Computer Security – CSCI_6531 – Homework 2

Namana Y Tarikere – G21372717

1. Password Cracking

1.1 Setup (10pts)

Install Kali Linux as a virtual machine. Find Kali here: https://www.kali.org/get-kali/

For x86 Windows, x86AppleHardare, and x86 Linux:
I recommend using VirtualBox as a VM host: https://www.virtualbox.org/wiki/Downloads
Kali offers pre-built images for VirtualBox. It supports x86/AMD64 Mac, Windows, and Linux. M1/M2 Mac (and other ARM processors such as Raspberry Pi) are not supported by these images.

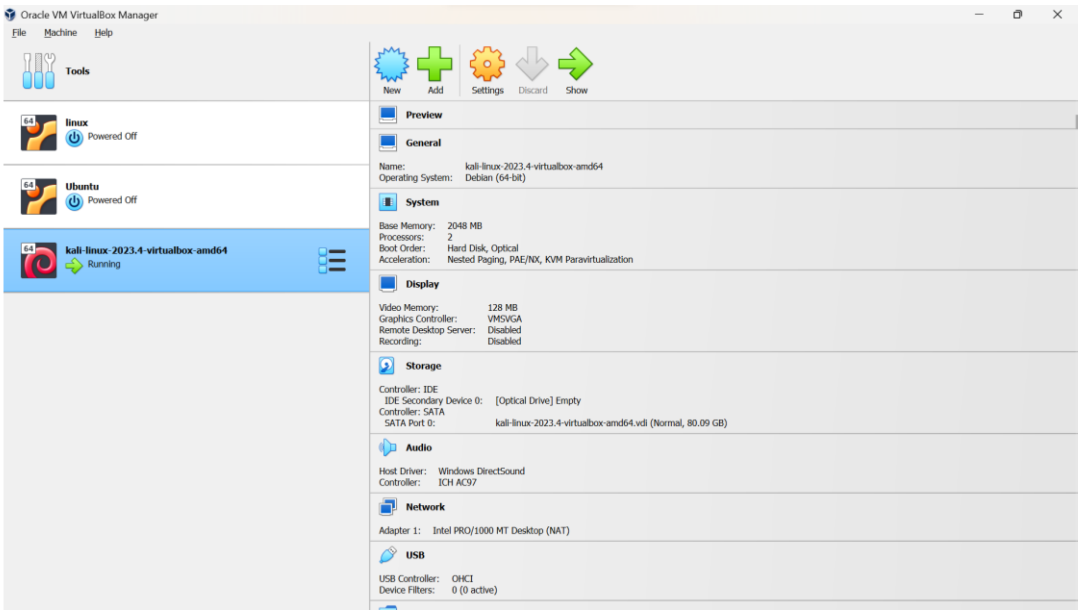
For M1/M2 Apple:
I recommend using VMWare Fusion as VM Host. There are no pre-built images for M1/M2 Apple Silicon, however you can install Kali using the following procedure:

- Install VMWare Fusion Player (free version) for Apple Silicon (https://vmware.com). This requires a (free) account with VMWare.
- Create a new Debian 1064-bit ARM VM in VMWare
- Mount the Kali Linux **Apple Silicon (ARM64) Installer** ISO (you cannot use an x86 ISO) to the CD/DVD drive in VMWare Fusion: https://www.kali.org/get-kali/#kali-installer-images
- Install Kali Linux through the GRUB installer, selecting the default options

For all hardware configurations: You are free to use different VM hosts than my suggestions (QEMU, Parallels, or e.g., VirtualBox on Apple M1, etc.); these suggestions are simply the most straightforward way to get Kali Linux running as a VM. Whatever VM host you use must support networking multiple VMs.

To receive credit for this part of the assignment: take a screenshot of your running Kali Linux installation. **Alternatively:** you may use another security-focused distribution if you are familiar with it. Take a screenshot of this distribution running in a virtual machine.

<u>**Solution:**</u> I am using a HP laptop with Windows 11 pro Operating System. I already had the Oracle VM virtual box set up in my system for Ubuntu, I installed the Kali Linux on the same Oracle VM virtual box as seen in the screenshot below.



linux
Powered Off

Ubuntu
Powered Off

kali-linux-2023.4-virtualbox-amd64
Running

Preview

General

```
Name:              kali-linux-2023.4-virtualbox-amd64
Operating System:  Debian (64-bit)
```

System

```
Base Memory:   2048 MB
Processors:    2
Boot Order:    Hard Disk, Optical
Acceleration:  Nested Paging, PAE/NX, KVM Paravirtualization
```

Display

```
Video Memory:            128 MB
Graphics Controller:     VMSVGA
Remote Desktop Server:   Disabled
Recording:               Disabled
```

Storage

```
Controller: IDE
  IDE Secondary Device 0:   [Optical Drive] Empty
Controller: SATA
  SATA Port 0:              kali-linux-2023.4-virtualbox-amd64.vdi (Normal, 80.09 GB)
```

Audio

```
Host Driver:   Windows DirectSound
Controller:    ICH AC97
```

Network

```
Adapter 1:   Intel PRO/1000 MT Desktop (NAT)
```

USB

```
USB Controller:   OHCI
Device Filters:   0 (0 active)
```

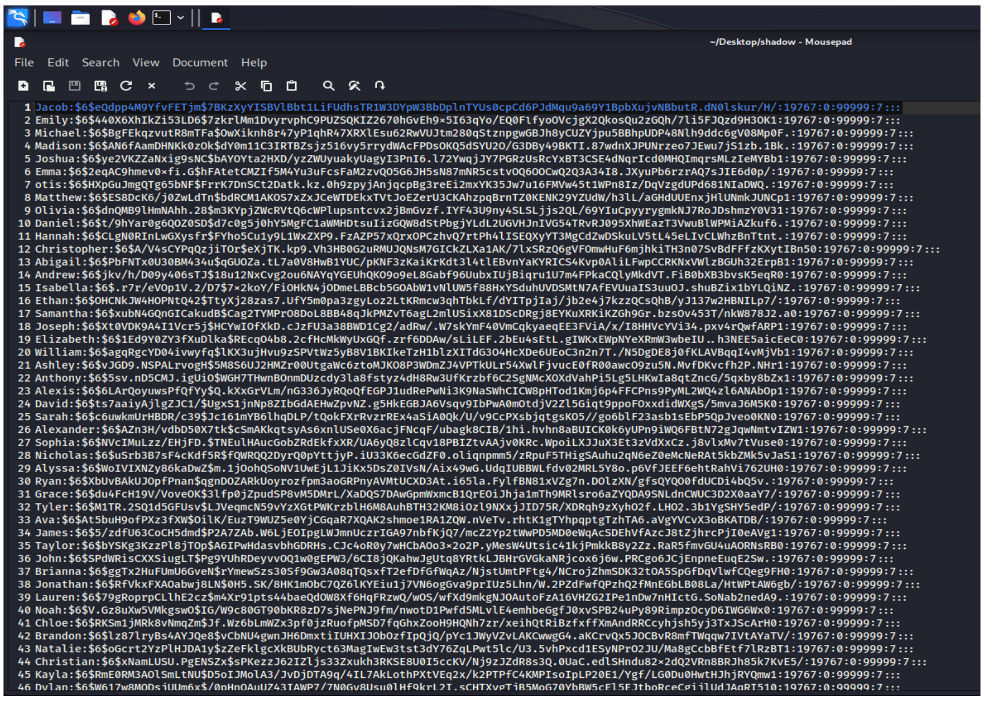
In the screenshot below, you can see the home screen of Kali Linux setup and running in my laptop.

-------------------------o-------------------------o-------------------------o-------------------------

1.2 Shadow File Analysis (10 pts)

Included in this assignment is a 'shadow' file containing numerous passwords associated with various user accounts. Examine the file's text (do not use any software tools beyond a text editor). Are you able to determine anything about the plaintext of the passwords from examining the file?

<u>Solution:</u> In the below screenshot, you can see the shadow file in a note pad with 1 user (Jacob) highlighted.


~/Desktop/shadow - Mousepad
File Edit Search View Document Help

```
Jacob:$6$eQdpp4M9YfvFETjm$7BKzXyYISBVlBbt1LiFUdhsTR1W3DYpW3BbDplnTYUs0cpCd6PJdMqu9a69Y1BpbXujvNBbutR.dN0lskur/H/:19767:0:99999:7:::
Emily:$6$440X6XhIkZi53LD6$7zkrlMm1DvyrvphC9PUZSQKIZ2670hGvEh9x5I63qYo/EQ0FlfyoOVcjgX2QkosQu2zGQh/7li5FJQzd9H3OK1:19767:0:99999:7:::
Michael:$6$BgFEkqzvutR8mTFa$OwXiknh8r47yP1qhR47XRXlEsu62RwVUJtm280qStznpgwGBJh8yCUZYjpu5BBhpUDP48Nlh9ddc6gV08Mp0F.:19767:0:99999:7:::
Madison:$6$AN6fAamDHNKk0zOk$dY0m11C3IRTBZsjz516vy5rrydWAcFPDsOKQ5dSYU2O/G3DBy49BKTI.87wdnXJPUNrzeo7JEwu7jS1zb.1Bk.:19767:0:99999:7:::
Joshua:$6$ye2VKZZaNxig9sNC$bAYOYta2HXD/yzZWUyuakyUagyI3PnI6.l72YwqjJY7PGRzUsRcYxBT3CSE4dNqrIcd0MHQImqrsMLzIeMYBb1:19767:0:99999:7:::
Emma:$6$2eqAC9hmev0xfi.G$hFAtetCMZIf5M4Yu3uFcsFaM2zvQO5G6JH5sN87mNR5cstvOQ6OOCwQ2Q3A34I8.JXyuPb6rzrAQ7sJIE6d0p/:19767:0:99999:7:::
otis:$6$HXpGuJmgQTg65bNF$FrrK7DnSCt2Datk.kz.0h9zpyjAnjqcpBg3reEi2mxYK35Jw7u16FMVw45t1WPn8Iz/DqVzgdUPd681NIaDWQ.:19767:0:99999:7:::
Matthew:$6$ES8DcK6/j0ZwLdTn$bdRCM1AKOS7xZxJCeWTDEkxTVtJoEZerU3CKAhzpqBrnTZ0KENK29YZUdW/h3lL/aGHdUUEnxjHlUNmkJUNCp1:19767:0:99999:7:::
Olivia:$6$dnQMB9lHmNAhh.28$m3KYpjZWcRVtQ6cWPlupsntcvx2jBmGvzf.IYF43U9ny4SLSLjjs2QL/69YIuCpyyrygmkNJ7RoJDshmzY0V31:19767:0:99999:7:::
Daniel:$6$t/9hYar0g6QOZ0SD$d7c0g5j0hY5MgFC1aWMHDtsuIizGQW8dStPbgjYLdL2UGVHJnIVG54TRvRJ095XhWEazT3VwuBlWPMiAZkuf6.:19767:0:99999:7:::
Hannah:$6$CLgN0RInLwGXysfr$FYho5Cu1y9L1WxZXP9.FzAZP57xQrxOPCzhvQ7rtPh4lISEQXyYT3MgCdZwDSkuLV5tL45eLIvCLWhzBnTtnt.:19767:0:99999:7:::
Christopher:$6$A/V4sCYPqQzjiTOr$eXjTK.kp9.Vh3HB0G2uRMUJQNsM7GICkZLXa1AK/7lxSRzQ6gVFOmwHuF6mjhkiTH3n07SvBdFFfzKXytIBn50:19767:0:99999:7:::
Abigail:$6$PbFNTx0U30BM434u$qGUOZa.tL7a0V8HwB1YUC/pKNF3zKaiKrKdt3l4tlEBvnYaKYRICS4Kvp0AliLFwpCCRKNxVWlzBGUh32ErpB1:19767:0:99999:7:::
Andrew:$6$jkv/h/D09y406sTJ$18u12NxCvg2ou6NAYqYGEUhQKO9o9eL8Gabf96UubxIUjBiqru1U7m4FPkaCQlyMkdVT.FiB0bXB3bvsK5eqR0:19767:0:99999:7:::
Isabella:$6$.r7r/eVOp1V.2/D7$7x2koY/FiOHkN4jODmeLBBcb5GOAbW1vNlUW5f88HxYSduhUVDSMtN7AfEVUuaIS3uuOJ.shuBZix1bYLQiNZ.:19767:0:99999:7:::
Ethan:$6$OHCNkJW4HOPNtQ42$TtyXj28zas7.UfY5m0pa3zgyLoz2LtKRmcw3qhTbkLf/dYITpjIaj/jb2e4j7kzzQCsQhB/yJ137w2HBNILp7/:19767:0:99999:7:::
Samantha:$6$xubN4GQnGICakudB$Cag2TYMPrO8DoL8BB48qJkPMZvT6agL2mlUSixX81DScDRgj8EYKuXRKiKZGh9Gr.bzsOv453T/nkW878J2.a0:19767:0:99999:7:::
Joseph:$6$Xt0VDK9A4I1Vcr5j$HCYwIOfXkD.cJzFU3a38BWD1Cg2/adRw/.W7skYmF40VmCqkyaeqEE3FViA/x/I8HHVcYVi34.pxv4rQwfARP1:19767:0:99999:7:::
Elizabeth:$6$1Ed9Y0ZY3fXuDlka$REcqO4b8.2cfHcMkWyUxGQf.zrf6DDAw/sLiLEF.2bEu4sEtL.gIWKxEWpNYeXRmW3wbeIU..h3NEE5aicEeC0:19767:0:99999:7:::
William:$6$agqRgcYD04ivwyfq$lKX3ujHvu9zSPVtWz5yB8V1BKIkeTzH1blzXITdG3O4HcXDe6UEoC3n2n7T./N5DgDE8j0fKLAVBqqI4vMjVb1:19767:0:99999:7:::
Ashley:$6$vJGD9.NSPALrvogH$5M8S6UJ2HMZr00UtgaWc6ztoMJKO8P3WDmZJ4VPTkULr54XwlFjvucE0fR00awcO9zu5N.MvfDKvcfh2P.NHr1:19767:0:99999:7:::
Anthony:$6$5sv.nD5CMJ.igUiO$WGH7THwnBOnmDUzcdy3la8fstyz4dH8Rw3UfKrzbf6C2SgNMcXOXdVahPi5Lg5LHKwIa8qtZncG/5qxby8bZx1:19767:0:99999:7:::
Alexis:$6$6LArQoyuwsPfQfYy$Q.kXxGrVLm/nG336JyRQoQfEGPJ1udRePwNi3K9NaSWhCICW8pHTod1Kmj6p4FFCPns9PyML2WQ4zl6ANAbOp1:19767:0:99999:7:::
David:$6$ts7aaiyAjlgZJC1/$UgxS1jnNp8ZIbGdAEHwZpvNZ.g5HkEGBJA6Vsqv9IbPwA0mOtdjV2Zl5Giqt9ppoFOxxdidWXgS/5mvaJ6M5K0:19767:0:99999:7:::
Sarah:$6$c6uwkmUrHBDR/c39$Jc161mYB6lhqDLP/tQokFXrRvzrREx4aSiA0Qk/U/v9CcPXsbjqtgsKO5//go6blF23asb1sEbP5QpJveo0KN0:19767:0:99999:7:::
Alexander:$6$AZn3H/vdbD50X7tk$cSmAKkqtsyAs6xnlUSe0X6acjFNcqF/ubagk8CIB/1hi.hvhn8aBUICK0k6yUPn9iWQ6FBtN72gJqwNmtvIZW1:19767:0:99999:7:::
Sophia:$6$NVcIMuLzz/EHjFD.$TNEulHAucGobZRdEkfxXR/UA6yQ8zlCqv18PBIZtvAAjv0KRc.WpoiLXJJuX3Et3zVdXxCz.j8vlxMv7tVuse0:19767:0:99999:7:::
Nicholas:$6$uSrb3B7sF4cKdf5R$fQWRQQ2DyrQ0pYttjyP.iU33K6ecGdZF0.oliqnpmm5/zRpuF5THigSAuhu2qN6eZ0eMcNeRAt5kbZMk5vJaS1:19767:0:99999:7:::
Alyssa:$6$WoIVIXNZy86kaDwZ$m.1jOohQSoNV1UwEjL1JiKx5DsZ0IVsN/Aix49wG.UdqIUBBWLfdv02MRL5Y8o.p6VfJEEF6ehtRahVi762UH0:19767:0:99999:7:::
Ryan:$6$XbUvBAkUJOpfPnan$qgnDOZARkUoyrozfpm3aoGRPnyAVMtUCXD3At.i65la.FylfBN81xVZg7n.DOlzXN/gfsQYQO0fdUCDi4bQ5v.:19767:0:99999:7:::
Grace:$6$du4FcH19V/VoveOK$3lfp0jZpudSP8vM5DMrL/XaDQS7DAwGpmWxmcB1QrEOiJhja1mTh9MRlsro6aZYQDA9SNLdnCWUC3D2X0aaY7/:19767:0:99999:7:::
Tyler:$6$M1TR.2SQ1d5GFUsv$LJVeqmcN59vYzXGtPWKrzblH6M8AuhBTH32KM8iOzl9NXxjJID75R/XDRqh9zXyhO2f.LHO2.3b1YgSHY5edP/:19767:0:99999:7:::
Ava:$6$At5buH9ofPXz3fXW$OilK/EuzT9WUZ5e0YjCGqaR7XQAK2shmoe1RA1ZQW.nVeTv.rhtK1gTYhpqptgTzhTA6.aVgYVCvX3oBKATDB/:19767:0:99999:7:::
James:$6$5/zdfU63CoCH5dmd$P2A7ZAb.W6LjEOIpgLWJmnUczrIGA97nbfKjQ7/mcZ2Yp2tWwPD5MD0eWqAcSDEhVfAzcJ8tZjhrcPjI0eAVg1:19767:0:99999:7:::
Taylor:$6$bYSKg3KzzPl8jTOp$A6IPwHdasvbhGDRHs.CJc4oR0y7wHCbAOo3x2o2P.yMesW4Utsic41kjPmkkB8y2Zz.RaR5fmvGU4uAORNsRB0:19767:0:99999:7:::
John:$6$SPdWRisCXXSiugLT$Pg9YUhRDeyvvOQ1w0gEPW3/6CI8jQKahwJgUtq8YRtkLJBHrGVGkaNRjcox6j6w.PRCgo6JCjEnpneEuqE2Sw.:19767:0:99999:7:::
Brianna:$6$ggTx2HuFUmU6GveN$rYmewSzs30Sf9Gw3A08qTQsxfT2efDfGfWqAz/NjstUmtPFtg4/NCrojZhmSDK32tOA5SpGfDqVlwfCQeg9FH0:19767:0:99999:7:::
Jonathan:$6$RfVkxFXAOabwj8LN$0H5.SK/8HK1mObC7QZ6lKYEiu1j7VN6ogGva9prIUz5Lhn/W.2PZdFwfQPzhQ2fMnEGbLB08La/HtWPtAW6gb/:19767:0:99999:7:::
Lauren:$6$79gRoprpCLlhE2cz$m4Xr91pts44baeQdOW8Xf6HqFRzwQ/wOS/wfXd9mkgNJOAutoFzA16VHZG2IPe1nDw7nHIctG.SoNab2nedA9.:19767:0:99999:7:::
Noah:$6$V.Gz8uXw5VMkgswO$IG/W9c80GT90bKR8zD7sjNePNJ9fm/nwotD1Pwfd5MLvlE4emhbeGgfJ0xvSPB24uPy89RimpzOcyD6IWG6Wx0:19767:0:99999:7:::
Chloe:$6$RKSm1jMRk8vNmqZm$Jf.Wz6bLmWZx3pf0jzRuofpMSD7fqGhxZooH9HQNh7zr/xeihQtRiBzfxffXmAndRRCcyhjsh5yj3TxJScArH0:19767:0:99999:7:::
Brandon:$6$lz87lryBs4AYJQe8$vCbNU4gwnJH6DmxtiIUHXIJObOzfIpQjQ/pYc1JWyVZvLAKCwwgG4.aKCrvQx5JOCBvR8mfTWqqw7IVtAYaTV/:19767:0:99999:7:::
Natalie:$6$oGcrt2YzPlHJDA1y$zZeFklgcXkBUbRyct63MagIwEw3tst3dY76ZqLPwt5lc/U3.5vhPxcd1ESyNPrO2JU/Ma8gCcbBfEtf7lRzBT1:19767:0:99999:7:::
Christian:$6$xNamLUSU.PgENSZx$sPKezzJ62IZljs33Zxukh3RKSE8U0I5ccKV/Nj9zJZdR8s3Q.0UaC.edlSHndu82x2dQ2VRn8BRJh85k7KvE5/:19767:0:99999:7:::
Kayla:$6$RmE0RM3AOlSmLtNU$D5oIJMolA3/JvDjDTA9q/4IL7AkLothPXtVEq2x/k2PTPfC4KMPIsoIpLP20E1/Ygf/LG0Du0HwtHJhjRYQmw1:19767:0:99999:7:::
Dylan:$6$W617w8MODsjUUm6x$/0pHnOAuUZ43IAWP7/7N0Gv8Usu0lHf9krL2I.sCHTXvgTjB5MoG70YbBW5cEl5EJtboRceCgiilUdJAgRI510:19767:0:99999:7:::
```

The shadow file consists of list of users with their username, hashing algorithm used, salt, the hash code for the password, password age and other related information [1] as shown below for user Jacob:

Jacob:$6$eQdpp4M9YfvFETjm$7BKzXyYISBVlBbt1LiFUdhsTR1W3DYpW3BbDplnTYUs0cpCd6PJdMqu9a69Y1BpbXujvNBbutR.dN0lskur/H/:19767:0:99999:7:::

From the above example, you can see that there are 7 parameters involved for hashing the password for each user and these parameters are separated by the **"colon (:)"** sign. The parameters are involved are:

1. Jacob is the **username**,
2. $6 indicates that it is encrypted with **SHA-512** hashing algorithm,
3. $eQdpp4M9YfvFETjm is the **salt** used for hashing the password to avoid dictionary attacks,
4. 7BKzXyYISBVlBbt1LiFUdhsTR1W3DYpW3BbDplnTYUs0cpCd6PJdMqu9a69Y1BpbXujvNBbutR.dN0lskur/H/ is the **encrypted hash code** for the password,
5. 19767 is the value when the user **last changed** the password,
6. 0:99999 are the **minimum and maximum age** of the password which are 0 and 99999 respectively,
7. 7::: represents user **account warning, inactivity, expiration and reservations** related details.

I noticed in the shadow file that the passwords are encrypted with SHA-512 hashing algorithm and the password last changed, age, account activity information are the same for all users. I observed that there are some readable plaintexts in the encrypted hash code of the passwords in the shadow file, but it is difficult to determine any patterns in the plaintext of the passwords by just using a text editor.

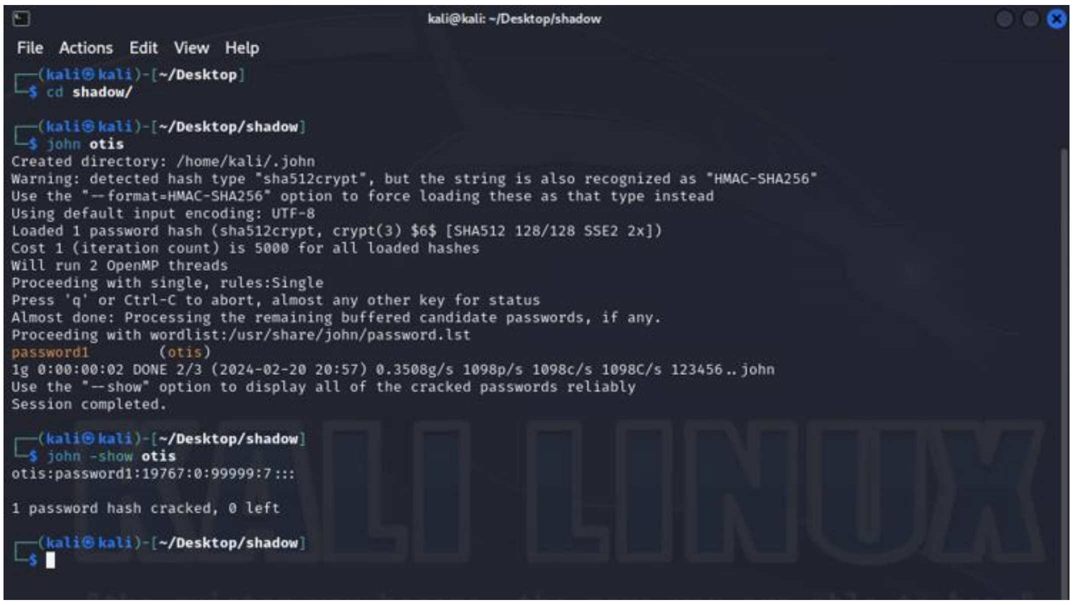
------------------------o------------------------o------------------------o------------------------

1.3 Single Password (10 pts)

Crack the password for the user otis.

<u>Solution:</u>

The user otis is mentioned twice in the shadow file with the same hashing details and hence has the same password. I copied the information of only otis to a separate file called otis and ran John the Ripper on the file to crack the password efficiently. I obtained the password for user **otis** to be **password1** as shown below:



```
┌──(kali㉿kali)-[~/Desktop]
└─$ cd shadow/

┌──(kali㉿kali)-[~/Desktop/shadow]
└─$ john otis
Created directory: /home/kali/.john
Warning: detected hash type "sha512crypt", but the string is also recognized as "HMAC-SHA256"
Use the "--format=HMAC-SHA256" option to force loading these as that type instead
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:/usr/share/john/password.lst
password1        (otis)
1g 0:00:00:02 DONE 2/3 (2024-02-20 20:57) 0.3508g/s 1098p/s 1098c/s 1098C/s 123456..john
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(kali㉿kali)-[~/Desktop/shadow]
└─$ john -show otis
otis:password1:19767:0:99999:7:::

1 password hash cracked, 0 left

┌──(kali㉿kali)-[~/Desktop/shadow]
└─$ █
```

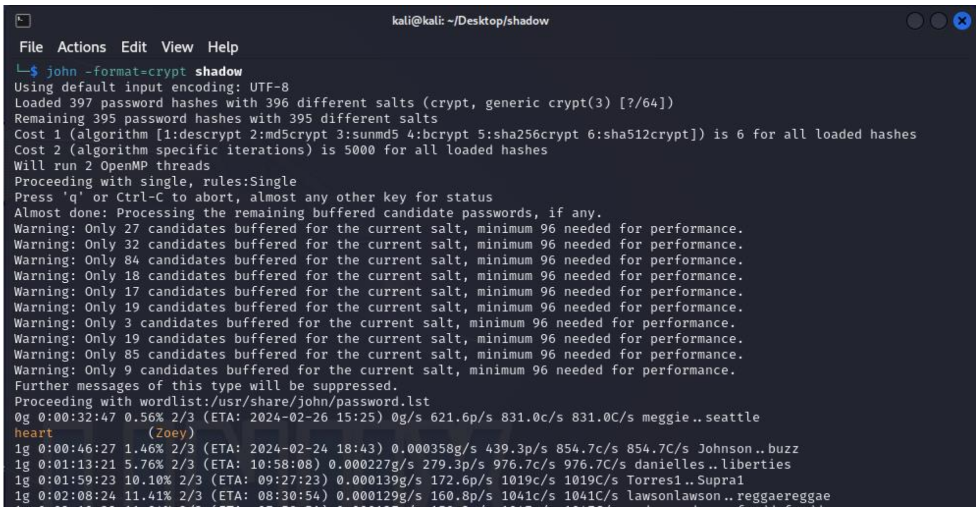
------------------------o------------------------o------------------------o------------------------

1.4 Multiple Passwords (35 pts)

Crack passwords for three more users (four total). Some of the passwords are more complicated than others. Some of the answers are right in front of your nose. When in doubt, ask! Successful methods will vary. The process might last a long time. You may need to 'mangle' a dictionary to succeed. A larger dictionary may not make your process easier. Comment on what features of password plaintext made passwords easier or harder to crack. Social engineering hint: I, your professor, made this assignment, and I made it deliberately feasible for you to be able to complete.

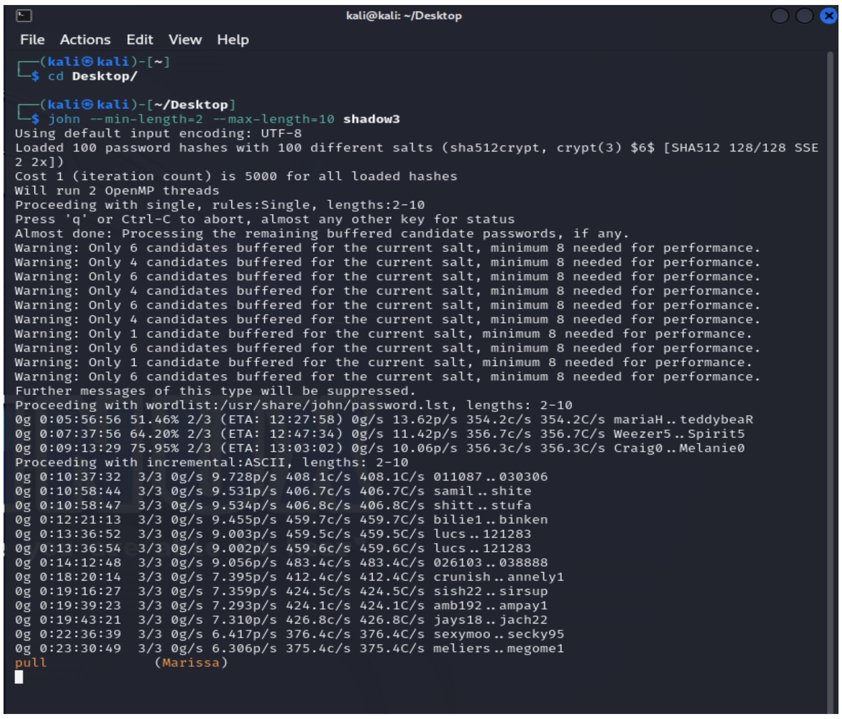
Solution:

I performed an **exhaustive brute force method** of John the Ripper [2] on the given shadow file for 2 days and was able to crack the password of only one user **Zoey** which is **heart** as shown in the below screenshot:



```
└─$ john -format=crypt shadow
Using default input encoding: UTF-8
Loaded 397 password hashes with 396 different salts (crypt, generic crypt(3) [?/64])
Remaining 395 password hashes with 395 different salts
Cost 1 (algorithm [1:descrypt 2:md5crypt 3:sunmd5 4:bcrypt 5:sha256crypt 6:sha512crypt]) is 6 for all loaded hashes
Cost 2 (algorithm specific iterations) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 27 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 32 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 84 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 18 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 17 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 19 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 3 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 19 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 85 candidates buffered for the current salt, minimum 96 needed for performance.
Warning: Only 9 candidates buffered for the current salt, minimum 96 needed for performance.
Further messages of this type will be suppressed.
Proceeding with wordlist:/usr/share/john/password.lst
0g 0:00:32:47 0.56% 2/3 (ETA: 2024-02-26 15:25) 0g/s 621.6p/s 831.0c/s 831.0C/s meggie..seattle
heart            (Zoey)
1g 0:00:46:27 1.46% 2/3 (ETA: 2024-02-24 18:43) 0.000358g/s 439.3p/s 854.7c/s 854.7C/s Johnson..buzz
1g 0:01:13:21 5.76% 2/3 (ETA: 10:58:08) 0.000227g/s 279.3p/s 976.7c/s 976.7C/s danielles..liberties
1g 0:01:59:23 10.10% 2/3 (ETA: 09:27:23) 0.000139g/s 172.6p/s 1019c/s 1019C/s Torres1..Supra1
1g 0:02:08:24 11.41% 2/3 (ETA: 08:30:54) 0.000129g/s 160.8p/s 1041c/s 1041C/s lawsonlawson..reggaereggae
```

Since it was taking a lot of time to crack passwords using the exhaustive brute force method, I switched my approach to **single crack method**, divided the shadow file into 4 equal parts (1, 2, 3, and 4) and limited the passwords length to be between 2 and 10 while executing John the Ripper on the third part of the split shadow file [3]. I cracked the password for user **Marissa** which is **pull** as shown in the screenshot below:

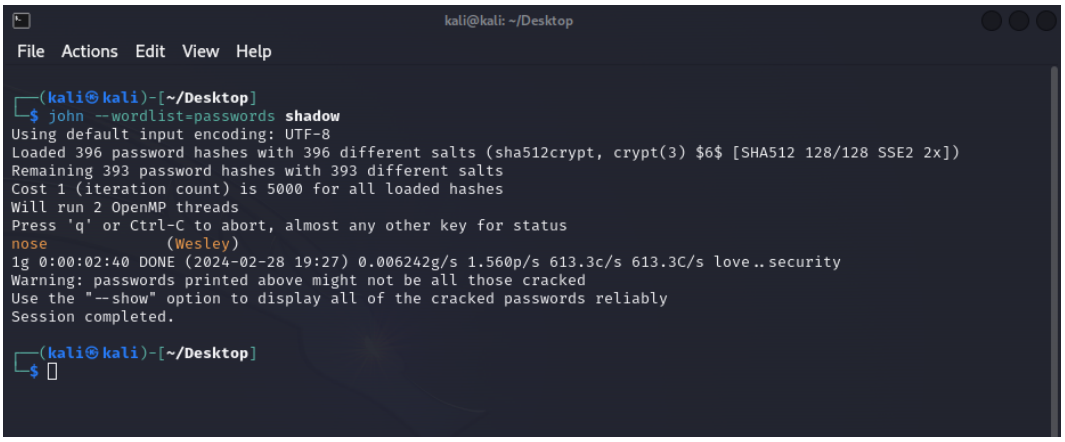


```
┌──(kali㉿kali)-[~]
└─$ cd Desktop/

┌──(kali㉿kali)-[~/Desktop]
└─$ john --min-length=2 --max-length=10 shadow3
Using default input encoding: UTF-8
Loaded 100 password hashes with 100 different salts (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE
2 2x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Proceeding with single, rules:Single, lengths:2-10
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 4 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 1 candidate buffered for the current salt, minimum 8 needed for performance.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed for performance.
Warning: Only 1 candidate buffered for the current salt, minimum 8 needed for performance.
Warning: Only 6 candidates buffered for the current salt, minimum 8 needed for performance.
Further messages of this type will be suppressed.
Proceeding with wordlist:/usr/share/john/password.lst, lengths: 2-10
0g 0:05:56:56 51.46% 2/3 (ETA: 12:27:58) 0g/s 13.62p/s 354.2c/s 354.2C/s mariaH..teddybeaR
0g 0:07:37:56 64.20% 2/3 (ETA: 12:47:34) 0g/s 11.42p/s 356.7c/s 356.7C/s Weezer5..Spirit5
0g 0:09:13:29 75.95% 2/3 (ETA: 13:03:02) 0g/s 10.06p/s 356.3c/s 356.3C/s Craig0..Melanie0
Proceeding with incremental:ASCII, lengths: 2-10
0g 0:10:37:32  3/3 0g/s 9.728p/s 408.1c/s 408.1C/s 011087..030306
0g 0:10:58:44  3/3 0g/s 9.531p/s 406.7c/s 406.7C/s samil..shite
0g 0:10:58:47  3/3 0g/s 9.534p/s 406.8c/s 406.8C/s shitt..stufa
0g 0:12:21:13  3/3 0g/s 9.455p/s 459.7c/s 459.7C/s bilie1..binken
0g 0:13:36:52  3/3 0g/s 9.003p/s 459.5c/s 459.5C/s lucs..121283
0g 0:13:36:54  3/3 0g/s 9.002p/s 459.6c/s 459.6C/s lucs..121283
0g 0:14:12:48  3/3 0g/s 9.056p/s 483.4c/s 483.4C/s 026103..038888
0g 0:18:20:14  3/3 0g/s 7.395p/s 412.4c/s 412.4C/s crunish..annely1
0g 0:19:16:27  3/3 0g/s 7.359p/s 424.5c/s 424.5C/s sish22..sirsup
0g 0:19:39:23  3/3 0g/s 7.293p/s 424.1c/s 424.1C/s amb192..ampay1
0g 0:19:43:21  3/3 0g/s 7.310p/s 426.8c/s 426.8C/s jays18..jach22
0g 0:22:36:39  3/3 0g/s 6.417p/s 376.4c/s 376.4C/s sexymoo..secky95
0g 0:23:30:49  3/3 0g/s 6.306p/s 375.4c/s 375.4C/s meliers..megome1
pull             (Marissa)
```

Since I obtained only 2 passwords along with otis and both methods were consuming too much time, I changed my approach to build a custom wordlist called passwords with all the possible words from my class notes, ppt, words and clues from the problem statement, commonly found birds, celebrities, fruits, food, colours, some password patterns, and some of the most used passwords from [4]. I executed a dictionary attack on the shadow file using John the Ripper tool with this custom file [5] [6] and cracked the passwords for 4 more users **Wesley, Conner, Jazmin and Edwin** which are **nose, last, drop and vary** respectively as shown in the below screenshots:

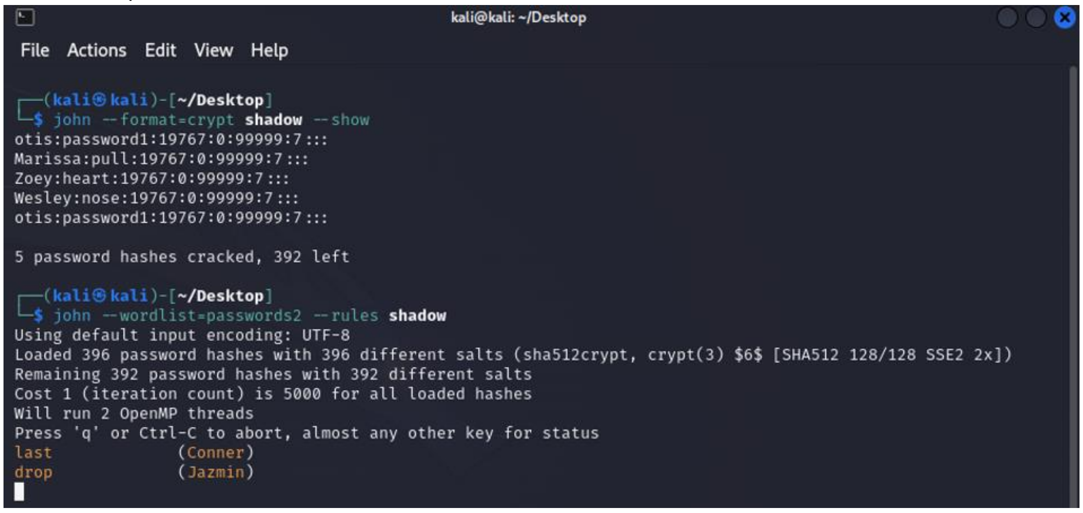
Wesley: nose



```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --wordlist=passwords shadow
Using default input encoding: UTF-8
Loaded 396 password hashes with 396 different salts (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Remaining 393 password hashes with 393 different salts
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
nose             (Wesley)
1g 0:00:02:40 DONE (2024-02-28 19:27) 0.006242g/s 1.560p/s 613.3c/s 613.3C/s love..security
Warning: passwords printed above might not be all those cracked
Use the "--show" option to display all of the cracked passwords reliably
Session completed.

┌──(kali㉿kali)-[~/Desktop]
└─$
```

Conner: last
Jazmin: drop

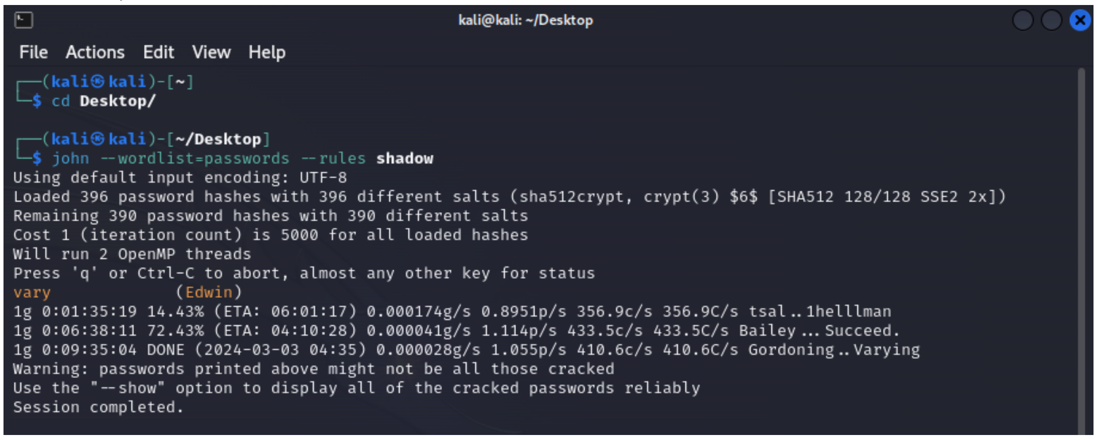


```
┌──(kali㉿kali)-[~/Desktop]
└─$ john --format=crypt shadow --show
otis:password1:19767:0:99999:7:::
Marissa:pull:19767:0:99999:7:::
Zoey:heart:19767:0:99999:7:::
Wesley:nose:19767:0:99999:7:::
otis:password1:19767:0:99999:7:::

5 password hashes cracked, 392 left

┌──(kali㉿kali)-[~/Desktop]
└─$ john --wordlist=passwords2 --rules shadow
Using default input encoding: UTF-8
Loaded 396 password hashes with 396 different salts (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Remaining 392 password hashes with 392 different salts
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
last             (Conner)
drop             (Jazmin)
```

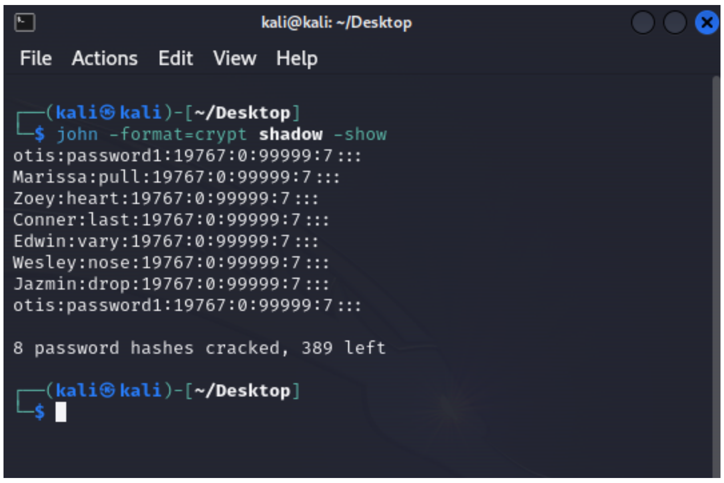
Edwin: vary



```
┌──(kali㉿kali)-[~]
└─$ cd Desktop/

┌──(kali㉿kali)-[~/Desktop]
└─$ john --wordlist=passwords --rules shadow
Using default input encoding: UTF-8
Loaded 396 password hashes with 396 different salts (sha512crypt, crypt(3) $6$ [SHA512 128/128 SSE2 2x])
Remaining 390 password hashes with 390 different salts
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
vary             (Edwin)
1g 0:01:35:19 14.43% (ETA: 06:01:17) 0.000174g/s 0.8951p/s 356.9c/s 356.9C/s tsal..1helllman
1g 0:06:38:11 72.43% (ETA: 04:10:28) 0.000041g/s 1.114p/s 433.5c/s 433.5C/s Bailey...Succeed.
1g 0:09:35:04 DONE (2024-03-03 04:35) 0.000028g/s 1.055p/s 410.6c/s 410.6C/s Gordoning..Varying
Warning: passwords printed above might not be all those cracked
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

I was able to crack passwords for 6 users apart from otis and the users along with their cracked passwords can be seen in the below screenshot:



```
┌──(kali㉿kali)-[~/Desktop]
└─$ john -format=crypt shadow -show
otis:password1:19767:0:99999:7:::
Marissa:pull:19767:0:99999:7:::
Zoey:heart:19767:0:99999:7:::
Conner:last:19767:0:99999:7:::
Edwin:vary:19767:0:99999:7:::
Wesley:nose:19767:0:99999:7:::
Jazmin:drop:19767:0:99999:7:::
otis:password1:19767:0:99999:7:::

8 password hashes cracked, 389 left

┌──(kali㉿kali)-[~/Desktop]
└─$ 
```

Therefore, to summarize, the total number of users for which I was able to crack the passwords are:

Sl No	Username	Cracked Password
1	Otis	password1
2	Zoey	heart
3	Marissa	pull
4	Wesley	nose
5	Conner	last
6	Jazmin	drop
7	Edwin	vary

---

2. Cryptographic Primitives: Synthesis (35 pts)

Adrijan and Bruno wish to communicate several messages (it could be written text, or a two-channel stereo audio signal, or something else), over an insecure channel carried on wired LAN and fibre optic IP networks. Ivo is listening. This will require using a combination of "primitive" elements together. Describe how and why Adrijan and Bruno would use each of the following together to communicate securely:

• Symmetric encryption

• Public key encryption

• Resonant low pass filter (e.g. 4-pole ladder filter)

• Multifactor authentication

• SHA-512

• SKey

• SpreadSpectrum

• Diffie-Hellman key exchange

Are there any external resources or prerequisites that Adrijan and Bruno need to communicate securely? How does your answer change if Ivo is able to transmit on the same channel rather than only listen? Note that you do not need to include all of the above elements, and if your answer is functional and complete, omitting one or more of the above elements is expected. There is no answer that uses all the above. If in doubt, ask!

Solution:

Since Adrijan and Bruno are communicating over an insecure channel and fibre optic IP network, they need to use a combination of the below primitive elements together for secure communication:

a. Symmetric Encryption: Here Adrijan and Bruno can use symmetric encryption like AES-256 for encrypting and decrypting data using the shared secret key to access the information over an insecure channel. The main issue with using this method is the sharing the secret key over an insecure channel which can be achieved by using Public Key Encryption method.

b. Public Key Encryption: An asymmetric public key encryption like RSA can be used for secure key exchange using 2 keys i.e. public and private keys to encrypt and decrypt data. Here, Adrijan can encrypt the messages with Bruno's public key ensuring that only Bruno with his private key can decrypt the data providing an extra layer of security and vice versa. Hence Adrijan and Bruno can use asymmetric public key encryption to securely exchange symmetric encryption keys over an insecure channel.

c. Multi-factor Authentication: Adrijan and Bruno can add an additional layer of security with the shared key (something they know) by implementing a one-time code or physical token (something they have) to strengthen their communication and key exchange [7]. In this case, both Adrian and Bruno need both the factors to authenticate their identity to each other and even if Ivo gets access to one of the factors (password or token), he will still be unsuccessful in tampering the data without having the other factor for authentication.

Therefore, Adrijan and Bruno should first implement Symmetric Encryption to encrypt their data, use public key encryption to share their symmetric keys. They can detect possible data tampering by implementing multi-factor authentication to provide an additional layer of security to ensure secure communication of information over an insecure channel.

External resources or prerequisites that Adrijan and Bruno need to communicate securely are:
1. They should have strong understanding cryptographic techniques to analyse any possible vulnerabilities in their communication methods
2. They should be able to implement encryption methods and execute authentication steps effectively
3. They should have the software and tools to implement cryptographic functions and facilitate MFA etc.

If Ivo can transmit data on the same channel rather than only listen, the situation gets more complicated because of susceptibility to Man-in-the-Middle attacks. Ivo can interfere and modify messages to mimic either Adrijan or Bruno. To prevent this, Adrijan and Bruno should use **Diffie-Helman** algorithm instead of public key encryption. Here, Adrijan and Bruno agree on a publicly available prime number or any other factors and encrypt it with their own private keys to generate a shared secret key [8]. The shared secret key generated with this method is used to exchange the symmetric encryption secret key. This combination of encryption helps in mitigating man-in-the-middle attacks caused due to Public Key Encryption over an insecure channel. Additionally, they can also use **Message Authentication Codes (MAC)** or **Hash Based Message Authentication Codes (HMAC)** [9] where data is encrypted and hashed to generate a fixed-length string that gets tampered if data is modified. **SHA-3** Hash codes and **Digital Signatures and Certificates** [10] can also be used to verify if the information sent over a network is either from Adrijan or Bruno as intended and not altered by intruder Ivo.

References:

[1] Abdmuizz Yekeen , "*The Meaning of the Symbols !, !! and * in /etc/shadow*", 5th Sept 2023, Available: The Meaning of the Symbols !, !! and * in /etc/shadow

[2] Ed Moyle, "*John the Ripper password cracker*", 13th Apr 2023, Available: How to use John the Ripper cracker

[3] Article on "*Comprehensive Guide to John the Ripper*", Available: Comprehensive Guide to John the Ripper

[4] Daniel Miessler, "*Top 10000 passwords*", Available: Top 10000 Passwords: Daniel Miessler: Github

[5] Deniz Halil, "*John the Ripper Cheat Sheet*", 2nd November 2023, Available: John the Ripper Cheat Sheet

[6] Article on "*John the Ripper usage examples*", Available: John the Ripper - usage examples (openwall.com)

[7] Article on "*Multifactor Authentication*", Available: What is: Multifactor Authentication - Microsoft Support

[8] Article on "*Diffie-Hellman Key Exchange*", Available: Diffie–Hellman key exchange - Wikipedia

[9] Kyle Schurman, "*What are MACs and HMACs*", 8th October 2023, Available: What are MACs and HMACs?

[10] Article on "*Digital Signatures and Certificates*", Available: Digital signatures and certificates - Microsoft