

158.335 IoT and Cloud Computing

Security and Privacy in the Internet of Things

Kuda Dube

22/03/2020

Learning Objectives

- At the end of this topic, you should be able to:
 - 1. Discuss the loopholes in the IoT ecosystem;*
 - 2. Describe the challenges in providing security and privacy in IoT;*
 - 3. Explain and make use of generic solutions to IoT security and privacy issues; and*
 - 4. Apply security solutions to the Raspberry Pi used as a gateway in IoT.*

Section I

Introduction

The challenges of security and privacy in IoT

Introduction

- Imagine billions of devices connected to the internet and running without appropriate security measures — an open invitation to hackers who can take control of less-secure devices, and then we will be at their mercy.
- These devices may include highly critical medical equipment, lockers with valuable items, connected vehicles, and city infrastructure, which at any cost cannot be left out in the open for hackers to get access to.
- Therefore, security and privacy in IoT becomes the central and most important pillar of an IoT ecosystem.

Introduction: Security and Privacy Breach Examples (1/2)

Example I:

Medical equipment — e.g., drug diffusion pumps are at risk, which could be hacked to alter the diffusion rate of drugs inside the body, which is a life-threatening situation for a patient.

Introduction: Security and Privacy Breach Examples (2/2)

Example 2:

- Internet-connected vehicles — a hacker can take control of the vehicle and drive to any location or may cause an accident.
- Famous case — two experts demonstrated the security loopholes in a jeep and took complete control over the car remotely. See video at <https://www.youtube.com/watch?v=MK0SrxBCIxs>.

Section 2

IoT Security Challenges

A summary of security and privacy challenges in IoT

IoT Security Challenges:

Security in endpoint devices

(1/2)

- Endpoint devices (constrained) — sensors, actuators, and controllers;
- They don't have enough memory or processing power, and run on low and limited power;
- Traditional security approaches don't work because they use heavy encryption and decryption algorithms requiring high processing power, a lot of memory, and power to facilitate the computations in real time;

IoT Security Challenges:

Security in endpoint devices

(2/2)

- Possible solution 1 — make use of embedded encryption and decryption techniques that are part of sensors and controllers themselves.
- Possible solution 2 — put the devices on separate networks and use firewalls to overcome their limitations.

IoT Security Challenges: Authorization and authentication (1/2)

- Only authentic devices should have authorization to participate in the network for IoT;
- Problem — this basic requirement is largely unmet or there is weak password policies that make these devices prone to attacks;

IoT Security Challenges:

Authorization and authentication (2/2)

- Possible solutions — make sure that a device that participates in the network:
 - *is authentic and has correct authorization;*
 - *make use of SSL certificates at both the device and application level;*
 - *dual authentication, such as password and SMS (or in-app passcodes), biometric signatures, and multi-factor authentication (MFA).*

IoT Security Challenges: Device firmware upgrade (1/3)

- Firmware Over The Air (FOTA) — updating the firmware and software of IoT devices and gateways for adding or upgrading security features
- FOTA challenges:
 - *the number of devices can be huge;*
 - *difficult to keep track of current software/firmware versions of all the devices;*
 - *hard to determine what updates are available for all of devices;*
 - *different types of devices present in the same network with completely different software and firmware increase the complexity of the upgrade process;*

IoT Security Challenges: Device firmware upgrade (2/3)

- Further FOTA challenges:
 - *devices may be in completely different networks supporting different protocols — an added challenge to the ones above;*
 - *the devices may not have a FOTA facility implying manual upgrade or replacement;*
 - *for consumer devices — the right to update the device software is left to owner and if owner opts out of upgrading, device remain prone to attacks;*
 - *when IoT device is manufactured by third-party vendor and they stop producing it, then its a challenge to upgrade*

IoT Security Challenges: Device firmware upgrade (3/3)

- Main solution — have a **device manager application** that keeps a record of all the devices, along with their software and firmware versions.
- **For devices that have a FOTA facility** — the device manager application automatically upgrades the device whenever there is a newer version available and in case the upgrade fails, it rolls back to a previous stable version of the software.
- **For the devices that don't have FOTA** — keep the software simple and minimal, to perform the basic actions required, and the rest of the complex computation can be done using gateway devices to which it connects, and a gateway device enables connectivity to the internet.
- **Gateway devices** (e.g., the Raspberry Pi) are more sophisticated than sensors and actuators and have FOTA as well.

Section 3

Secure Communication

- Cloud IoT — communication takes place devices, the Cloud and applications;
- Problem: unencrypted data communication, so network security is important in IoT
- Possible solution 1 — make use of *Transport Level Security* (TLS) and the *Secured Sockets Layer* (SSL) certificates at the device and application or Cloud level to encrypt the data before sending it.
- Possible solution 2 — Cloud IoT service providers (e.g. Amazon) give additional security features as part of their services, which further enhance the security of the IoT system (you already saw this in previous lab practicals on AWS IoT).

Data Security (I/2)

- Large number of IoT devices — generation of huge volumes of data;
- Medical information, financial information, or organizational and national information may be personal or sensitive and can be of interest to data thieves;
- Data should not be stored with the identity of its owner wherever possible;
- Owner of the data should have the option to share their data with the service provider for further services.

Data Security (2/2)

- Possible solutions to data security and privacy:
 - *all user and sensor data generated by the IoT system should be stored in a highly secured data storage system;*
 - *data storage should be:*
 - guarded by firewalls;
 - inside private networks;
 - not be available to the outside world over the internet;
 - free from identity of its owner wherever possible.

High Availability

- **Challenge:** *keeping the IoT system up and running with almost zero downtime;*
- **Critical applications** requiring high availability: *medical equipment, traffic control signals, and power grids;*
- **Focus of high availability:** *end devices, gateway devices, mobile and desktop applications and cloud services;*
- **Possible solution 1:** *multi-cluster architecture — one primary and multiple secondary systems running in sync with failure of primary system triggering takeover by secondary system with no downtime;*
- **Possible solution 2:** Use a Cloud IoT platform such as AWS IoT that provide high availability, secure and scalable services for IoT.

Identifying cyber attacks

- **Cyber attacks** — often disguised as one of the legitimate nodes in a network;
- **Difficult to identify when the attack takes place** — due to the large number of devices, multiple networks, different protocols, and varied working nature of multiple sensors;
- **Possible solutions** — monitor all the networks to:
 - *detect anomalies,*
 - *check the logs of the communication that takes place between devices and applications,*
 - *perform penetration testing,*
 - *identify which devices are compromised, and*
 - *see what data has been stolen*

Absence of standards

- There is **general lack of standards** for the realisation of IoT;
- An IoT system is made up of **multiple different components combined together** — next to impossible to have a set standard of security in IoT because one set of security features may work in one case and fail in another;
- **Promising scenario** — organizations across the world are getting together to form **IoT alliances** for setting up **IoT standards**;
- **What to do** in the meantime — follow **established best practices** for implementing each component of the IoT system until IoT standards become available.

Ignorance from customers and manufactures

- **Vendor focus** — highlighting a long list of features with security coming last or not at all;
- **Research evidence** — security is considered just another feature of the product for which consumers are not willing to pay and so de-prioritised by manufacturing;
- **Solution** — whenever a new product is designed, security features should have top-most priority and other features should be designed around it, instead of having security as an optional add-on feature (cf. AWS IoT Core);

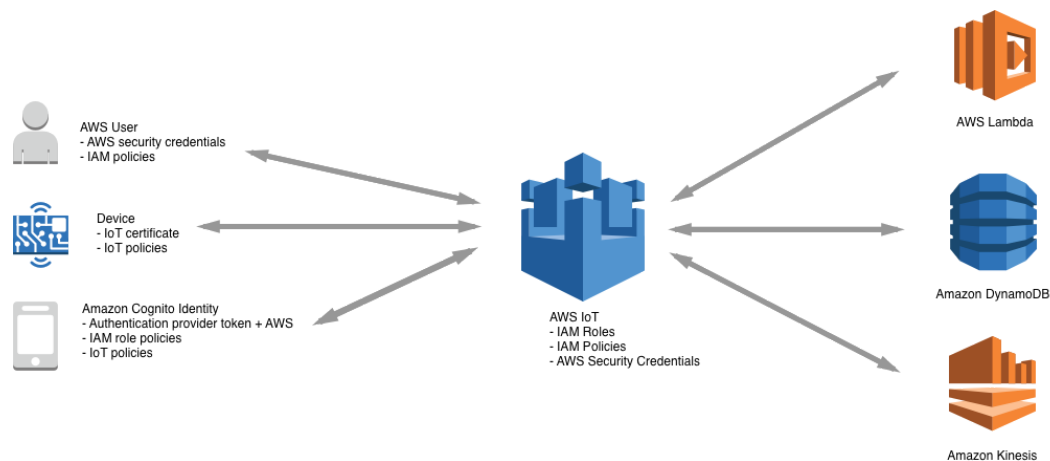
Section 4

Security in AWS IoT

AWS IoT Security (1/6)

- Each connected device or client must have a credential to interact with AWS IoT.
- All traffic to and from AWS IoT is sent securely over Transport Layer Security (TLS).
- AWS cloud security mechanisms protect data as it moves between AWS IoT and other AWS services.

AWS IoT Security 2/6



AWS IoT Security (3/6)

Your responsibilities

- You are responsible for managing
 - **device credentials** - X.509 certificates, AWS credentials, Amazon Cognito identities, federated identities, or custom authentication tokens; and
 - **policies** in AWS IoT.
- For more information, see [Key Management in AWS IoT](#).
- You are also responsible for:
 1. assigning **unique identities** to each device and
 2. managing the **permissions** for each device or group of devices.

AWS IoT Security (4/6)

- Your devices connect to AWS IoT using **X.509 certificates** or **Amazon Cognito identities** over a secure **TLS connection**.
- During research and development, and for some applications that make **API calls** or use **WebSockets**, you can also **authenticate using IAM users and groups** or custom authentication tokens.
- For more information, see [IAM Users, Groups, and Roles](#).

AWS IoT Security (5/6)

- When using **AWS IoT authentication**, the **message broker** is responsible for:
 - *authenticating your devices,*
 - *securely ingesting device data, and*
 - *granting or denying access permissions you specify for your devices using AWS IoT policies.*

- When using **custom authentication**, a **custom authorizer** is responsible for:
 - *authenticating your devices and*
 - *granting or denying access permissions you specify for your devices using AWS IoT or IAM policies.*

AWS IoT Security (6/6)

- The AWS IoT rules engine forwards device data to other devices or other AWS services according to rules you define.
- It uses AWS Identity and Access Management to securely transfer data to its final destination.
- For more information, see [Identity and Access Management for AWS IoT](#).

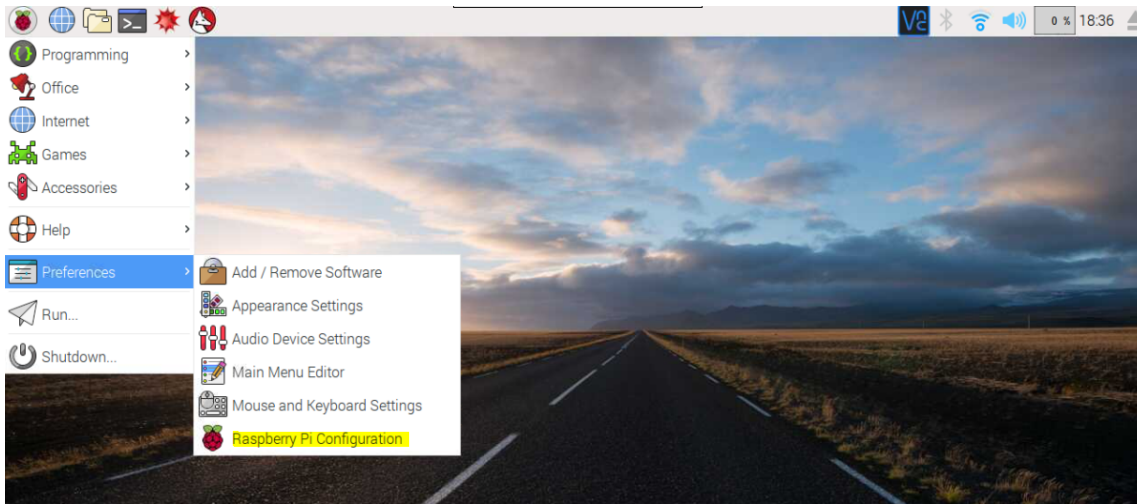
Section 5

Securing the Raspberry Pi

Now lets focus on securing the Raspberry Pi, which has been adopted as the IoT gateway of choice in this course

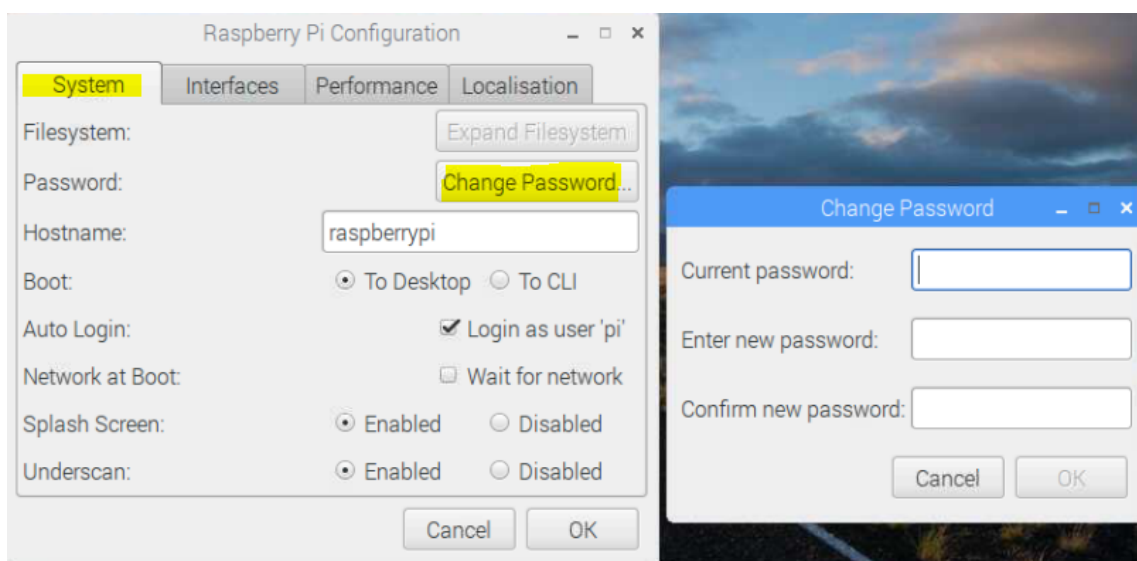
Securing the RPi: Changing the default password via the GUI (1/2)

Step 1: Open the menu and go to “Preferences” and then “Raspberry Pi Configuration”



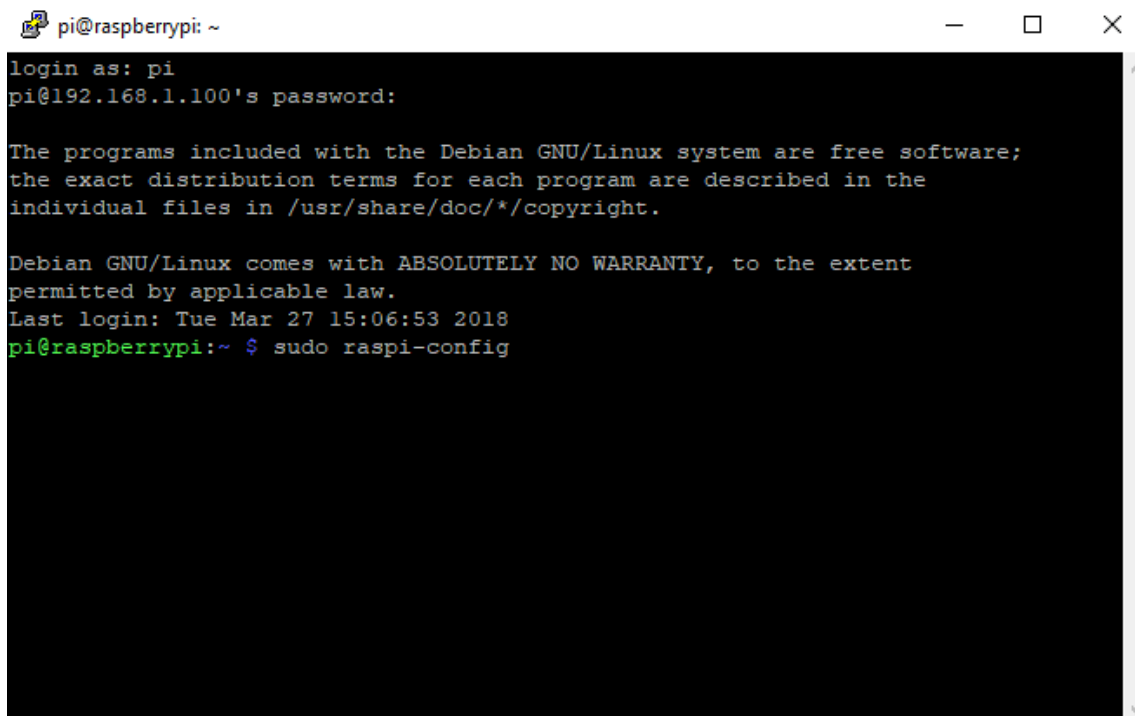
Securing the RPi: Changing the default password via the GUI (2/2)

Step 2: Within “Raspberry Pi Configuration” under the “System” tab, click the “Change Password” button. This will prompt you to provide a new password. After that, click on “OK” and the password is changed



Securing the RPi: Changing the default password via the Command Line (1/3)

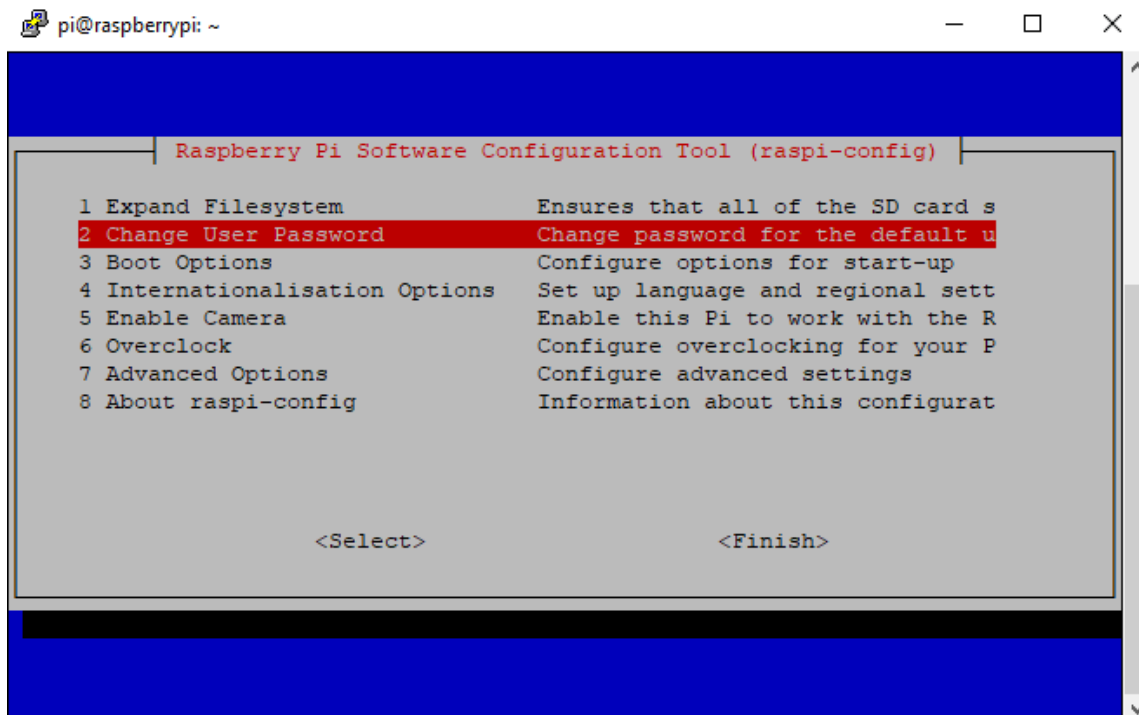
Step 1: If you are logging into the RPi through PuTTY using SSH, or you are simply using the RPi commandline, then open the configuration setting by running the **sudo raspi-config** command. (**sudo** means “superuser do” or “do as superuser”, superuser is the admin user)



```
pi@raspberrypi: ~  
login as: pi  
pi@192.168.1.100's password:  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Mar 27 15:06:53 2018  
pi@raspberrypi:~ $ sudo raspi-config
```

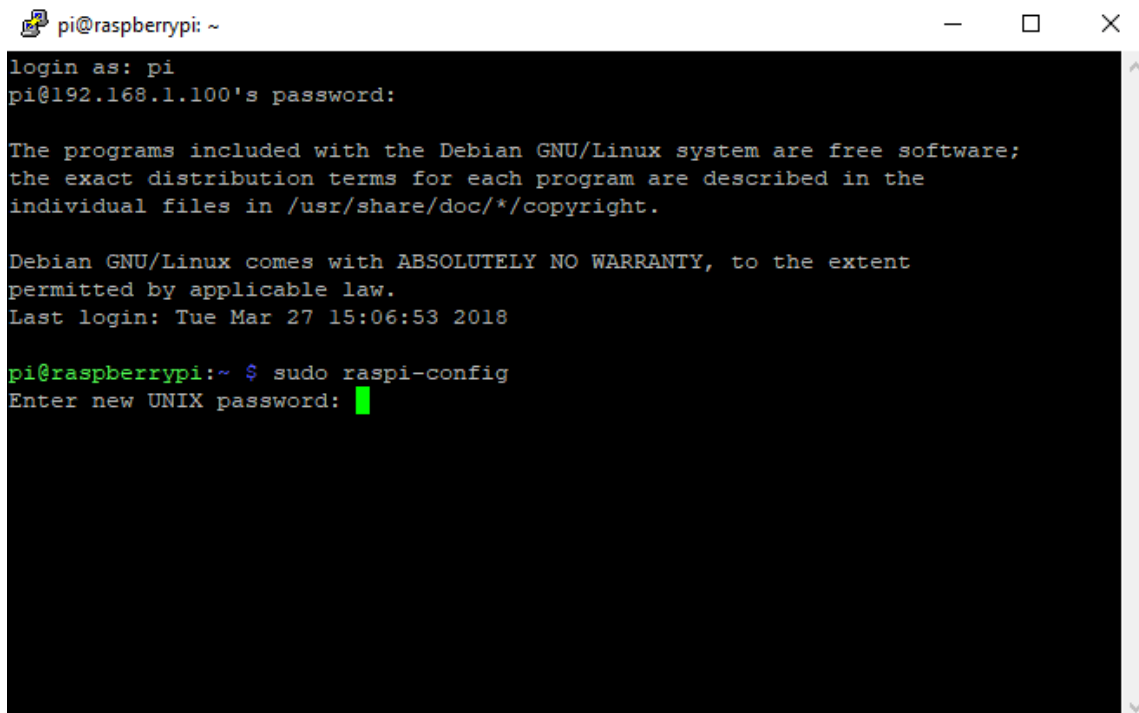

Securing the RPi: Changing the default password via the Command Line (2/3)

Step 2: On successful execution of the command, the configuration window opens up. Then, select the second option to “change the password” and then “finish”.



Securing the RPi: Changing the default password via the Command Line (3/3)

Step 3: It will prompt you to provide a new password; you just need to provide it and exit. Then, the new password is set.



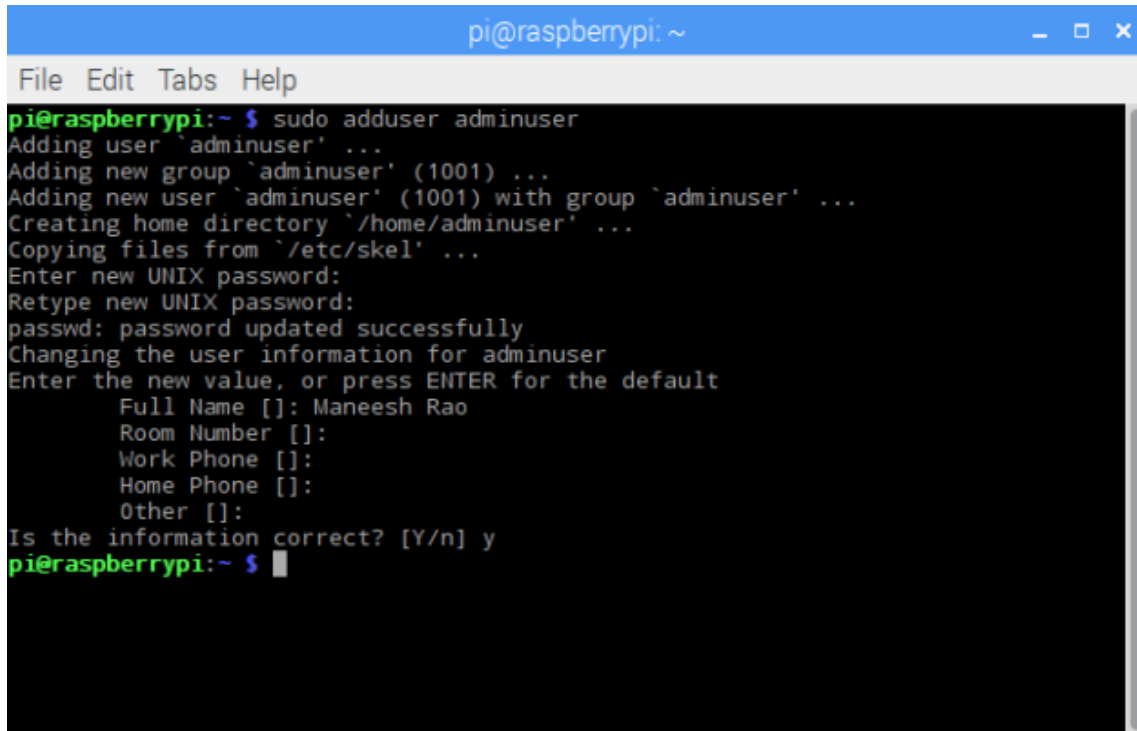
```
pi@raspberrypi: ~  
login as: pi  
pi@192.168.1.100's password:  
  
The programs included with the Debian GNU/Linux system are free software;  
the exact distribution terms for each program are described in the  
individual files in /usr/share/doc/*/copyright.  
  
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent  
permitted by applicable law.  
Last login: Tue Mar 27 15:06:53 2018  
  
pi@raspberrypi:~ $ sudo raspi-config  
Enter new UNIX password: 
```

Securing the RPi: Changing the default admin username (I/4)

- All RPi's come with the default username "pi", which should be changed to make the RPi more secure.
- Reason: Denying an attacker knowledge of the username makes its a little harder to hack the RPi;
- We create a new user and assign it all rights, and then delete the "pi" user.

Securing the RPi: Changing the default admin username (2/4)

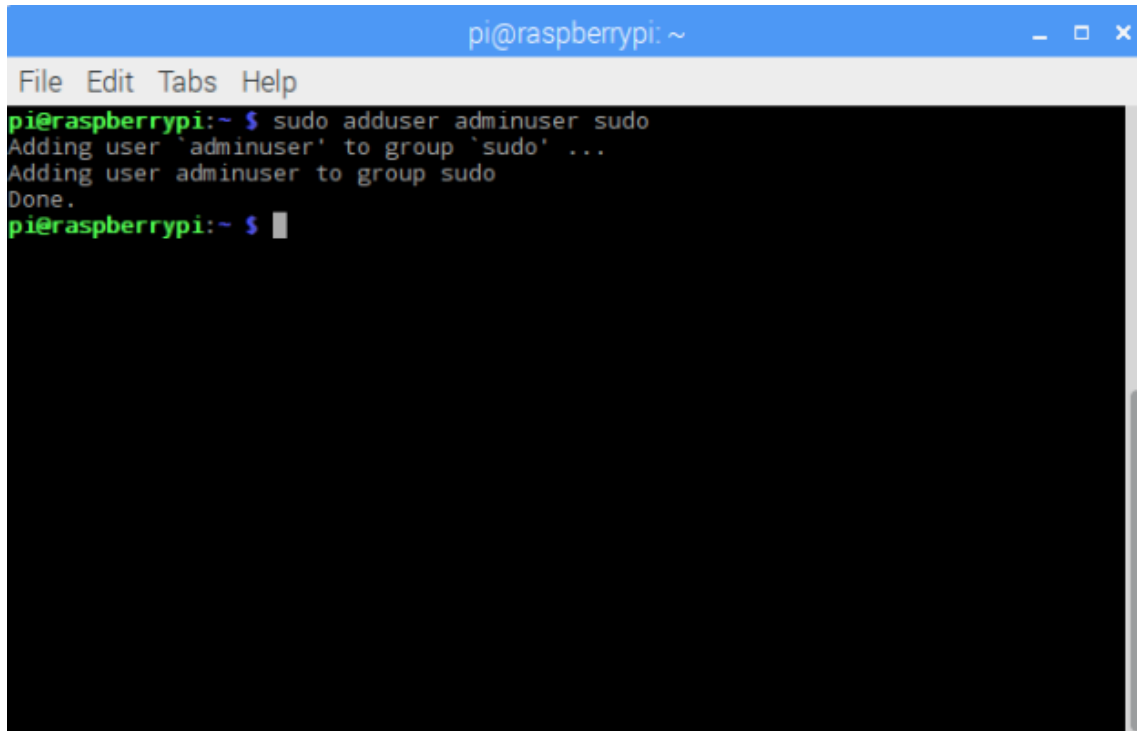
Step 1: To add a new user, run the **sudo adduser adminuser** command in the terminal. It will prompt for a password; provide it, and you are done



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo adduser adminuser  
Adding user `adminuser' ...  
Adding new group `adminuser' (1001) ...  
Adding new user `adminuser' (1001) with group `adminuser' ...  
Creating home directory `/home/adminuser' ...  
Copying files from `/etc/skel' ...  
Enter new UNIX password:  
Retype new UNIX password:  
passwd: password updated successfully  
Changing the user information for adminuser  
Enter the new value, or press ENTER for the default  
    Full Name []: Maneesh Rao  
    Room Number []:  
    Work Phone []:  
    Home Phone []:  
    Other []:  
Is the information correct? [Y/n] y  
pi@raspberrypi:~ $
```

Securing the RPi: Changing the default admin username (3/4)

Step 3: Now, you will add your newly created user to the **sudo** group so that it has all the root-level permissions (admin permissions).

A terminal window titled 'pi@raspberrypi: ~' with a blue header bar. The window contains a menu bar with 'File', 'Edit', 'Tabs', and 'Help'. The terminal text shows the command 'sudo adduser adminuser sudo' being executed. The output indicates that the user 'adminuser' is being added to the 'sudo' group. The prompt returns to 'pi@raspberrypi:~ \$' with a cursor.

```
pi@raspberrypi:~ $ sudo adduser adminuser sudo
Adding user 'adminuser' to group 'sudo' ...
Adding user adminuser to group sudo
Done.
pi@raspberrypi:~ $
```

Securing the RPi: Changing the default admin username (4/4)

Step 4:

- Now, we can delete the default user, “pi”, by running the `sudo deluser pi` command.
- This will delete the user;
- Note that its repository or home folder **/home/pi** will still be there.
- If required, you can delete this home folder as well.
- To login as the new user, type the **reboot** command in the terminal to restart the RPi.

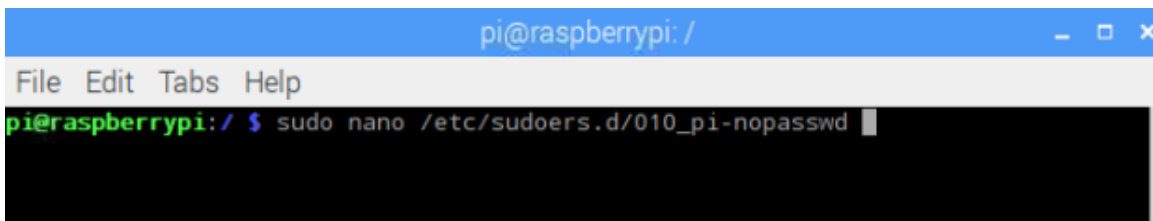
Securing the RPi: Making “*sudo*” require a password (1/2)

- When a command is run with **sudo** as the prefix, then it will execute it with superuser or admin privileges.
- By default, running a command with **sudo** doesn't need a password, but this can cost dearly if a hacker gets access to Raspberry Pi and takes control of everything.

Securing the RPi: Making “sudo” require a password (2/2)

Step I:

- To make sure that a password is required every time a command is run with superuser or admin privileges, edit the **010_pi-nopasswd** file under **/etc/sudoers.d/** by executing the command shown in the screenshot below.
- This command will open up the file in the nano editor; replace the content with the one line of text:
username ALL=(ALL) PASSWD: ALL
(where “username” is your new username or “pi” if you have not yet changed/deleted it), and save it.



```
pi@raspberrypi: /  
File Edit Tabs Help  
pi@raspberrypi:/ $ sudo nano /etc/sudoers.d/010_pi-nopasswd
```


Securing the RPi: Updating the Raspbian OS

- To get the latest security updates, it is important to ensure that the Raspbian OS is updated with the latest version whenever available.
- Visit <https://www.raspberrypi.org/documentation/raspbian/updating.md> to learn the steps to update Raspbian OS.

Securing the RPi: Improving SSH security (I)

- SSH is one of the most common techniques to access Raspberry Pi over the network;
- SSH becomes necessary to use if you want to make it secure.

Securing the RPi: Improving SSH security (2)

Username and password security

- Apart from having a strong password, we can allow and deny access to specific users in SSH connections.
- This can be done by making changes in the **sshd_config** file.
- Run the **sudo nano /etc/ssh/sshd_config** command.
- This will open up the **sshd_config** file; then, add the following line(s) at the end to allow or deny specific users:
 - To allow users, add the line: **AllowUsers tom john merry**
 - To deny users, add this line: **DenyUsers peter methew**
- It is necessary to reboot the Raspberry Pi for these changes to take effect.

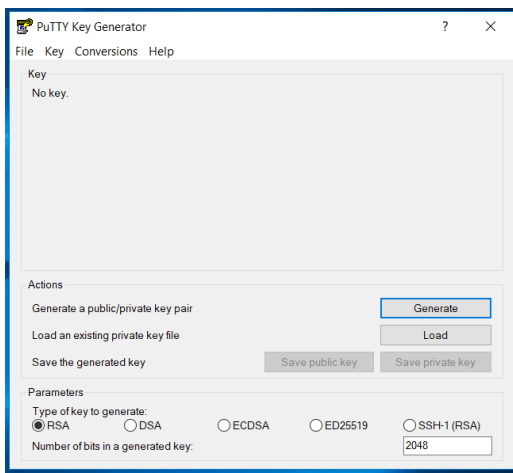
Securing the RPi: Improving SSH security (3)

Key-based authentication

- Using a **public-private key pair** for authenticating a client to an SSH server (Raspberry Pi), we can secure our Raspberry Pi from hackers.
- To enable **key-based authentication**, we first need to generate a public-private key pair using tools called **PuTTYgen for Windows** and **ssh-keygen for Linux**.
- Note that *a key pair should be generated by the client* and not by Raspberry Pi.

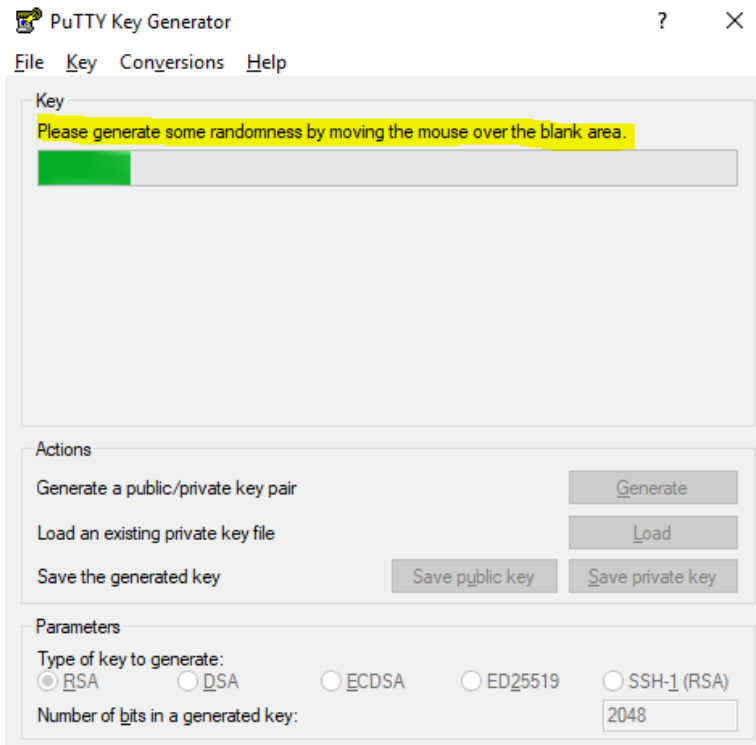
Securing the RPi: Improving SSH security (4)

- For our purpose, you will use PuTTYgen for generating the key pair.
- Download PuTTY from the following web link: <https://www.putty.org/>
- Note that puTTYgen comes with PuTTY, so you need not install it separately.
- Open the puTTYgen client and click on Generate, as shown in the screenshot below:



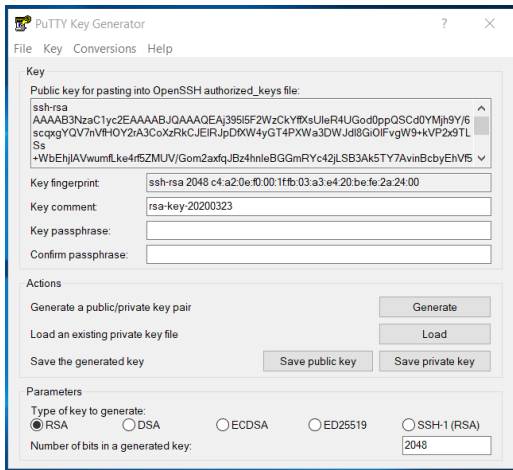
Securing the RPi: Improving SSH security (5)

Next, you need to hover the mouse over the blank area to generate the key, as highlighted in the screenshot:



Securing the RPi: Improving SSH security (6)

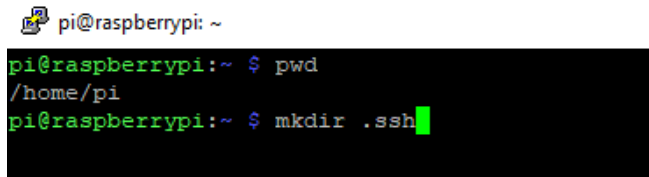
- Once the key generation process is complete, there will be an option to save the public and private keys separately for later use, as shown in the screenshot;
- Ensure you keep your private key safe and secure.



- Name the public key file **rpi_pubkey**, and the private key file **rpi_privkey.ppk** and transfer the public key file **rpi_pubkey** from our system to Raspberry Pi.

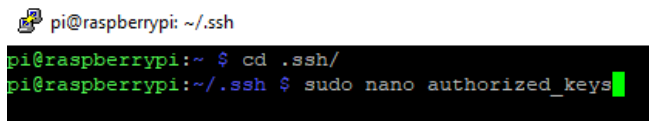
Securing the RPi: Improving SSH security (7)

Log in to Raspberry Pi and under the user repository, which is **/home/pi** or **/home/username** (replace “username” with yours), create a special directory with the name **.ssh**, as shown in the screenshot:



```
pi@raspberrypi: ~  
pi@raspberrypi:~ $ pwd  
/home/pi  
pi@raspberrypi:~ $ mkdir .ssh
```

Now, move into the **.ssh** directory using the **cd** command and create or open the file with the name **authorized_keys**, as shown in the screenshot:



```
pi@raspberrypi: ~/.ssh  
pi@raspberrypi:~ $ cd .ssh/  
pi@raspberrypi:~/.ssh $ sudo nano authorized_keys
```

The **nano** command opens up the **authorized_keys** file in which we will copy the content of our public key file, **rpi_pubkey**. Then, save (Ctrl + O) and close the file (Ctrl + X).