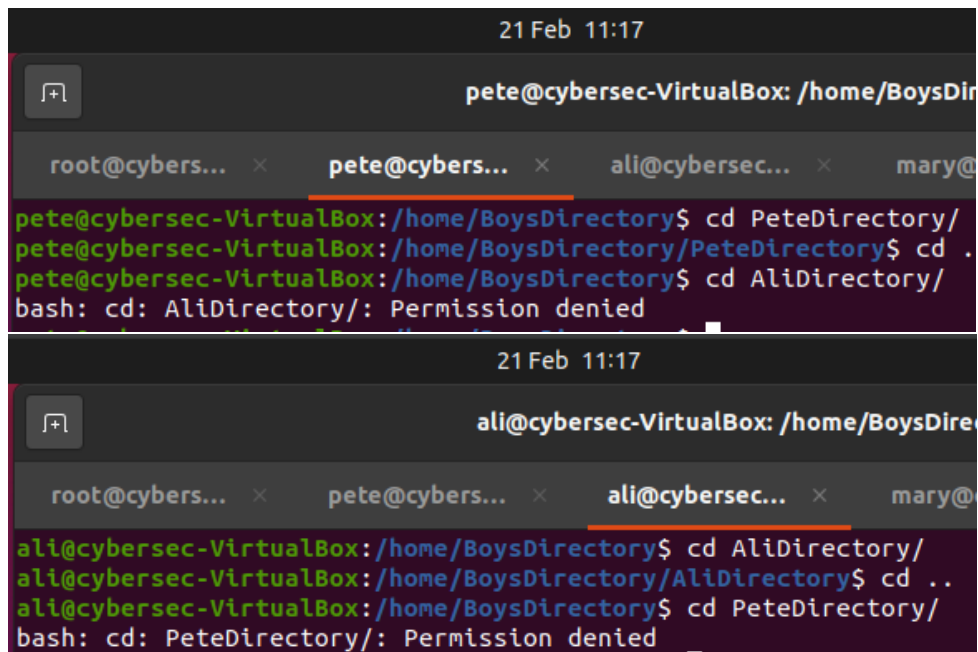
³700 is "rwx—", which means only the owner may read, write and execute.



The figure consists of two terminal window screenshots. The top screenshot shows a terminal for 'pete@cybersec-VirtualBox' at 11:25. The prompt is '/home/BoysDirector'. The terminal shows a tab bar with 'root@cybers...', 'pete@cybers...' (selected), 'ali@cybersec...', and 'mary@cyber...'. The command 'ls -ld PeteDirectory/' is entered, resulting in the output: 'drwx----- 2 pete pete 4096 Feb 21 11:15 PeteDirectory/'. The bottom screenshot shows a terminal for 'ali@cybersec-VirtualBox' at 11:23. The prompt is '/home/BoysDirector'. The terminal shows a tab bar with 'root@cybers...', 'pete@cybers...', 'ali@cybersec...' (selected), and 'mary@cyber...'. The command 'ls -ld AliDirectory/' is entered, resulting in the output: 'drwx----- 2 ali ali 4096 Feb 21 11:16 AliDirectory/'.

Figure 5.12: Viewing the permissions of Pete and Ali's directories.

We can then prove that only the owners of the directories may access them by first accessing their own directory, but then attempting to access the other user's directory:



The figure consists of two terminal window screenshots. The top screenshot shows a terminal for 'pete@cybersec-VirtualBox' at 11:17. The prompt is '/home/BoysDir'. The terminal shows a tab bar with 'root@cybers...', 'pete@cybers...' (selected), 'ali@cybersec...', and 'mary@...'. The commands 'cd PeteDirectory/', 'cd .', and 'cd AliDirectory/' are entered. The output for the last command is 'bash: cd: AliDirectory/: Permission denied'. The bottom screenshot shows a terminal for 'ali@cybersec-VirtualBox' at 11:17. The prompt is '/home/BoysDire'. The terminal shows a tab bar with 'root@cybers...', 'pete@cybers...', 'ali@cybersec...' (selected), and 'mary@...'. The commands 'cd AliDirectory/', 'cd ..', and 'cd PeteDirectory/' are entered. The output for the last command is 'bash: cd: PeteDirectory/: Permission denied'.

Figure 5.13: Successfully accessing their own directory, but failing to access the other because the user isn't the owner.

Lab 6 - Password Cracking

Conclusion

Remember to change the title page to not be yellow!

Check Lab2 Sec 2.1.1, is sudo -s actually the same? Also in Figure 2.4, both of those are the same.

Looking at your header, specifically in Lab 5 as well, it may be worth removing the "Lab 1 -" "Lab 2 -" parts of the chapter names as you already did it using chapter numbering.

Bibliography

- Heinlein, P. (13th Sept. 2016). *OpenSSL Command-Line HOWTO*. URL: <https://www.madboa.com/geek/openssl/#how-do-i-simply-encrypt-a-file> (visited on 24/01/2024).
- Kolletzki, S. (1996). ‘Secure internet banking with privacy enhanced mail — A protocol for reliable exchange of secured order forms’. In: *Computer Networks and ISDN Systems* 28 (14), pp. 1891–1899. DOI: [https://doi.org/10.1016/S0169-7552\(96\)00089-X](https://doi.org/10.1016/S0169-7552(96)00089-X).