|  |
| --- |
| IU |
| Software Architectures of Mobile Devices |
| DLBCSEEMT02\_E  Panji Harawa  If you do not wish to be credited, please indicate it below:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

# Learning Objectives

The course **Software Architectures of Mobile Devices** offers a deep dive into the intricacies of mobile device technology, focusing on the various aspects that make up the handset technology stack. As mobile devices become an essential part of everyday life, understanding the underlying software architectures, and ensuring their efficient operation and security, has grown increasingly important. In this course, you will examine the fundamental components of the handset technology stack, including hardware, firmware, operating systems, applications, and ecosystem. You will develop an understanding of the software architectures that underpin Android and iPhone Operating System (iOS) operating systems, focusing on their unique features, security, and performance capabilities. Additionally, you will delve into the variety of mobile device types and their associated operating systems, such as the internet of things (IoT), Linux, real-time operating systems (RTOSs), and embedded systems.

Throughout the course, you will study the common threats and vulnerabilities that impact mobile devices and learn the principles of mobile device management for their prevention and mitigation. Furthermore, you will assess the role and impact of software ecosystems and cloud services on mobile device security and usability. By the end of the course, you will be able to investigate the main components and functions of the handset technology stack, including hardware, firmware, operating systems, applications, and ecosystem. You will also be able to compare the software architectures of Android and iOS operating systems, evaluating their features, security, and performance capabilities.

In the second part of the course, you will build on your understanding of mobile device software architectures and delve deeper into advanced topics related to mobile device management, security, and usability. You will explore the ever-evolving landscape of mobile device technology and the challenges it presents. By the end of the course, you will be able to differentiate between distinct types of mobile devices and their operating systems, such as IoT, Linux, RTOSs, and embedded systems. You will also be able to identify and analyze common threats and vulnerabilities in mobile devices and apply mobile device management techniques for prevention and mitigation. Lastly, you will evaluate the role and impact of software ecosystems and cloud services on mobile device security and usability and develop strategies for optimizing device performance and user experience.

## Basic Reading

Tanenbaum, A. S., & van Steen, M. (2015). *Modern operating systems* (4th ed.). Pearson Education.

Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating system concepts* (10th ed.). John Wiley & Sons.

## Further Reading

# Unit 1 – Handset Technology Stack?

Study Goals

On completion of this unit, you will be able to …

… Describe the main components of hardware in mobile computing.

... Understand how mobile operating systems deal with mobile threats.

… Describe how a mobile device ecosystem is structured.

… Discuss in detail how firmware works in a mobile device.

## Introduction

In recent decades, the technology space has witnessed unparalleled change in modular design, high-level architecture engineering, and use of mobile devices. In today’s world, the mobile technology stack comes with a rich ecosystem of mobile applications, which would not be possible without the advancements in software and hardware architecture of the past two decades. Furthermore, the modern mobile handset technology stack provides multipurpose computing platforms that can execute tasks beyond conventional voice and text, as most legacy devices were limited to. To this end, mobile devices nowadays are extended with a number of sensors that provide information to them, as well as through access to the cloud. Mobile devices are equipped with several communication technologies, including cellular network technologies (4G/5G), wireless fidelity (WiFi), Bluetooth, near field communication (NFC), and the global positioning system (GPS), as well as several sensor technologies: face recognition, fingerprint recognition, cameras, and acceleration sensors. The computing power of a mobile device nowadays is a million times higher than that of the personal computer (PC) that was used for the Apollo 11 mission – power that can be further extended with computing resources present in the cloud. In this section we will discuss the hardware architecture of mobile devices, the integration of their firmware and operating system (OS), and the application ecosystem on top. All of this makes it possible for a mobile device to fulfill its mission of being a personal computer in your pocket.

## 1.1 Hardware

Smartphones in the realm of mobile computing are often regarded as minicomputers due to their extensive array of features, which rival those of conventional computers. Like computers, mobile devices’ hardware is composed of the following main parts [Bhunia 2018]:

* device processor
* device memory
* peripherals

The end device is capable of capturing information from the physical world and interacting with it using its input/output peripherals. In a smartphone, a touch screen can act either as an input peripheral (it receives the physical touch signal from the user) or as an output peripheral (it displays various images on the screen). This incoming information from the physical world is transformed into digital signals so that they can be processed and stored. A similar approach occurs in the communication front end. The physical signals of the transmitted waves are transformed into digital signals by sampling the transmitted analog radio wave into I and Q samples, which will be processed further by digital signal processing (DSP) modules.

**System on a chip**

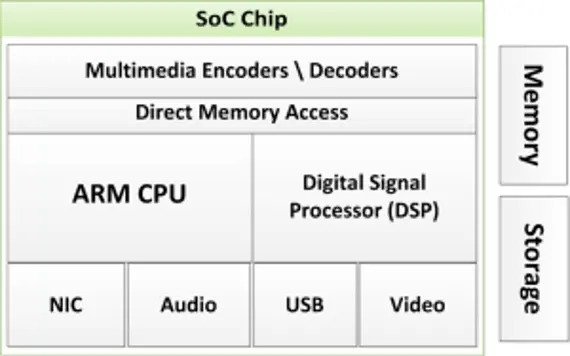
refers to the hardware stack on a mobile device.

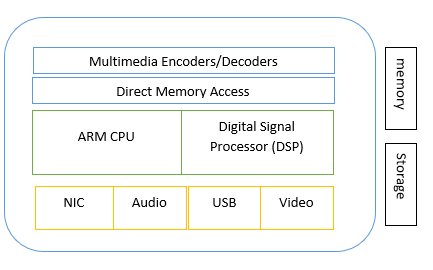
The hardware stack in a mobile device is consolidated within a system on a chip (SoC), which serves as an integrated circuit binding together all the functional components of a computer. Unlike the typical single processing chip used in a personal computer assembly, the SoC encompasses a package that integrates various processing parts, including modems, memory, and other components, into a single chip embedded within the phone’s circuitry.

The process of integrating these components onto a single chip is essential as it makes it possible for phones to save on cost, power, and space. The SoC can also be referred to as the “brain” of any smartphone; it is used to manage and handle all processes from the phone’s operating system and to receive and interpret all physical inputs on a phone, such as power off/on buttons. The SoC is also the component that connects other parts of the phone, such as cameras, memory, display, flash memory, and many more.

#### 1.1.1 SoC components

Most Common SoC Components





Source: Panji Harawa [2023] based on Katakkar, T. [2022, August 9]. What is an SoC? Engineers’ Garage. <https://www.engineersgarage.com/what-is-an-soc> .

According to the Harvard computer architecture [Pawson 2022], the computer architecture is composed of the control unit, data memory, instruction memory, input/output (I/O) devices, and arithmetic-logic unit (ALU). Unlike the computer architecture, where there is a clear physical separation between components, in mobile devices all the components are integrated in the SoC for better performance and lower power consumption. In the figure *Most Common SoC Components*, the most common components are shown and can be explained as follows [Singh 2014]:

* **Central Processing Unit (CPU)**: This is considered the “brain” of the phone, as it runs most of the phone’s OS and mobile apps, e.g., the ARM CPU. The CPU performs the instruction executions from the instruction memory; it performs arithmetical and logic operations (AND, OR, NOT), performs clock management for all the other components in the SoC, and handles interrupts from the peripheral devices.
* **Graphics Processing Unit (GPU)**: This is responsible for handling graphics related to visualization tasks, an app’s user interface, and 2D/3D gaming. It performs graphics and video processing; it drives the output of the display and is able to perform parallel computing. Because of this, it is also used in parallel computing tasks, such as machine learning algorithms and crypto mining.
* **Image/Multimedia Processing Unit (ISP)**: This is used mostly to convert camera data and process it into image and video files.
* **Digital Signal Processor (DSP)**: This is used to manage all mathematical and processor-intensive functions, such as decompressing and compressing music files and analyzing gyroscope sensor data. It performs a variety of processing on signals coming from peripherals. It helps in the processing of audio signals, video signals, and sensor data, as well as signals coming from wireless interfaces, e.g., WiFi, Bluetooth, and NFC.
* **Multimedia Encoder/Decoder**: This converts and manages power during conversion of video files and formats.
* **Memory Processor:** This is used to manage device memory blocks used for computation. The device has two types of memory: volatile, which is used to store information on a temporary basis, and non-volatile, used for long term storage of information and data. Another categorization of memory is random-access memory (RAM), used to store data on a temporary basis, and read-only memory (ROM), which can store data even under a power-off condition of the device.
* **External Storage Interfaces:** SoCs include interfaces for connecting external storage devices. Common interfaces to external storage units found in SoCs include secure digital (SD) card interfaces, universal flash storage (UFS) interfaces, serial ATA (SATA) interfaces, and universal serial bus (USB) interfaces,
* **Modem/Network Interface Card (NIC)** — This handles all wireless signals and converts them into data the phone understands; this module is responsible for converting and communicating with wireless signals generated from 4G LTE, 5G, WiFi, and Bluetooth, infrared, and other modems.

With advancements in machine learning (ML) and artificial intelligence (AI), more AI algorithms are running on mobile devices for different tasks, e.g., for video analytics. To speed up AI algorithms based on neural networks, on the one hand, neural networks are being composed to speed up processing time, while on the other hand, more powerful processing units, such as neural processing units (NPUs), are being included in mobile devices [Tan 2022].

The integration of mobile hardware core functionalities using integrated circuits (ICs), as part of SoC, has made it possible for smartphones to be smaller and have many more functionalities than before, when they were primitive and bulky. The use of ICs has also made it possible for mobile devices to execute millions of instructions per second. The sizes of SoCs in mobile devices is an important factor in the competitiveness between hardware manufacturers. SoCs, which are measured in nanometers (nm), are designed to be small and efficient, with the goal of making mobile devices faster and more efficient. As technology continues to evolve, these chips are becoming increasingly smaller, which is a trend that is likely to continue [Delgado et al. 2020].

#### 1.1.1 ARM CPU

As mentioned previously, one of the most widely used CPU architectures for mobile devices is the ARM CPU. For mobile devices, the ARM 32-bit architecture is used. It is calculated that around 60% of today’s mobile phone run on the ARM CPU architecture. The reason why the ARM chip has become so popular in the market are

* It provides a highly customized SoC that achieves high passage efficiency. The customization is based on watts used per operation, making it suitable for use in battery powered devices like smartphones and other IoT devices.
* It is highly scalable, allowing vendors to adapt the usage of the SoC for different use cases and applications.
* It has strong industry support.

The ARM CPU architecture has three main profiles (as of AMRv7) [Singh 2014]:

* Application profile or A-profile, supported by the Cortex-A series.
* Real time profile or R-profile, supported by the Cortex-R series.
* Microcontroller profile or M-Profile, supported by the Cortex-M series.

### Self-Check Questions

* List some of the main hardware components of a mobile device, and how do they work together to enable the device to function?
* *- Processor (CPU): The central processing unit (CPU) is the “brain” of the device, responsible for executing instructions and performing tasks.*
* *- Display: The display is used to show information to the user and is usually a touchscreen that allows the user to interact with the device. Asks.*
* What is the main function of the CPU in a mobile device?
* *The CPU, or central processing unit, is the main processing unit in a mobile device and is responsible for executing instructions and performing tasks.*
* What is the main difference between RAM and ROM in a mobile device?
* *RAM, or random-access memory, is used to store data temporarily and is volatile, meaning the data is lost when the device is powered off. ROM, or read-only memory, is used to store data that is not intended to be changed and is non-volatile, meaning the data is not lost when the device is powered off.*
* What are some examples of input/output (I/O) devices in a mobile device?
* *Some examples of I/O devices in a mobile device include buttons, keyboard, touchpad, and speakers. These devices allow the user to input data and receive output from the device.*

## 1.2 Firmware

Mobile devices have several core functional components that work together at a high architecture level. One of the key functional components is the firmware, which is hardcoded software that is stored in mobile devices’ static RAM, where content is retained on a hardware device even after interruption of external source of power. As there are different hardware devices, for each hardware device there will be different firmware. In the figure *Mobile Device Components and Place of Firmware in Architecture,* we have shown the layers of a mobile device; the firmware is located (logically) in-between the hardware and operating system.

**Firmware**

This is a read only memory type of software that is used for low-level management of any small but critical programs on a mobile device.

In mobile devices the firmware used to access the cellular network and the firmware used for operating the general-purpose operating system of the device are fully separated [Franklin 2020]. The former, together with the hardware, makes up the telephony subsystem, which in combination with the real-time operating system (RTOS) gives the mobile device access to the cellular network. The latter, together with the general-purpose operating system, make up the application process and can be accessed by both the user and the administrator of the device. In addition to hardware and firmware separation, the telephony subsystem utilizes a completely separate SoC called the baseband processor [Franklin 2020]. The telephony subsystem collaborates with the subscriber identity module (SIM) card, which stores the international mobile subscriber identity (IMSI), while the international mobile equipment identifier (IMEI) is stored in the device firmware.

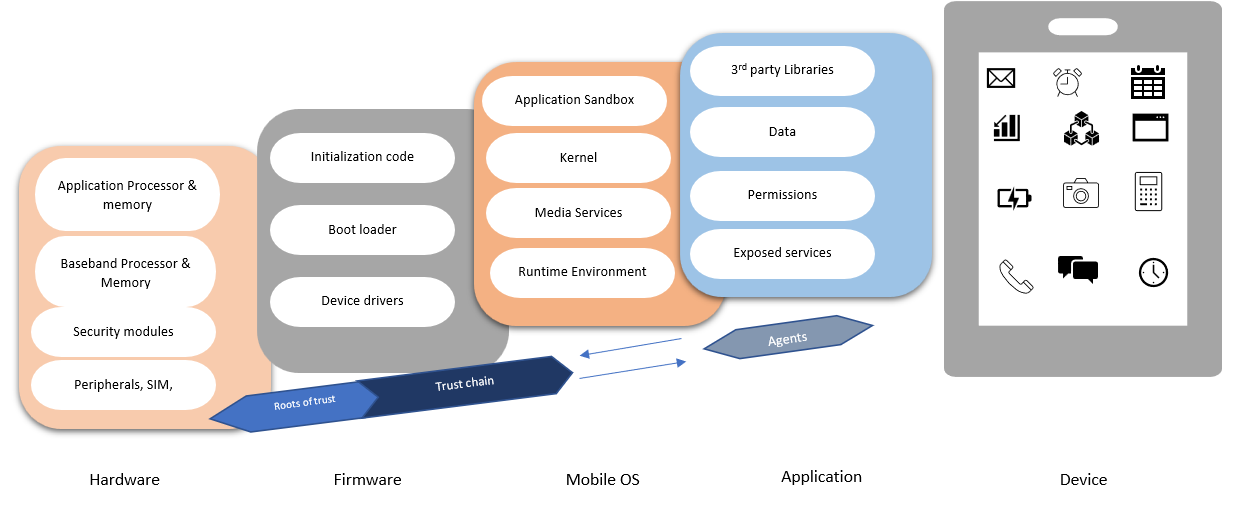
#### 1.2.1 What is firmware?

Firmware ***is a read only memory type of software that is used for low level management of any small but critical programs on a mobile device***. It is used for specific hardware-related components such as the basic input-output system (BIOS), which mostly provides high-level abstraction on a mobile device. By design the firmware is usually part of a hardware stack that is not easily accessible by regular end users of the smartphone.

Mobile Device Components and Place of Firmware in Architecture

Diagram

Description automatically generated



Source: Panji Harawa [2023] based on Source: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-124r2-draft.pdf> .

Firmware is a critical part of a mobile device and is key to the high-level abstraction, which also varies depending on the type of mobile operating system being used in the mobile device.

The key roles of the firmware are [Franklin 2020]

* Device booting: Firmware boots up the device once it is powered on and takes care of hardware initialization. Before the operating system starts, the firmware performs self-tests of each hardware component.
* Loading device drivers: Firmware loads the device drivers that enable communication between the hardware components and the operating system. Using the device drivers, the operating system can access hardware devices (e.g., WiFi interface card) and interact with them (e.g., send/receive packets).
* Hardware control: It provides the necessary instructions to control hardware devices. For example, in the case of a Bluetooth interface, it provides the necessary instructions to communicate with this from higher layers (from the operating system).
* Root of trust: It provides the root of trust for all logical layers on top [Zhao 2014].

Firmware provides different levels of abstraction. Based on the abstraction level, firmware can be divided into high-level firmware and low-level firmware. High-level firmware relates to abstraction levels toward the operating system. In a mobile device (e.g., Samsung or Apple phones), it provides abstraction levels towards the Android and iOS operating systems, respectively. On the other hand, low-level firmware provides direct control over hardware devices, such as the CPU, memory, peripherals, and interface cards.

As part of a mobile device all peripherals and application processors formulate a root of trust. These hardware components interface with the firmware to power or boot the device, also known as a **boot loader**. The boot loader will first initialize and then allow additional code that is responsible for loading device drivers for all the hardware elements of the mobile device and other peripherals.

**Boot loader**

Refers to a component on a mobile device that is responsible for booting the device.

#### 1.2.2 Threats towards firmware and telephony subsystem

A mobile device will not boot and function properly if there has been any tampering with the boot loader part of the firmware. To protect the firmware from such code tampering, part of the firmware for secure boat load can be executed in the TEE. The code saved in the TEE cannot be changed or altered, even by the owner or user of the device. The TEE can be used as an anchor of trust for securely booting the device. An example of a TEE is the ARM TrustZone, which is a set of hardware secure extension features to the ARM SoC in processor, memory, and peripherals that can be used for secure bootup of the device [Zhao et al. 2014]. This then creates a high-level abstraction that is useful for storing and executing sensitive cryptographic and secure computations that prevent mobile threats coming through applications. All these operations are done by the device in a TEE that is provided by the hardware of the processor itself. [Mahendra et al. 2014].

Another threat to the telephony subsystem of a mobile devices is SIM card swapping or SIM card code tampering [Lin 2018]. To mitigate such threats and possible attacks, SIM cards integrated with the telephony subsystem (as described in Section 1.2.1), known as eSIMs, are used nowadays.

### Self-Check Questions

* What is firmware in a mobile device?

*Firmware is a type of software that is stored in a mobile device’s hardware and is used to control the device’s functions.*

* List three functions of firmware in mobile devices.
* *Firmware controls the hardware and software of a mobile device, ensuring that the device functions as intended.*
* *Firmware is responsible for booting up the device and preparing it for use.*
* *Firmware can be used to configure the device’s settings, such as the display, network, and power options.*
* *Firmware can include security measures to protect the device from malware and unauthorized access.*

## 1.3 Operating System

**Mobile operating system**

This is part of the core software that acts as an intermediary between the mobile user and the device’s peripheral components, to execute various instructions and input from the user to the mobile device.

The **mobile operating system** (OS) is part of the core software that acts as an intermediary between the mobile user and the device’s peripheral components, to execute various instructions and input from the user to the mobile device. At a low level, the mobile OS helps a user to run applications and makes computing easy with few constraints by making the mobile hardware run efficiently [Stallings 2012].

The mobile device market has diverse mobile operating systems available; as technology advances, the core functionality of secure devices has also been growing. Operating systems have not yet reached full parity with traditional desktop and legacy computers when it comes to the availability of management technology. The engineering of mobile technologies is designed differently from conventional desktop computers; this is because devices such as smartphones, laptops, and tablets were designed to run with very constrained resources. Consequently, the underlying software, known as the operating system, is designed to be different from that used in desktop computers [Chauhan Naresh 2014]. Each mobile device has a distinct OS depending on the different hardware stacks/capabilities it has. This section discusses the various characteristics of smartphones that make them different from desktop computers, focused on the mobile operating system.

The main themes that distinguish a mobile operating system from a desktop (or laptop) operating system are [Srikanth 2020]

* It is highly optimized to run in less powerful devices (smartphones, tablets etc.) in terms of processing power and power efficiency.
* It is optimized for a touch-screen user interface, in contrast to the traditional pointer-based user interface in desktops.
* Mobile OS’s have dedicated application stores where applications developed for that particular OS are maintained, while desktop OSs have more diverse software applications.

As for any operating system, the mobile operating system is composed of the kernel space and user space. While applications run in user space, the kernel is the central part of the operating system and is the first part to be loaded in the device. It facilitates interactions between the user space and the device drivers and underlying hardware. In addition to this, as we showed in the figure *Mobile Device Components* in Section 1.2, the mobile operating system includes an application “sandbox” that can protect unwanted interactions between applications [Franklin 2020].

#### 1.3.1 System calls

In mobile computing, a system call is a process by which an application on a device can interact with the phone programmatically, hence providing an interface between user input/instruction and the operating system***.*** This therefore allows software engineers and users and to develop mobile applications without worrying about the high-level **machine**/assembly language. They can easily develop apps (applications) on top of the OS without needing to employ low-level architectural controls, which require a lot of expertise to configure and run. This requires that the mobile OS has easy-to-use interfaces that allow it to access a wide range of mobile devices. However, since mobile devices are designed with portability in mind, their architectural design requires specific types of OS that cater for the vast capabilities mobile devices must achieve. Due to this difference in architectural design, smartphones, tablets, and personal digital assistants (PDAs) will normally run different types of mobile OS. Since by design a mobile device executes multiple instructions, they are designed to run multiple tasks in real time, and these instructions are required to be completed within a short period. This is achieved through the SoC and embedded software executed in tandem with the mobile OS to then issue instructions to peripheral devices and network software through device drivers. To achieve this, the mobile device has applications that are run on top of the smartphone OS. The OS by design considers the hardware restrictions of the device. To address this performance challenge, mobile device manufactures are continually upgrading their devices and associated OSs with new features [Chauhan 2014].

**System call**

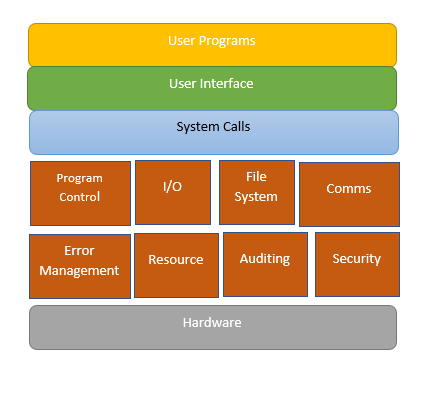
This is a process by which an application on a device interacts with the device programmatically, hence providing an interface between the user input/ instruction and the operating system.

At a hardware level, the mobile operating system uses a predesigned set of application programming interface (APIs) known as system calls. In mobile computing, a system call is a process by which an application on a device interacts with the device programmatically, hence providing an interface between the user input/instruction and the operating system***.***

Introduction to a System Call

A picture containing graphical user interface

Description automatically generated



Source: Panji Harawa [2023] based on Source: Shah, P. [2022, January 28].

Referring to the figure above, in mobile devices, system calls facilitate the interaction between applications running in user space and the underlying operating system running in kernel space, enabling access to privileged operations and system resources. When an application needs to perform such operations, it initiates a system call, causing a transition from user mode to kernel mode. The operating system’s kernel validates the request, executes the operation on behalf of the application, and returns the results. System calls provide a secure and controlled mechanism for applications to access system resources while maintaining the overall integrity and security of the mobile device’s operating system.

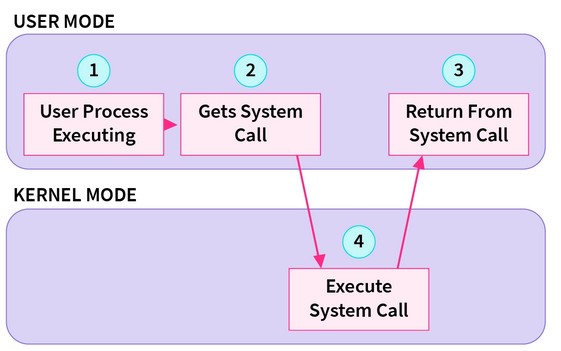
There are different types of system calls in an operating system. Each system call type and their function are shown in the table *Types of System Calls and Their Functions*.

Types of System Calls and Their Functions

|  |  |
| --- | --- |
| Types of System Calls | Function |
| File Management | Used to manage files by creating, deleting, opening, closing, read, and writing to files |
| Device Management | Used to manage device calls such as device writing, reading, and getting device attributes |
| Information Maintenance | Used for information calls such as read, set, and get system data, |
| Communication | Used for communication calls such as sending, receiving messages |

How does a system call actually work? When a user application is running a process in the user space and needs to access a hardware device or some OS resources to perform some computing, it makes a system call. The system call is pushed by the process to the system call handler, which runs in the kernel space. The kernel space executes the system calls sequentially and gives the results back to the requesting user space processes. This is shown in the figure *System Call Operation*.

System Call Operation



Source: <https://www.scaler.com/topics/operating-system/system-calls-in-operating-system/>

#### 1.3.4 Types of Mobile OS

There has been three-phased evolution in mobile operating system designs: the first phase was PC-based operating systems; thereafter an embedded OS took over in the second era; the third era saw smartphones taking over using integrated SoCs. This evolution has been backed by rapid architectural changes in the designs of the OS [Li et al 2012]. As highlighted, modern-day smartphones have been enhanced to better handle the core capabilities of a mobile device, such as memory, processes, and security. These functions have been inherited from legacy operating systems but have been improved over time. This has resulted in modern day computers that manage power efficiently, multitask, and support multiple sensors. The advancement of mobile operating systems has also seen optimizations to enhance user friendliness, delivering highly responsive, easy-to-use graphical user interfaces (GUIs). This section highlights some of the common and popular mobile operating systems and their features.

The ability for users to have access to the low-level hardware of a device is made possible by the operating system [Silberschatz et al. 2012]. Just like conventional computers’ OSs, there is a wide variety of mobile operating systems. These mobile OSs receive frequent updates, upgrades, and modifications to keep up with the needs of users. These updates are also especially important due to modern advances in hardware. However, in recent years we have seen modern mobile devices become defined more by the type of software they are running. The contemporary mobile operating system has a vast array of features and functionalities, and many companies have developed mobile operating systems to cater for the high-end needs of consumers/users. These mobile tech companies make phones that, by design, have features that vary between models of the device.

The most widely used mobile OSs nowadays are Android, iOS, Chrome OS, Windows phone, Tizen, and KaiOS. There has not been much research to date on Tizen and KaiOS as most discussions center on the two most popular operating systems, Android and iOS. However, these operating systems differ in their core architectural designs, their security, and many core features; in addition, significant differences exist between the traditional OSs and the emerging ones [Omelchenko et al. 2018].

Android OS is a mobile operating system that was developed by Google and is based on the Linux kernel. Its base implementation is offered open source under Apache License 2.0 [Android]; however, apps and drivers that provide functionality are bound to the platform on which they run and are proprietary solutions. Currently according to Gartner, it has the highest mobile OS market share in the world, with 70.79% [Marketshare 2023]. The first version to work with touchscreen mobile devices was released in 2011 as Android 2.3, while the latest version is Android 13 released in August 2022.

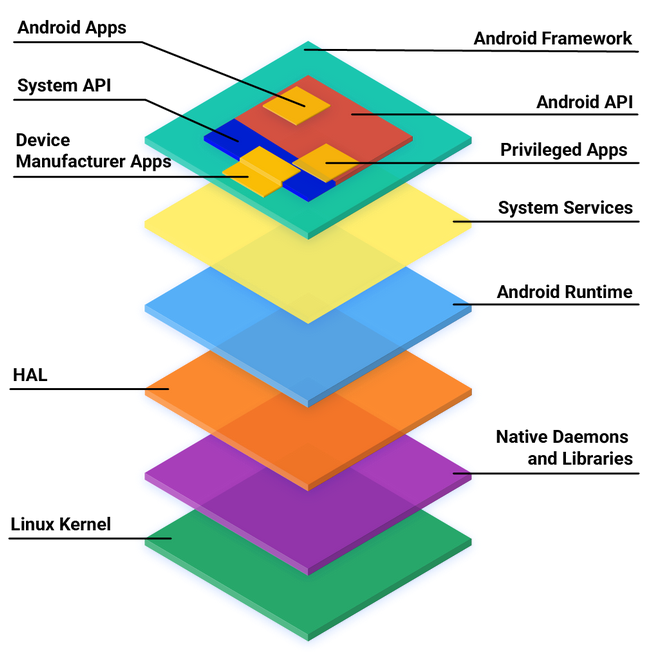
The second largest market share in mobile OSs is held by iOS, known as the iPhone Operating System. iOS was developed by Apple and, in contrast to Android, is a closed-source, proprietary solution. The first version was released in 2008 as iOS 1, with the latest release, iOS 18, planned in late 2023.

1.3.5 Example Case: Android Software Stack Architecture

The Android software stack architecture is open-source code that is provided by Google. It consists in a number of layers, which are shown in the figure *Android Software Stack Architecture* and can be summarized as follows [Arch 2023]:

* Applications are developed to interact with end users and, depending on the APIs they use, fall into three categories: (1) Android apps that are developed by third parties and have access only to publicly available Android APIs; (2) privileged apps that come pre-installed in the device and use both Android APIs and system APIs; and (3) device manufacturer apps, which are developed by the device manufacturer and have access directly to APIs provided by the Android framework.
* The Android framework is a set of libraries, interfaces, and APIs that is made available to the device manufacturer under a licensing agreement, while part of it is made available publicly for third party developers.
* System services provide communication between the Android framework APIs and underlying hardware, while the application runtime environment (ART) compiles the app’s bytecode into processor instructions.
* The hardware abstraction layer (HAL) is a hardware-agnostic abstraction API on top of the hardware platform. This ensures that all hardware functions are exposed similarly to higher layers, no matter what hardware is being used.
* Native daemons and libraries interact directly with the kernel and do not depend on the HAL.
* The kernel of the Android operating system is based on the open-source Linux kernel.

Android Software Stack Architecture



Source: Arch [2023].

### Self-Check Questions

* Name some system call types.
  + *File management system calls, device management system calls, information maintenance system calls, and communication system calls.*
* List four ways in which multithreading in mobile OSs is important.
* *By allowing multiple threads to execute concurrently, multithreading can improve the overall performance of a mobile device.*
* *Multithreading can improve the responsiveness of a mobile device by allowing it to perform multiple tasks concurrently.*
* *Multithreading allows different threads to share resources, such as memory and I/O devices, which can improve the efficiency of the device.*
* *Multithreading allows certain tasks, such as network communication and data synchronization, to be performed in the background without disrupting the user’s experience.*

## 1.4 Applications

**Mobile applications**, or “apps,” are designed to be used on a variety of mobile devices, including smartphones and tablets, and developers need to consider the context in which the app will be used. This includes the screen size of the device, its resolution, and its connectivity and power limitations. To create a successful mobile app, developers need to consider the needs and preferences of their target audience, as well as the constraints of the devices on which the app will be used.

**Mobile applications**

are a type of software application developed to be used on smartphones, tablets, and other mobile devices.

Developing a mobile app requires careful planning and design, as well as a thorough understanding of the platforms and technologies involved. It is important for developers to consider the user experience, performance, security, and other key factors when designing and developing a mobile app.

Overall, the portability and mobility of a mobile app can be crucial to its success, as it enables users to access and use the app anytime and anywhere [Harrison 2013].

As technology advances, mobile apps are also booming, as they are used daily for personal and professional purposes [Weichbrod 2020]. These applications are used in a broad range of areas, including education, entertainment, business, social networking, and transportation. They frequently have commonly used functionalities that support use cases of service delivery to users; hence, they often offer improved access to online services at reduced cost while also providing data security and increased safety. As such, mobile application use calls for quality usability assessments designed to check how usable and accessible an application is compared to alternative applications accessed using traditional computers. The use of mobile applications is increasing, and hence the number of developers is also increasing; however, some applications have seen their use diminish over time as users begin to experience various issues, such as unsubstantiated service claims, changes in app behaviors, or unfriendly human interaction with the app’s features [Evans, S et al. 2021]. According to the International Standards Organization (ISO), there are some standards that mobile applications should follow during the design phase. Mobile apps are expected to be usable, accessible, portable, reliable, easy to maintain, and secure. It is, however, noted that usability models are still not fully established as these are often isolated and not proven. This is despite the prevalence of mobile apps in health, education, e-commerce, and the provision of financial solutions [Attah et al. 2021].

Mobile applications can be used to perform various tasks for home, social, or office uses. This has made mobile applications contribute to a fast-developing sector in this information technology era. Most apps are quite easy to use and available for free, and some come preinstalled on the device as they offer core functionality, such as accessing the internet, making calls, and using social media. If an application is not available on a mobile device, it can be downloaded from a wide range of software markets available, with the dominant ones being the Google Play store for Android OS and the Apple Store for iOS. Technically, all these applications are managed by the operating system of the mobile device. Hence, mobile applications run through one or more of the following managed platforms: Android, Symbian, Blackberry, iOS, and Windows. On an abstraction level, these mobile apps run on Java/j2me, flashlight, brew, and Silverlight virtual machines.

#### 1.4.1 Mobile application types and development process

Mobile applications can be categorized into three different types based on the way they are developed and how they interact with the network:

* Native apps are developed solely for a specific mobile device and operating system. As such a native mobile app that is developed for a device running iOS cannot be used in an Android based mobile device, and vice versa. Usually, developers of native apps will use development tools provided by the relevant platform (e.g., Java/Kotlin for Android [Kotlin 2023]).
* Web-based mobile apps are developed to give the same experience as mobile apps when they are accessed via a mobile browser from a mobile device. This does not require a specific development of the app for the mobile device; however, the web server should support optimization for access by a mobile web-browser.
* Hybrid mobile apps are apps where native apps are combined with web technologies to provide the required service. In this case, for the mobile web client app the developer should use the specific development tools for the given platform (e.g., Kotlin for Android [Kotlin 2023]).

Depending on how mobile apps interact with the underlying software platform and device hardware, they can also be categorized as follows:

* Platform-specific apps are apps that use only the publicly available platform-specific API to interact with the underlying software stack and device hardware. In the case of Android, the Android apps have access only to the Android API [A-API 2023], which provides limited access to the underlying system services.
* Privileged apps are usually pre-installed apps that come bundled with the mobile device and operating system that is running on it. They make use of system-specific APIs that are publicly available as well as system-specific APIs that are made available by the device producer on a licensing basis.
* Device manufacturer apps use platform-specific APIs that are publicly available, system-APIs, and direct access to the HAL framework. These apps come pre-installed with the device. In the case of Android mobile devices, such apps make use of the Android APIs and system APIs, as well as having direct access to underlying hardware and software functionalities [Arch 2023].

Based on the reason for which apps are used, they can also be categorized as

• Communications: Internet access, social networking, emails

• Multimedia: Graphics, video, and audio players

• Productivity: Word processors, spreadsheets

• Navigation: GPS/maps, schedules, weather

• Utilities: Task manager, file manager

#### 1.4.2. Application runtime

The runtime environment is used by a mobile operating system to execute mobile applications. It is responsible for running the mobile app’s bytecode (compiled code) and managing its execution within the mobile operating system. Different platforms use different runtime environments and execution mechanisms. Each platform has its own runtime, designed to optimize performance and resource usage based on its specific architecture and requirements.

For example, iOS uses the Objective-C runtime for apps written in Objective-C and Swift [Objective-C 2023]. It employs just-in-time (JIT) compilation for Swift, while Objective-C uses a mix of ahead-of-time and JIT compilation. On the other hand, Android uses the Android Runtime (ART) environment and Dalvik (in older releases of the OS) [ART 2023]. In general, we can say that any runtime environment for mobile apps should support at least the following features:

* **Ahead-of-Time (AOT) Compilation:** A runtime environment uses an AOT compilation approach, which means that the app’s bytecode is converted into native machine code during installation. This results in faster app startup times since the native code can be directly executed by the device’s processor without further interpretation.
* **Improved Garbage Collection:**  A runtime environment should introduce a more efficient garbage collection mechanism that helps manage memory more effectively, reducing pauses during app execution and improving overall system performance.
* **Support for Native Code:** A runtime environment should support executing native code written in C/C++ using the native development kit (NDK) for the specific platform. This will allow developers to use native code for performance-critical parts of the app.

### Self-Check Questions

* Name three main features of Android Runtime (ART) environment.
  + *Ahead-of-time compilation, improved garbage collection, and support for native code.*
* What are some of the standards of mobile applications?
* *The user experience (UX) is the overall feel of the app and how it is perceived by the user.*
* *The performance of a mobile app is critical, as users expect apps to be responsive and perform well. This includes fast loading times and smooth transitions.*
* *Mobile apps should be compatible with a wide range of devices, including different screen sizes, resolutions, and hardware configurations.*
* *Mobile apps should be secure and protect sensitive data, such as user credentials and personal information.*

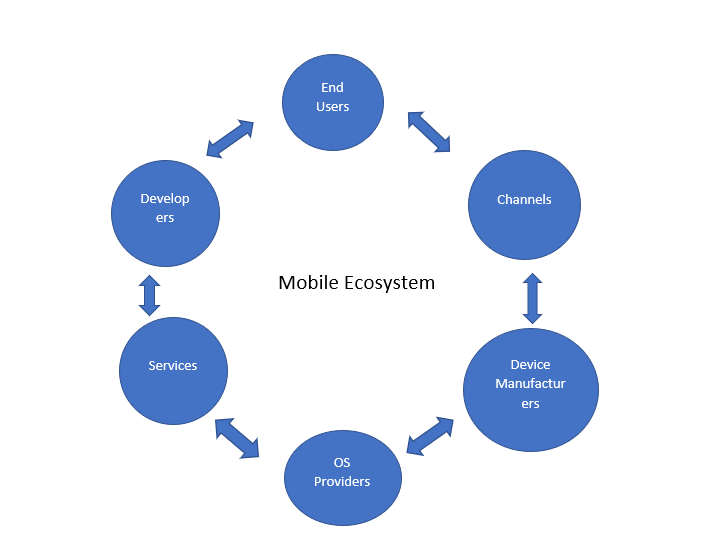
## 1.5 Ecosystem

The mobile application ecosystem is composed of different components, stakeholders, and factors that interact closely to contribute to the development, distribution, and use of mobile applications. It consists of the entire lifecycle of a mobile application from its creation to its distribution and usage by end users. Though the mobile application ecosystem is dynamic, the main stakeholders and components of the mobile application ecosystem are shown in the figure *Mobile App Ecosystem* and include [Xu 2016]

* **App Developers:** Individuals or companies that create mobile apps, ensuring app functionality and user-friendly interface.
* **Mobile Operating Systems:** The core software platforms that power mobile devices, such as Android, iOS, and Windows Phone (now Windows 10 Mobile). Each operating system has its unique app distribution channels, guidelines, and development tools.
* **App Stores:** Digital platforms where users can discover, download, and install mobile apps. Examples include the Apple App Store (iOS) and Google Play Store (Android).
* **App Distribution Channels:** The process of submitting mobile apps to app stores for approval and making them available for download by users.
* **App Users:** The end users who download, install, and use the mobile apps on their devices.
* **App Security and Privacy:** Ensuring that mobile apps are secure and protect users’ data and privacy. This involves implementing secure coding practices and adhering to platform-specific security guidelines.

Once the application has been submitted to the mobile app store, other processes followed by the app developer include app monetization (the way developers will generate revenues from their work (advertisements, subscriptions, etc.)), app maintenance and bug fixing, and app analytics to track app usage and user behavior.

Mobile App Ecosystem



Source: Panji Harawa [2023].

#### 1.5.1 Android Ecosystem

The Android app ecosystem consist of an interaction between the main developer of the open-source Android operating system (Google), equipment manufacturers, app developers, and users. Google, in addition to the operating system, which is based on the Linux kernel, has provided the framework for app development [Kotlin 2023]. The core stakeholders in the Android ecosystem are users, developers, app stores, and the Open Handset Alliance, which is composed of Google itself, original equipment manufacturers (OEMs), and carriers [Xu 2016]. The Android ecosystem stakeholders are shown in the figure *Main Android Ecosystem Stakeholders.*

Main Android Ecosystem Stakeholders



Source (<https://www.geeksforgeeks.org/android-ecosystem/>).

The Android ecosystem is decentralized as Google provides only the operating system to device manufacturers through a license, which manufacturers can tailor to their devices [Xu 2016; DC 2023]. Devices come with some apps preloaded from the manufacturer. Google also provides an Android app store named “Google Play” through which applications for Android can be distributed. However, Android users can download and install apps from third parties as well, such as the Amazon app store and Xiaomi app store.

In addition to stakeholders, the Android ecosystem includes a number of components including hardware devices, software applications, and services, as well as security measures. While the hardware devices that run Android range from mobile smartphones to smartwatches and TVs, the number of software applications in the Google Play store has reached 2.7 million [STA 2023]. Google offers a number of services for Android users that come preloaded in the device, such as Google Play Services (apps for Google Maps, Google Drive, and Google Analytics), Google Assistant (a digital assistant to device users), and Google Pay (a payment assistant for end users).

#### 1.5.2 iOS Ecosystem

In contrast to the Android ecosystem, which is more decentralized, the iOS ecosystem is fully centralized by Apple. Apple provides the iOS operating system, it provides the devices (iPhones or iPads), and it provides the only app store. The only decentralized part is the development of iOS apps, which is done by different third parties, rather than Apple itself. However, such apps coming from third parties cannot access the private and proprietary API, which is provided only for the preloaded Apple apps [DC 2023]. While in the Android ecosystem other app stores are allowed, for iOS devices only the Apple App Store, the official distribution app store, is allowed, limiting user choice and increasing the centrality of the ecosystem.

Given the centralized nature of the iOS ecosystem, the security of apps distributed via the App Store is high due to the vetting process that each app must pass before it becomes available in the store. Currently, there are more than 2 million apps available for users in the App Store [APP 2023].

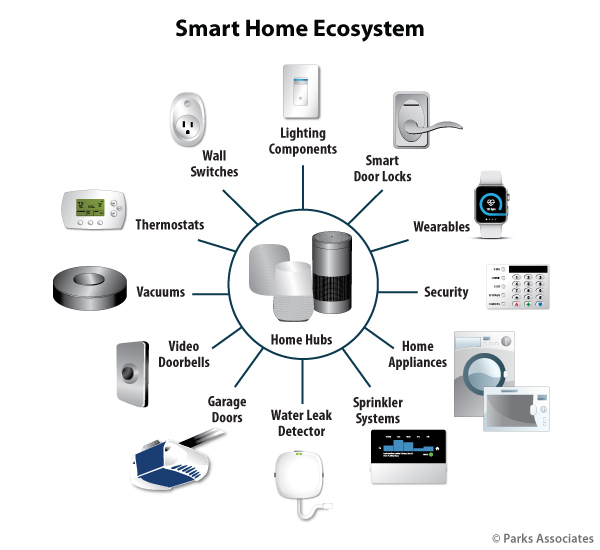
#### 1.5.3 Connected device ecosystem

As the number of connected devices has progressively grown, mobile devices have also drastically changed how users perceive, interact, and engage with mobile applications, their products, and the services they offer. As the mobile ecosystem has grown, we have seen a variety of devices, such as mobile phones and tablets, using various software and operating systems designed and developed by multiple companies passing data, sharing apps, and communicating between users from one device to another. App software designers often consider the cohesion that connected devices create, coupled with a user experience that provides accessibility and usability for such apps, as creating a flexible environment that enhances efficiency and dynamicity.

The device prioritizes enhancing the user experience with other connected mobile devices, making it seamless for users to meet their needs across various daily applications. This is possible since most of these devices can connect to each other and share data – including in voice, text, or video format – across different operating systems. This is possible because the collections of such devices create an ecosystem. This then enables the various components to “talk to” each other through a secure cloud, from which, through mobile network carriers or channels, users can install the required software applications for various online or cloud services, which users can then subscribe to. The manufactures of mobile devices, such as Sony, Nokia, XIOAMi, and Apple, therefore contribute to this by allowing mobile apps to be managed by OSs [Lauren et al. 2015].

Connected Device Ecosystem

Diagram

Description automatically generated

Source: Panji Harawa [2023] based on <https://www.parksassociates.com/report/race-control-ecosystem> .

The figure *Connected Device Ecosystem* shows a detailed mobile application ecosystem of connected devices. In the diagram, smart devices are talking to each other through a centralized hub that the modern technology stack has established and the interconnection of devices such as mobile phones, laptops, and smart watches enables. Through this interconnection, devices can interface and share data to make informed decisions or carry out tasks. For example, in the event of a water leakage, you would get informed on your smart phone or smartwatch via an application that receives data from your home cloud. Alternatively, in the event of increased temperature in a room, the connected thermometer can interact directly with the window-opening system to cool down the room based on the outside temperature retrieved from internet services. These are just some examples of how an interconnected mobile ecosystem can help in everyday life.

### Self-Check Questions

* Identify key stakeholders of the Android ecosystem.

*Google, OEMs, Users, and App developers.*

Summary

The mobile technology stack is built on an advanced software and hardware architecture that enables rich ecosystems of applications and enhanced mobile security. This unit establishes that the key hardware components in mobile computing are integrated into a system on a chip (SoC), including the central processing unit (CPU), graphics processing unit (GPU), image/multimedia processing unit (ISP), digital signal processor (DSP), video encoder/decoder, memory processor, and modems/NICs. These components work together to enable various functionalities and maintain device security. As technology evolves, SoCs are becoming increasingly smaller, driving competitiveness in hardware manufacturing. One cannot talk about hardware without reference to mobile firmware, a type of ROM software, which plays a crucial role in managing low-level programs in mobile devices. It is part of the hardware stack and handles initializing and loading device drivers. The unit further highlights how firmware also provides high-level abstraction for secure storage and execution of sensitive cryptographic computations, ensuring protection against mobile threats. Overall, understanding the mobile technology stack, including hardware components and firmware, is essential for software engineers and designers to be able to develop and enhance smartphone user experiences. However, the ability for hardware and firmware to easily interface would not be possible without the mobile OS, which serves as an intermediary between users and mobile devices, enabling efficient hardware operation and user-friendly application execution. With multiple mobile operating systems in the market, there is a growing emphasis on security and management technology. Mobile devices like smartphones, laptops, and tablets were designed to run with limited resources, so their OSs differ from those in desktop computers. Over the years, operating systems, such as Android and iOS, have evolved to offer improved memory management, enhanced security, and increased user-friendliness. They also support power efficiency, multitasking, and multiple sensors, as well as dedicated mobile applications designed for smartphones, tablets, and other mobile devices. Mobile apps are used in various domains, including education, entertainment, business, and social networking. Developers must consider user experience, performance, security, and other factors when designing and developing mobile apps. They must also meet international standards that allow them to be deployed on platforms such as Google Play Store and Apple Store. As user expectations increase, there is a growing need for sophisticated mobile applications across various industries. In this unit we showed the main stakeholders in both Android and iOS ecosystem. While the Android ecosystem offers a decentralized approach, where Google offers only the operating system and app development platform, while devices are developed by OEMs and apps by third parties, the iOS ecosystem is highly centralized by Apple, which provides all devices, the iOS operating system, and the only app store.

# Unit 2 – Hardware

**Study Goals**

On completion of this unit, you will be able to …

… distinguish how RF technology works in mobile devices.

… describe what a trusted execution environment is.

… learn about how biometric devices enhance security and prevent mobile threats.

… understand how location technologies work/function.

## Introduction

The hardware stack of mobile devices has connectivity at the center of it; with the coming of the internet, we have seen mobile devices used increasingly both for personal and business tasks. However, just like computers, mobile devices are also susceptible to attacks due to their ability to connect to various networks and each other. Hardware plays a significant role in ensuring that security is maintained and mobile threats are minimized. However, the security of most components is also challenging as running robust security systems depends on high computational power as well as an understanding of the hardware ecosystem. For instance, all mobile components already employ proven technologies to enhance device security. However, there is a need to align the components and their independent conventional security mechanisms at the hardware level in addition to the embedded hardware-based mobile security integrated circuits (IC) used to prevent or minimize vulnerabilities and attacks. Key elements of mobile device hardware include the system on a chip (SoC), which houses components such as the central processing unit (CPU), modems/network interface card (NIC), radio frequency (RF) modules, and portable digital assistant (PDA) modules. RF modules enable wireless communication between mobile devices and networks, while PDA modules provide essential functionality for personal digital assistants. These components not only contribute to the device’s functionality but also collaborate to support security measures such as encryption, secure boot processes, and trusted execution environments (TEEs), ensuring comprehensive defense against potential threats.

### 2.1 RF Modules

The mobile device would not have become omnipresent in our lives without its ability to connect seamlessly with different remote systems and other mobile devices. The connectivity of mobile devices is supported by the radio frequency (RF) module, which enables wireless communication in the different technologies that the mobile device supports. Nowadays mobile devices support a multitude of wireless communication technologies, ranging in scale from satellite communication support (GPS), through cellular network technologies (4G/5G) and mid-range communication technologies (WiFi), to short-range communication (Bluetooth, near field communications (NFC), and radio frequency identification (RFID)). We will next dig more into the concept of radio frequency, the RF module components, and the basics of wireless technologies incorporated in modern mobile devices.

#### 2.1.1 Radio frequencies (RF)

A radio frequency (RF) signal is generated as an alternating current by a transmitter and propagates through the communication medium, whether it is wire-based or wireless. For the wireless transmitter, this signal is emitted by the antenna into the air. In an ideal vacuum environment, the RF signal will be emitted at the speed of light at 299 792 458 m/sec (~300 000 km/s). An RF signal radiates away from the antenna with a continuous pattern and is governed by the wave’s properties, such as amplitude, frequency, and phase.

Thinking of the RF signal like a sinusoidal wave that travels away from the antenna, the wavelength is defined as the distance between two successive peaks or valleys of the sinusoid. The frequency is inversely related to the wavelength. The following formula describes the relationship between the wavelength, λ, the frequency, f, and the speed of light, c:

*𝜆=cf*

where the unit of λ is meters, the unit of c is m/s and the unit of f is hertz [Hz].

Different wireless technologies operate in different frequency bands, e.g., the Institute of Electrical and Electronics Engineers (IEEE) 802.11 wireless fidelity (WiFi) based technologies operate at 2.4 GHz, 5 GHz, and 6 GHz frequency bands; cellular networks operate at 900 MHz, 1.8 GHz, 1.9 GHz, and 2.1 GHz (among others); Bluetooth technology operates at 2.4 GHz [Tanenbaum 2021], while NFC and RFID operate at 433 MHz, 2.4 GHz, and 5 GHz [Want 2006]. Different frequency bands are used for different reasons, e.g., the lower the frequency, the greater the communication range; the lower the frequency, the higher the penetration in indoor environments. For example, for a cellular network operating in the 900 MHz band, the coverage zone is much greater than for cellular networks operating in the 1700 MHz band.

#### 2.1.2 RF Module Components

The RF module in the mobile device is responsible for supporting communication using various technologies. Some of the RF module components are shared between different technologies and include the following [Egan 2004]:

**RF transceiver:** converts the digital data to analog and vice-verse depending on the direction of operation (transmit/receive).

**Antenna:** transmits and receives the radio frequency from the wireless medium. A single antenna cannot operate in the different spectrum bands; hence, typical mobile devices employ several antennas, one for each communication technology (cellular, WiFi, Bluetooth, etc.).

**Power amplifier:** increases the signal power before transmission to prepare it according to regulatory rules for transmission power levels on specific channel and bands. E.g., the maximum transmit power in the 2.4 GHz band by a WiFi module should not exceed 30 dBm.

**Filter:** is used to separate the desired signal from other signals in the spectrum band. This can be done in either the digital or analog domain. Usually, a filter is used to filter out any signals from other channels and keep only the signal desired in the operational channel.

**Low-noise amplifier (LNA):** is used on the receiving path of the RF module to amplify the weak received signal without amplifying the noise, thus improving the signal-to-noise ratio before the signal is processed by the baseband processor.

**Baseband processor:** handles the signal according to the technology used. It performs modulation/demodulation of the signal, handles the physical layer protocol used by the technology, and performs error correction and encoding/decoding.

#### 2.2.3 Modulations

In communication engineering, the modulation process is achieved by varying certain parameters of the carrier signal based on the inputs of the modulation signal. The carrier signal is a continuous electromagnetic wave, while the modulation signal can be a continuous signal or a discrete signal. Analog modulation modulates the carrier wave with a continuous signal (such as audio or video signals), while digital modulation modulates it with a discrete signal that is composed of 0s and 1s. Basic analog modulation techniques are amplitude modulation (AM), where the amplitude of the carrier is changed according to the modulation signal; frequency modulation (FM), where the frequency of the carrier is changed according to the modulation signal; and phase modulation (PM), where the phase of the carrier is varied based on the modulation signal. The last two are known as angle modulation techniques. Basic digital modulations are amplitude shift keying (ASK), frequency shift keying (FSK), and phase shift keying (PSK). In the following sections, we will discuss each of the digital modulation techniques. The transmitter block that performs the modulation process is called the modulator, while the receiver block that performs the reverse process is called the demodulator.

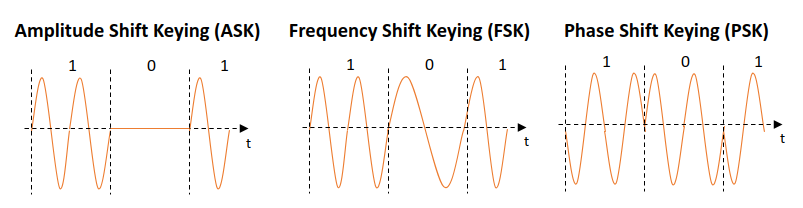
#### 2.2.3.1 Digital Modulations

The basic digital modulation technique is amplitude shift keying (ASK). In this modulation technique, each of the two binary values (0 and 1) are represented by a different level of amplitude of the carrier signal. In the figure *Digital Modulation Techniques,* an example of ASK is shown where one amplitude is 0, representing binary 0, and the high amplitude represents binary 1. Even though ASK is simple to implement, it is susceptible to interference and multipath distortion. Moreover, due to channel propagation characteristics (multi-path, noise, path loss, etc.), a constant amplitude cannot be ensured; thus, ASK is not used for wireless communication.

The most appropriate modulation technique for wireless communication is frequency shift keying (FSK), where a shift in the frequency of the carrier signal represents each of the binary values (0 and 1). The simplest way to implement such a modulation technique is to switch the transmission of the carrier signal between two different oscillators with different frequencies. Then the demodulator will detect the frequency changes of the signal to detect the transmitted binary value. FSK is more robust than ASK, as it is less susceptible to channel propagation characteristics. However, on the other hand it requires a greater bandwidth compared to ASK. The figure *Digital Modulation Techniques* illustrates FSK modulation.

Like FSK, phase shift keying (PSK) uses shifts in the angle of the carrier to modulate data. In this case, it shifts the phase of the carrier signal for each modulated 0 or 1. The figure *Digital Modulation Techniques* illustrates PSK modulation, where a phase shift of 180 degrees represents a change from 1 to 0 and a phase shift of –180 degrees represent a change from 0 to 1. PSK that uses such phase shifts is called binary PSK (BPSK). Compared to FSK, PSK is even more robust to wireless channel propagation characteristics; however, it requires a more complex modulator and demodulator as both transmitter and receiver need to be synchronized in frequency as well as in phase.

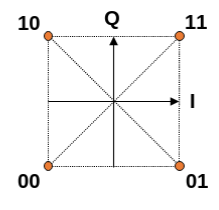
Digital Modulation Techniques



#### 2.2.3.2 Advanced shift keying

To improve the bit rate of modulation techniques, multiple levels of varying parameters are used. E.g., to transmit two bits at once (four different possible symbols), quadrature PSK (QPSK) is used. Instead of using only two possible phase shifts (+−180 degrees), QPSK uses four different phase shifts (+−90 and +−180 degrees) to allow it to modulate two bits at once. The constellation diagram is shown in the figure *QPSK Constellation Diagram*. The constellation diagram shows all the possible positions of a signal in the quadrature (Q) and in-phase (I) planes. QPSK modulation requires less bandwidth for the same amount of data transmitted compared to BPSK. However, it is more complex in implementation, and the transmitter and receiver need to be synchronized using a reference signal. In QPSK, the transmitter will select chunks of data and modulate them as phase shifts compared to the reference signal. To avoid the need for a reference signal, differential-QPSK (DQPSK) is used, where the phase shift of the current two bits is determined based on the position of the previous two bits. Other higher-order PSK schemes use mode phase angle shifts to modulate the data, achieving higher bit rates.

QPSK Constellation Diagram

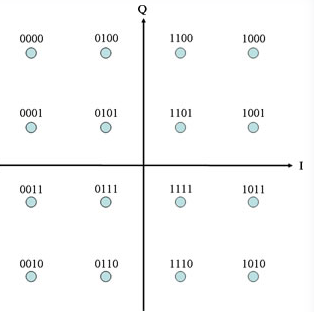


**Frequency spectrum**

This is the range of frequencies that can be used by a signal.

To use the assigned bandwidth more efficiently and increase the bit rate, PSK can be combined with ASK to achieve a higher number of varying parameters. In the figure *16-QAM Constellation Diagram*, 16-quadrature amplitude modulation is shown, where for each symbol, four bits are transmitted. It uses 12 different angle shifts and 3 different amplitudes. The number of bits per symbol for X-QPSK can be calculated at *log2⁡X*. In current wireless communications (e.g., IEEE 802.11ax), higher-order modulation schemes up to 1024-QAM are used to improve spectrum efficiency and data rate.

16-QAM Constellation Diagram



Source (<http://ecelabs.njit.edu/ece489v2/lab5.php>)

**Dynamic spectrum management**

This a process of using intelligent cognitive software techniques on concepts in network information and software definition to improve communication [Salhab 2016].

### Self-Check Questions

What are the components of a radio frequency?

*Amplitude, frequency, and phase.*

Name some of the RF module components.

*RF transceiver, antenna, power amplifiers, filters, LNAs, and baseband processors.*

Which varying parameters are used to modulate the signal in QPSK?

*QPSK uses both the amplitude and phase variations to modulate the signal.*

How many bits per symbol are transmitted in 1024-QPSK?

* 4
* *10*
* 5
* 12

## 2.2 PDA Module

The use of mobile devices has reached unprecedented levels, and the numbers continue to grow; this has indeed been a mobile computing revolution. Statista has estimated that as of 2022, the global number of mobile devices had reached 6.5 billion [Statista 2022]. There is, however, an expectation that this number may increase further; this has resulted in more devices being designed to be more sophisticated and have modern functions for use as day-to-day gadgets. However, within the scope of limitations on hardware and software that come with mobile devices, there are some challenges that have only been resolved using portable digital assistants (PDAs). This type of device is designed to enhance how users interact with mobile devices by making those devices more responsive to the provision of information to the user in the context that the information is required. This is because PDAs, often referred to as virtual assistants, have functions that are responsive to natural language. This includes functions such as completing tasks using applications and accessing online web services.

Due to the responsiveness that PDAs have, they have been quite useful in sectors such as tourism and the controlling of devices and gadgets at home and in the office [Edu et al 2020]. Recently there has been a lot of development of PDAs with the coming of modern and major virtual assistants such as Siri [Bellegarda 2014], which is commonly used in Apple desktop and mobile devices; Google Now, commonly used in Google products and services; and Alexa, an assistant developed by Amazon and referred to as Amazon Alexa Voice AI. The arrival of these modern technologies has changed the landscape of PDAs as they have been useful in helping users to get both proactive and reactive assistance using natural language and commands. Kepuska and Bohouta [2018] add that PDAs are more useful in providing proactive assistance to users by using the device’s agent to conduct an action or task without the user’s direct request, whereas the agents in PDAs can also reactively aid by being given voice-activated commands or simply by typing. As seen in Statistica [2022], the surging number of users of smartphones are also using PDAs as the features are also highly integrated with most smartphones, as well as in other gadgets such as tablets, and headsets; there is also a high level of integration in operating systems.

The use of these PDAs has also evolved significantly, and these mobile devices can easily be customized to adapt to individual personalities as they can be programmed to determine a user’s activities online – such as their profiles, locations, and normal routines. This can give PDAs a proactive role, allowing users to interact with these devices using notifications or by responding to commands or questions. These are mostly done and activated using natural language processing, which creates a real-life simulation of the user speaking to a human assistant [Kepuska & Bohouta 2018]. It has been noted that PDA’s relays proficiently users’ intents using the numerous services and mobile applications as it has become an intelligent feature within the mobile devices’ meta layer. This is because a user can either be explicitly detected when commands are actively given to the PDA to execute a particular action, or this can be inferred by the virtual assistant by detecting or suggesting results after evaluations triggered by some pre-existing conditions. The architectural design of PDAs applies concepts such as machine learning, text to speech, inferencing, analytics, data mining, and voice recognition systems, all integrated. Using this combination of technologies and applications, PDAs have become useful in assisting users to easily set up tasks such as alarms/reminders; getting users organized; and giving them access to online structured webservices and information, such as locations. Tasks can also involve automating users’ experiences through the use of collective functionalities, including the organization of a user’s information and data; for instance, a PDA can automatically scan flight-related information and organize all the details related to a flight, such as GPS coordinates to check flight status and weather forecasts, and notify the user accordingly. [Edu et al. 2020].

Ideally, PDAs are required to be personal, so that they can assist users; this requires identifying their own personal interests and needs, which will be unique and specific to every individual. However, challenges have still been encountered when responding to users’ commands for several reasons. First, as PDAs require the use of many machine learning and language processing models, there is inadequate data for virtual assistants to accurately model users’ experiences and interests. The limitations come from the fact that they are expected to only be used for and to understand basic user commands; hence its experience and feedback is limited only to this. Second, user behaviors also cannot be distinctively modeled due to low computing power, as machine learning requires a lot of computing power, and the PDAs do not have the capacity to do this. Third, it is not easy for PDAs to distinguish the user’s interests from those of the agent/platform; hence the mobile system can struggle to put the user’s interests over those of its other actors. Lasty, PDAs are challenged when it comes to the creation of content; this is because they will only conform to specific coded user interactions and thus may not be able to perform or generate the actual content the users may want [Sarikaya 2017]. Despite these challenges, however, PDAs have been adapted to take them closer to personalization by the use of more sensors that allow more learning, as well as the integration of cloud computing to enhance the quantity and quality of the data the PDAs can get access to, enhancing their models of user experiences and interests. Due to an extended connection to cloud, PDAs have been able to more fully represent users’ interests through access to user profile data, such as bio data and favorites, in addition to interests explicitly shared by the user. The cloud connectivity has also allowed for an increase in digital footprint, as well as the creation of the time and space to establish and retain more of the users’ interests, hence facilitating the generation of more personal responses.

## 2.2.1 PDA’s Agent System Architecture

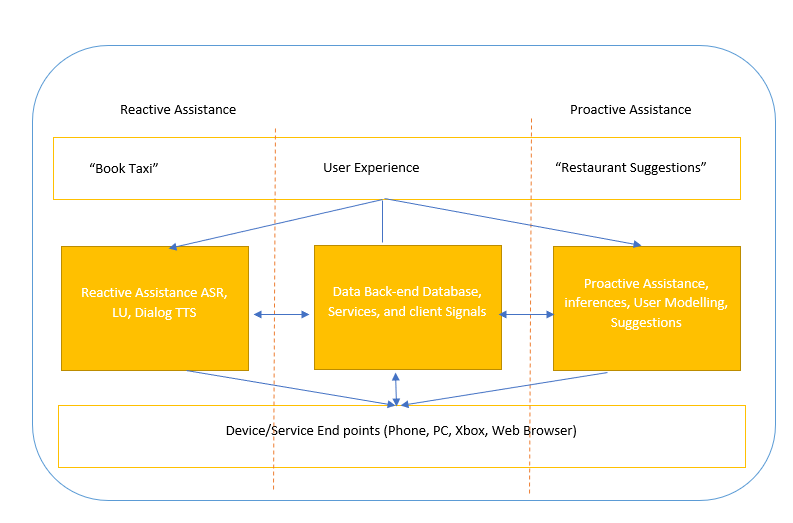
The design of the PDA support structure has two distinct modes of operation: **Proactive Assistance** and **Reactive Assistance**. This can be seen from the system architecture diagram (figure *The PDA Agent Architecture*) on how the two types of agents work.

The PDA Agent Architecture

Diagram

Description automatically generated

The PDA Agent Architecture For (A) Reactive Assistance And (B) Anticipatory Computing



Source: Panji Harawa [2023] based on Sarikaya, R. [2017].

The figure depicts two instances of the agent’s data and service points: in panel a, a user can explicitly give a command to the agent to book a taxi; the reactive assistance components work together after checking all explicit natural language queries to connect to the database and serve the user. In addition, the digital agent will identify what is assumed to be the expected valuable output and contextualize the results. The agent will then infer and apply user models and check previous experience from the endpoints.

### Self-Check Questions

* 1. Please name two common features and capabilities of PDAs and how they differ from those of smartphones.
* *Personal organization: PDAs are often used as digital organizers.*
* *Communication: a PDA can send and receive emails and text messages.*

## 2.3 Trusted Execution Environment component

In the ever-evolving landscape of mobile technology, security has become a paramount concern for users and businesses alike. As mobile devices increasingly handle sensitive information, from personal data to financial transactions, safeguarding these devices against cyber threats and unauthorized access has become a critical imperative. To address these security challenges, the concept of the trusted execution environment (TEE) has emerged as a key solution.

A trusted execution environment (TEE) is a secure and isolated area within a mobile device’s central processing unit (CPU) that provides a protected environment for running sensitive applications and processing critical data. It operates independently of the device’s main operating system (OS) and is characterized by a high level of security and tamper-resistant properties. The TEE leverages a combination of hardware- and software-based security mechanisms to ensure the confidentiality, integrity, and authenticity (CIA) of data and applications residing within it.

The TEE’s primary objective is to enhance mobile device security by creating a secure enclave within the device’s hardware, where sensitive operations can take place even in the event of a compromised OS and apps running on top. This isolation helps mitigate security threats associated with malware, unauthorized access, and attacks targeting the mobile device.

#### 2.3.1 TEE Architecture and its key features

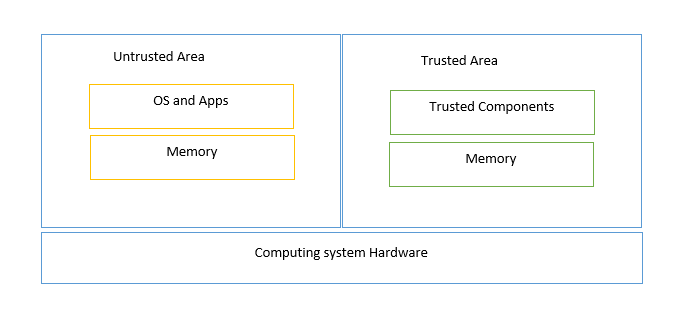
**Trusted execution environment (TEE)**

This is a mechanism that provides a computational environment that is isolated and safe for applications to be executed in a ***mobile phone system*.**

Trusted Execution Environment (TEE) General Architecture

Graphical user interface, diagram

Description automatically generated



Source: Panji Harawa [2023], based on Hosam and BinYuan [2022].

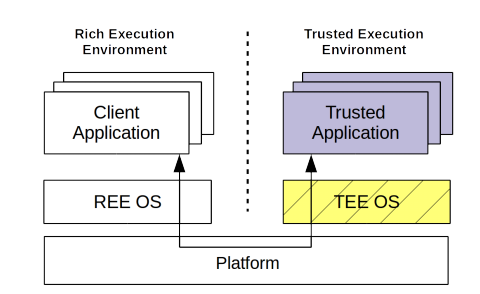
As shown in the figure TEE general architecture, the mobile computing ecosystem and the structure of a generic TEE platform is divided into two components, the trusted and non-trusted areas. These two environments are separated, with one (trusted) part dealing with operating systems (OSs) and specific secure applications, whereas general mobile applications are executed in their own untrusted environment(s). This makes it easier for security engineers to know which areas are untrusted and which are the trusted software stacks. As such, the trusted environments are also referred to as TEEs, whereas the untrusted environment is also known as a rich executionalenvironment (REE).

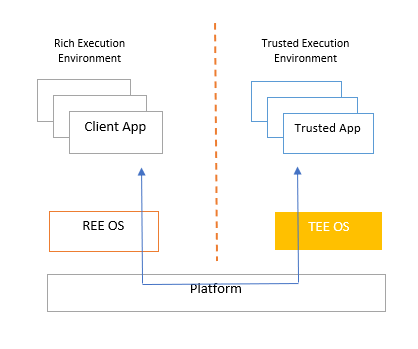
The key features of the TEE architecture can be summarized in four categories [Ekberg 2014]:

* Isolation: The TEE operates in a separate, isolated environment from the main OS, ensuring that sensitive operations and data are protected from interference by malicious software or other parts of the system.
* Secure Data Storage and Access: The TEE provides a secure storage space, often in the form of a trusted file system or key store, where sensitive data can be encrypted and stored.
* Secure Communication Channels: The TEE enables secure communication channels between trusted applications and external entities running in REEs.
* Secure Execution Environment: The TEE ensures that critical applications, such as mobile payment solutions, authentication services, and digital rights management (DRM), run in a secure execution environment.

#### 2.3.2 TEE versus REE

REE Versus TEE





Source: Panji Harawa [2023], based on Brian et al. [2015].

The figure *REE vs. TEE* illustrates the application of specific industry standards for trusted components, which facilitate addressing security issues tailored to various application ecosystems. Essential to this approach is the collaboration between TEEs and REEs. TEEs provide confidentiality, integrity, and authenticity by isolating security-critical tasks from the main operating system or the REE, which runs most applications and services but is more exposed to vulnerabilities. By working together, TEEs handle sensitive operations, such as cryptographic processes, secure storage, and secure boot, while REEs manage the execution of everyday applications and services. The early days saw the use of processors securely as a TEE guideline for smartphones and tablets using the ARM trust zone [Zhao 2014], which among other things provided a standard for mobile processing that is still being used today. As seen from the figure, there are number of critical components that mark trusted pillars and security mechanisms; these are common to any device and are used in creating a trusted environment [Brian et al. 2015].

**Trusted Computing Base** (TCB)

This is a part of a mobile device that consists of hardware and firmware that by default are supposed to be trusted without ***any limitations/ restrictions on a device***.

It is envisioned that to access standard interfaces, there will still be a need for further implementation of TEE on several platforms. This will, however, be facilitated by ensuring that these trusted components are accessible to everyone, hence increasing the use of trusted technologies in the development of mobile applications and embedded systems. This will eventually make it easy for engineers to make more solutions available, which will impact the standardization, use, and deployment of TEE for mobile devices, including its application in the lightweight trust platforms used in mobile computing.

The TEE has a component built upon it called the trusted computing base (TCB). This is a part of a mobile device that consists of hardware and firmware that, by default, are supposed to be trusted without any limitations or restrictions on a device. It is therefore important to note that the critical hardware components that form the base for the TEE have core parts (shown in grey) and non-core parts [Ekberg et al. 2014].

#### 2.3.4 TEE security mechanisms

The security mechanism employed by the TEE can be broken down into several critical components that require attention when it comes to dealing with mobile threats at the hardware level. These components include boot integrity, secure storage, device identification, isolated execution, and device authentication capabilities [Ekberg 2014].

The first component deals with the integrity of the platform. Such an integrity check can be achieved by ***secure booting*** or ***authenticated booting***. Secure booting includes a verification step that runs an early detection process to check if there has been any modification to the platform. If any modifications are detected, the device booting process is halted. The devices ensure this by implementing code signatures: A boot process ensures that some cryptographic code matches with previous stored values, hence making the boot sequence unchangeable. The device will only achieve this if the operating system executes from memory, which checks the boot code hashes that will be signed and has its public key stored safely for verification on the next boot process sequence [Chai et al 2014].

Once the signature validation of the boot loader is finalized, the next component of the software stack, in this case the kernel, is launched. It is through this process that the TEE ensures there is no tampering with the environment, since these hashes are stored in a read-only memory (ROM) part of the kernel. Secure booting is therefore achieved on the TCB by ensuring that the pillars of the block have an immutable booting order that is already verified, together with integrity-protected cryptographic algorithms. In the authenticated boot case, the only difference is that the platform components are loaded and measured but there is no verification. The bootloader measures the state of each component launched in turn one after the other; the information found is then used to fix any components that may have been modified [Kirkpatrick et al. 2012].

To prevent attackers from accessing sensitive information, the TCB employs a second component, secure storage, which requires one confidential and device-specific key. Usually such a key is initialized at manufacturing time in the processor memory [Ekberg 2014] and is accessed through the authenticated encryption mechanism. To maintain the integrity of data even in the event of rollback attempts, nonvolatile memory that retains its state during the device’s boot sequence is utilized for secure storage.

The next component is isolated execution of processes for certain REE applications. This is achieved by letting the TEE expose functionalities to the REE applications as predefined cryptographic algorithms, while ensuring that the cryptographic keys do not leave the TCB [Truong et al. 2021].

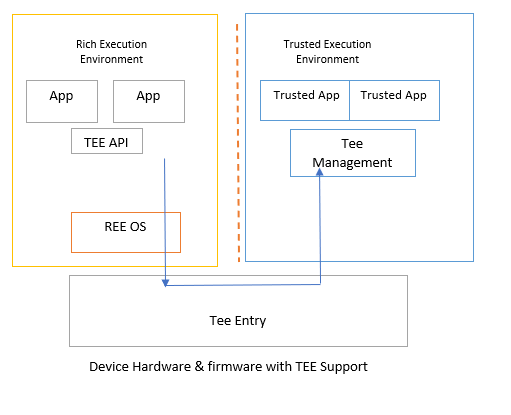
The last component of basic security mechanisms of mobile devices is the use of device authentication, which is one of the mechanisms relied upon by service providers; this entails the use of manufacturers’ information as a standard for compliance through the creation of unique identities. Different manufacturers of the hardware components of mobile devices use this mechanism to embed a unique, unchangeable serial number as an identity. These are often randomly generated identifiers that follow a unique pattern as per the manufacturer’s specifications. Using base keys and a root, a verification process can be established that will bind these uniquely assigned numbers to uniquely identify a device or hardware component. This mechanism is used for generating international mobile equipment identifiers (IMEIs) as well as the hardware-coded network link layer identification commonly used for accessing wireless networks (WiFi), such as MAC addresses and wireless connectivity (Bluetooth). The authentication is done in such way that digital certificates are generated for the devices by the manufacturers that bind the identity of a device or hardware component to the device’s public key, which provides authentication of the device with external entities [Asokan et al. 2014].

Device Hardware and Firmware with TEE Support

Diagram

Description automatically generated

Source: Asokan et al. [2014].



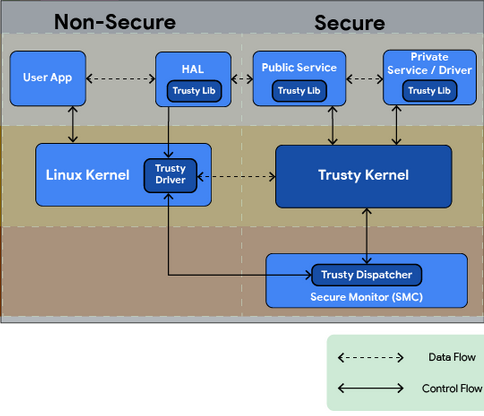
Source: Panji Harawa [2023], based on Asokan et al. [2014].

From the figure *Device Hardware and Firmware with TEE Support,* we can see that for a high-level architectural virtualization, there are indeed multiple components that are required to achieve an ideal TEE. From the figure, any isolated REE can be considered a TEE instance; this is because ideally a TEE architecture can be designed purely to use specifically designed software chips and then to use virtualization and processing to create separate executional instances for the TEE, which can be accessed using the entry interface of the TEE. This makes it easy for applications that are not trusted to be executed and run by connecting to an API exposing TEE services through the device’s OS, hence allowing such applications to be able to read and write to the TEE.

#### 2.3.5 Example of TEEs

There are several examples of technologies that support TEE in the market. One of them is Trusty TEE, which is used by Android devices to provide a secure operating system [Trusty 2023]. The architecture layout of Trusty TEE is shown in the figure *Trusty TEE*. As in the generic architecture, Trusty TEE consists of secure and non-secure environments. The secure environment is isolated from the rest of the operating system by both software and hardware means. The Trusty TEE is composed of the Trusty kernel, Trusty driver, and Trusty libraries. The Trusty kernel exposes the functions of its Linux kernel via Trusty driver, which supports data transfer between the Trusty kernel and the rest of the Android OS. Trusty libraries are used by the Trusty applications to communicate with the kernel driver. Via Trusty libraries, third party developers can choose to execute certain parts of their application processes in the Trusty environment. An example of such a use case would be a set of proprietary algorithms for online banking. Trusty TEE is compatible with ARM processors, and it uses ARM TrustZone to create the secure trusted environment at the processor level.

Trusty TEE



Source: [Trusty 2023].

Another example of a security mechanism in mobile devices that uses hardware TEE is the Samsung KNOX mechanism [Kanonov 2016]. This fully leverages the hardware TEE provided by Samsung mobile devices or the ARM TrustZone. It provides mechanisms for boot-time defense, load-time defense, run-time defense, and update-time defense [Samsung 2015]. At the hardware level, it provides secure boot and trusted boot processes, roll-back prevention, and device root key. The last of these is used to provide cryptographic operations associated with the device.

### Self-Check Questions

* What are the main roles of TEEs and REEs in mobile security?

T*EEs provide a secure environment for executing sensitive tasks and storing critical data, while REEs manage general applications and less sensitive tasks.*

**Biometric devices**

These employ a technology that uses and measures the physiology of a human body, such as fingerprints or facial recognition.

## 2.4 Biometric Devices

**Biometric devices** employ a technology that uses and measures the physiology of a human body, such as fingerprints or the use of hand gestures. In the modern era, it has proven useful to provide high device access security due to its accuracy in identifying individuals. The technology is based on the fact that human beings are all unique, in both physical appearance and behavior. Biometric devices harness this feature to provide an extra layer of security to mobile devices by distinguishing these unique traits to identify users [Ximenes et al. 2019]. For many mobile applications, such as those from the banking sector and mobile payments, there is a need for authentication of users, and to ensure security, most devices opt for using more secure biometric identification rather than the legacy approach of using a password. Biometric identification simply refers to the verification or use of a person’s physiological or behavioral traits to provide a secure identification and authentication to a device, which can be done using facial recognition, fingerprints, iris scanning, or voice recognition [Chen Y et al. 2022]. It is well known that smartphones have employed biometric devices to enhance security; this is because the use of biometric devices has proved to be more secure than the use of generic password-based authentication systems. Consequently, there have been attempts to use biometrics-based authentication that can be accessible remotely. However, this use of biometrics is realistic only in real-time and requires accurate time synchronization. The proposed mechanism of using time synchronization was deemed to have security loopholes as it was vulnerable to hacking – which made it easy for someone to predict an online or offline password – and server manipulation.

Now, since the use of biometrics has become more common, many smartphone applications have this feature enabled to enhance security as an add-on to the use of traditional passwords. Using this technology, mobile payments have advanced so that they have become more convenient than using traditional smart cards. The innovation has made it easy for companies such as Google (Google Pay) and Apple (Apple Pay) to provide virtual currencies and virtual credit cards as payment gateways that are integrated within most mobile applications. Achieving privacy and security in this scenario is very hard, as when a user has a virtual credit card, they must store the credit card number in their mobile device, which possess a security risk the device falls into the wrong hands. This can, however, be mitigated by using remote biometrics-based authentication, where a user’s biometric features, such as fingerprint, iris, or voice recognition, can be used to verify the users’ information to a biometric server through a bounding server [Yichen et al. 2020].

#### 2.4.1 Types of biometric authentication

Biometric authentication can be broken down into various types since it is designed to use the physical features of a person; such features include use of fingerprints, use of eyes (iris), and voice recognition [Timothy et al. 2016]. It can also be extended to use behavioral traits, like signatures, gait, and keystrokes. Biometrics have proved to have several advantages over generic passwords:

* Biometric keys cannot easily be lost.
* Biometric keys cannot easily be duplicated, copied, or shared.
* Biometric keys are inherently difficult to forge.
* Biometric keys are not easy to guess.
* Biometric keys cannot easily be broken/hacked.

Uniquely, biometrics’ use of human physiological features makes them almost immutable, and every person has unique traits that can be distinguished from person to person. As previously noted, the most-used biometric technology has been fingerprints, which have been used for decades because of their easily distinguishable features. More recently, we have seen other technologies emerge, such as face recognition, voice recognition, hand geometry, and iris recognition [Mohammad et al. 2020].

The original and most-used biometric authentication technology is fingerprint recognition. This technique uses sensors that are small and often embedded in mobile devices such as smartphones, laptops, and even desktops. The device identifies the patterns of ridges and valleys on someone’s fingertips and matches them against an existing record within a database. The patterns that are generated are uniquely distinguishable and can only belong to one person. As such, at a single touch of a finger, identification and authentication is established within the shortest period. However, during registration of fingerprints, any good fingerprint device requires multiple samples to ensure the accuracy of the results and prevent authentication errors. The only challenge with this type of authentication device comes when a user has unclear fingertip valleys and ridges that make it difficult for the device to match or read.

The other most-used biometric authentication technique is the use of voice recognition technology, which uses audio to distinguish individuals. The technology employs a blend of behavioral patterns associated with sound, including the formulation of accounting patterns that can discern the size and shape of a person’s throat. This information is then combined with matching of the user’s voice pitch and speaking style. This type of technology has some disadvantages: it is easily affected by any changes in the tone and pitch of the user’s voice, and if there are any background noises it can be difficult for the device to authenticate [Lai, X. 2021].

The next biometric used to authenticate users on mobile devices is face recognition. This technology creates a model that picks some of a user’s key immutable features from their face and creates a reference image having features that cannot easily change over a period. This requires ensuring that it skips over some problematic features, such as facial expressions, which can make authentication difficult for mobile devices. To create an easily identifiable model, at least six samples are required, and these are created by the technology to enhance authentication [Asmitha et al. 2022].

Another user authentication method that is based on behavioral biometrics is gait recognition. Gait recognition identifies different people based on characteristics of their walking by analyzing several features, such as stride length, walking speed, and the angles of movement of different body parts. A mobile device first needs to create a gait signature of the user. This is done by collecting information from the mobile device’s sensors (accelerometer, gyroscope, rotation vector, step counter, step detection, etc.), training the data model, and extracting the user’s gait features [Axente 2020]. Then, once the gait signature of the user is created and stored, it can be compared continuously against information collected from the mobile device’s sensors to take a decision as to whether or not the active user is the owner of the mobile device.

**2.4.2 Biometric system for verification and identification**

All the above biometric devices follow a similar pattern when it comes to identification and verification of biometric data. As such, an effective biometric system needs to consider some important aspects, such as how users will be registered and enrolled in the system that is being used, and the different use cases for such systems – whether they may be used for identification or only for verification of data.

A biometric system used for verifying will authenticate the user’s identity by checking against their record in a biometric database that they were enrolled in. This database has a format that is preconfigured for efficient processing of biometric references; the system compares the data on a one-to-one basis to confirm the identification, and the result can either be a verification or a rejection.

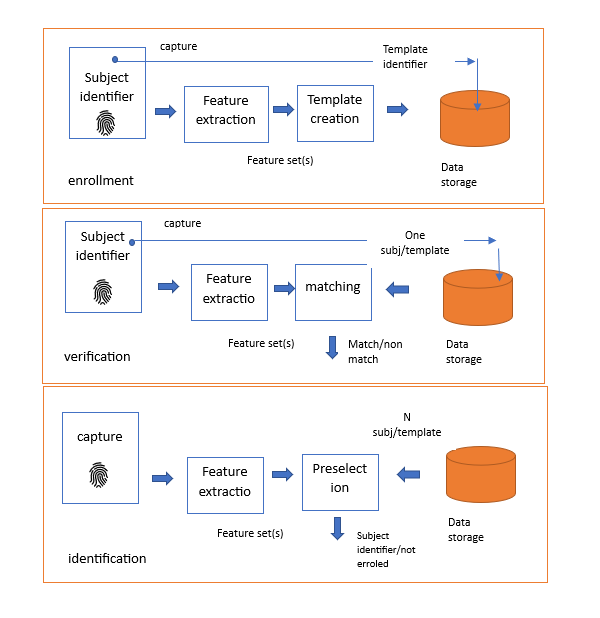
In contrast, a biometric system designed for identification must search the entire database the user was registered on to check for any matches. It therefore makes comparisons against all the records in the database in a one-to-many scenario as it must ensure that it uniquely identifies the user and authenticates them against all other identifiers. Hence making use of biometrics for identification is harder than their use only for verification [Lafta et al. 2013].

Biometric System Enrollment

Graphical user interface

Description automatically generated with medium confidence

Source: Lafta et al. [2013].



Source: Panji Harawa [2023], based on Lafta et al. [2013].

The figure *Biometric System Enrollment* demonstrates how biometric systems can be used for identity checks and what it takes to use them for verifying users. First, all users need to be enrolled into a particular database, which can then be checked either for identification or verification of the users. The process of identifying or verifying has two stages: capturing and extracting. These processes require that the biometric device receives a digital copy of the biometric feature in use, which generally varies in format depending on the type of physical feature being used. For example, for a fingerprint, the scanner will sense and generate an image that is then used as a sample within the fingerprint system. This sample is used as raw digital copy, and features are extracted by a feature extractor within the biometric device for use in an enhanced search for preselection and matching against the database [Kolda et al. 2019].

#### 2.4.2 Attacks against biometric authentication

As with any other authentication method, even biometrics-based authentication is prone to cyber-attacks that can breach system security. Based on the way the cyber-attack is performed, attacks on biometric authentication can be classified as follows [Rui 2018]:

* Spoofing or faking the sensor: In this type of attack, the attacker tries to spoof the sensor by presenting fake biometric features, such as a fake photo, a fake fingerprint mold, or a fake voice recording.
* Reply attacks: In this type of attack, the biometric sensor is fed with a replay of a previously recorded biometric feature that was presented to the system. This mostly only applies to systems that do not check the liveness and dynamics of the biometric features. Such replay samples can be stolen by the attacker through network eavesdropping at the time the information is provided to the authentication system.
* Network server attacks and biometric data breach: In this type of attack, attackers might try to gain control over the server containing the biometric data or manipulate the data on the server. This relates more to network attacks, such as server hijacking and database injection, rather than to the biometric sensor itself.

To reduce the potential for attacks in a biometric authentication system, it is advised to use multi-factor authentication – either a combination of biometric authentications or biometric authentication with traditional authentication (passwords, pins, tokens); to always encrypt the biometric data when they are shared over the network during the registration/enrollment phase; and to implement liveness checking methods to detect if the biometric data is coming from a live person and not from a static source.

### Self-Check Questions

* What are the two main steps involved in using biometric systems for identity checks?
* *The two main steps involved in using biometric systems for identity checks are enrolling users into a particular database and then checking the database to either identify or verify the users.*

## 2.5 Location Technology

Location technologies in mobile devices refer to the use of various systems and methods that enable devices to determine and share their geographic location. They have become an integral part of modern mobile devices and are used by multiple mobile apps that provide location-based services, ranging from localization and navigation maps, through safety and emergency services, to social media apps and food-ordering apps. Based on the environment in which the location technology works, they can be classified into outdoor localization technologies and indoor localization technologies.

Outdoor localization technologies include the Global Navigation Satellite System (GNSS) – known also as the global positioning system (GPS) – and cellular-based localization using cellular technologies. Indoor localization includes WiFi-, Bluetooth-, and RFID-based localization systems. Though each indoor technique can also be used for outdoor localization, the accuracy might be reduced depending on the conditions of the network deployed (e.g., the number of WiFi access points or Bluetooth anchors). There are also other localization techniques based on ultra-wide band (UWB) communication and low-power communication [Coppens 2022], but since these are not usually integrated with mobile devices, we will not cover them in this section. A summary of each of the relevant localization systems follows [Yeh 2009; Zafari 2019]:

* The global positioning system (GPS) is a satellite-based navigation and localization system that uses satellite communication to determine position on the Earth’s surface. In highly dense urban environments, GPS location can be enhanced with data from the cellular network to increase localization accuracy, as the GPS signals can be obstructed.
* The WiFi positioning system (WPS) leverages WiFi signals to estimate a device’s location, usually based on received signal strength (RSSI) value from multiple WiFi access points.
* Cellular location technologies leverage triangulation techniques and other cellular-based methods to estimate a device’s location based on its proximity to nearby cell towers.

Different localization technologies use different methods. Based on the methods and techniques that they use, location technologies can be classified as follows [Zafari 2019]:

* RSSI-based localization, where the distance from anchors is determined based on the received signal power given that the transmit power is known.
* Time of flight (ToF) localization, where the distance from anchors is determined based on the ToF of the signal. This requires accurate time synchronization between anchors and mobile devices.
* Time difference of arrival (TDoA) localization, where the location is determined based on the difference in time of arrival from different anchors. This does not require accurate time synchronization between devices.
* Angle of arrival (AoA) localization, where location is determined based on the angle of arrival of the signal.

#### 2.5.1 Global Positioning System (GPS)

GPS is a satellite-based navigation and localization system that is owned and managed by the US Government. GPS is a medium earth orbit (MEO) satellite system orbiting at ~20,000 km above the Earth. It has 32 satellites, out of which 27 are operational, covering the whole surface of the Earth [Tanenbaum 2021]. Nowadays every mobile device is equipped with a GPS radio receiver that is used for real-time localization purposes.

The technique used by GPS to locate mobile devices is based on time-of-flight (ToF) and trilateration methods. But how does this work in principle? To determine a location, GPS uses the following steps [Crato 2010]:

* Message transmission: All the GPS satellites are time synchronized with nanosecond accuracy [GPS 2023], and each satellite will broadcast a message periodically where it includes its position and the time of transmission (ToT) of the message.
* Determining ToF: When a mobile device receives the message, it timestamps the time of arrival (ToA) of the message. Then based on the ToT and ToA, the mobile device determines the ToF.
* Distance calculation: Since electromagnetic signals travel at the speed of light, knowing ToF allows the mobile device to determine the distance from the satellite itself. The ToF calculation is shown in the figure *GPS ToF calculation*.
* Position calculation based on trilateration: Trilateration is a geometric method to determine the position of a point based on distances to some other given points. In GPS, trilateration involves intersecting spheres centered at each satellite position with radius as the distance calculated in previous step. To determine the single position in 3D, four such spheres are necessary. One satellite will locate the device on a sphere; adding a second satellite will locate the device on a circle (eclipse) that is the intersection of two spheres. A third satellite will locate at just two points as intersection between three spheres. Thus, a fourth satellite is needed to accurately locate at a single point in 3D.

To improve GPS localization accuracy, in certain parts of the world (the USA and Canada), differential GPS (DGPS) is employed. The DGPS consists of fixed Earth stations that calculate the location based on GPS signal coming from satellites. DGPS fixed Earth stations calculate the position difference between their own fixed position and the position determined based on GPS signal reception. This difference is then transmitted in their coverage areas to help other mobile GPS devices to improve their localization accuracy.

### GPS security

GPS inherits the security threats of its satellite systems. Due to their large coverage zone, satellite communication systems are prone to eavesdropping and jamming attacks. The latter can render satellite communication services nonexistent in certain regions of the world. Other vulnerabilities of satellite systems include spoofing and relay/replay attacks. For example, GPS jamming attacks have been reported to be effective in changing the accuracy of the GPS system [Lenhart 2022]. It has also been reported that even with off-the-shelf hardware, relaying signals over large distances to achieve GNSS spoofing is possible [Lenhart 2022].

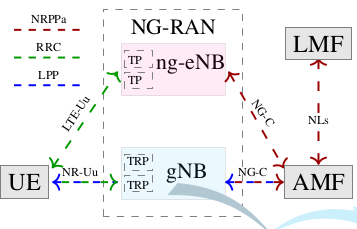
Jamming attacks are denial-of-service types of attacks against receivers in proximity. GPS signals are weak when reaching the Earth’s surface; hence, it is quite easy to perform jamming attacks. With as little as a 1 W transmit power in the GPS operation band, one can attack an area of 35 km [Papadimitratos 2008].

Spoofing attacks are the transmission of forged signals that make the receiver think the location is different than the actual one. To perform a spoofing attack, the attacker will generate a GPS satellite signal that includes a ToT synchronized with the GPS system; however, the position of the satellite is faked in the message. In the case of replay attacks, the attacker will record the GPS signals from satellites and replay them from another position to attack victim devices [Lenhart 2022].

#### 2.5.2 Localization based on cellular networks

Mobile devices can also use localization services offered by the cellular networks. The localization service offered by cellular networks has improved with the new generations of cellular networks, and in the 5G standard it supports an accuracy of several meters down to a few decimeters [Dwivedi 2021]. The localization service architecture in 5G is shown in the figure *Localization in 5G*. The mobile device can be localized using the new radio (NR) communication with the gNB as well as via communication with the LTE eNB. Localization information is exchanged via transmission reception points (TRPs) in gNB and transmission points (TP) in eNB. The base stations exchange the necessary information with the location management function (LMF) via the access mobility function (AMF) module of the 5G core network using the NR positioning protocol annex (NRPPa) signaling protocol.

Localization in 5G



Source: Dwivedi [2021].

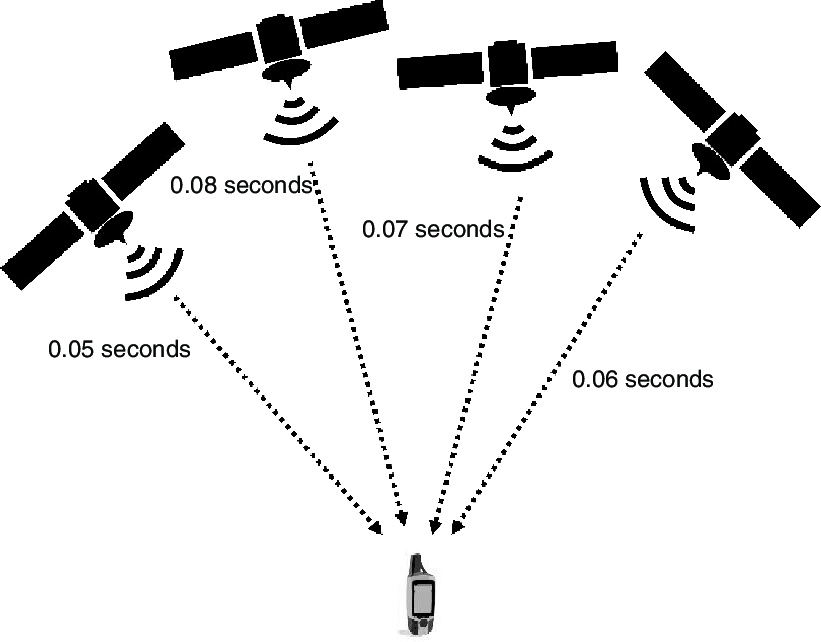
The algorithms that are implemented by the LMF are diverse and are left to operators to choose. Some new positioning methods that have been introduced in 5G cellular networks include [Dwivedi 2021]:

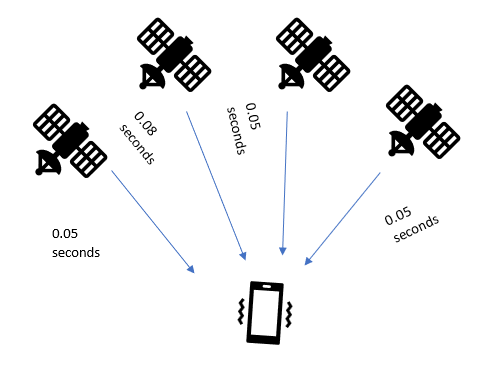
* Multi-round-trip time localization technique, where the localization is determined based on RTT measurements with multiple gNBs. Since RTT measurements are done in relative time, they are prone to network time synchronization errors.
* Downlink angle of departure (DL-AoD) and uplink angle of arrival (UL-AoA) are methods based on the transmitted/received signal angle to/from the mobile device.

In the case of the multi-RTT localization technique, the LMF initiates the localization procedure by requiring the gNBs to perform multiple Tx-Rx’s with a specific mobile device by transmitting in downlink a positioning reference signal (DL-PRS) and receiving in uplink the sounding reference signal (UL-SRS). Both UE and gNB report the measurements on their side to the LMF, which runs the multi-RTT algorithm.

In the case of DL-AoD, the mobile device will provide the beam angle of the DL-PRS to the LMF, while the gNB will provide the azimuth and elevation of its antenna. In the UL-AoA method, only the gNB will inform the LMF of the AoA of the UL-SRS.

GPS ToF Calculation





Source: Panji Harawa [2023], based on Lafta et al. [2013].

### Self-Check Questions

* How many satellites are needed for GPS mobile device to determine location in 3D?

*4 satellites.*

Summary

The unit aims to provide students with the ability to understand the RF technology used in mobile devices, the trusted execution environment (TEE), biometric devices for secure authentication, and location technologies.

In this unit, we highlight the need for communication between mobile devices and how this is done via radio frequencies. First, radio frequency is defined and its main properties – amplitude, frequency, and phase – described. Next, we describe the main components of the RF module and their roles, including RF transceivers, antennas, power amplifiers, LNAs, and baseband modems. We show the frequency bands they operate on based on the different technologies used by the mobile device (GPS, cellular, WiFi, Bluetooth). Then we dive into the basic modulation techniques and multi-order quadrature amplitude modulation (X-QAM).

The increasing use of mobile devices has led to the development of portable digital assistants (PDAs), which are designed to enhance user interaction with mobile devices and provide responsive information through natural language processing. PDAs such as Siri, Google Now, and Alexa have become useful in sectors such as tourism, controlling devices and automating user experiences. While challenges exist in adequately modeling and understanding user experiences and interests, PDAs have been adapted to improve personalization using sensors and cloud computing. We describe how PDAs offer both proactive and reactive assistance, making them an intelligent feature within the mobile devices’ meta layer; they use machine learning, text to speech, analytics, data mining, and voice recognition systems to assist users in setting up tasks and accessing online services.

The unit goes on to discuss the trusted execution environment (TEE), a mechanism that provides a secure computational environment for mobile phone applications. The TEE comprises hardware components that enable applications to execute in a separate secure area. The TEE is divided into two environments, the trusted and untrusted, and is designed to restrict access to sensitive data, such as cryptographic keys. The unit highlights how the TEE employs four critical components to prevent tampering of the environment, including secure booting, secure storage, execution of applications and services in an isolated environment, and device authentication.

The unit further analyzes how we use biometric devices and how biometric authentication has become increasingly common in mobile devices, replacing traditional password-based systems. Biometric technology uses unique human physiological or behavioral traits, such as fingerprint, iris, voice, and facial recognition, to provide secure identification and authentication. Biometric keys are difficult to forge, guess, duplicate, or break/hack, providing advantages over password-based systems. The unit continues by discussing the most widely used biometric technologies and their advantages and limitations. It also explains the differences between biometric systems designed for verification and identification and the process of capturing and extracting biometric data for authentication or verification.

Finally, we discuss location techniques used in determining the geographical position of mobile devices. We define localization methods based on RSSI, AoA, ToF, and TDoA. We dive into the GPS and explain how it works based on ToF to determine the distance of the mobile device from multiple satellites and then based on the trilateration method to determine the actual position in 3D. In addition, potential GPS security attacks (signal jamming and spoofing) are described. In conclusion, we describe localization based on the cellular network: 5G incorporates two new methods – multi-RTT localization and DL-AoD/UL-AoA.

# Unit 3 – Android Operating System

**Study Goals**

On completion of this unit, you will be able to …

… determine core components of the Android hardware architecture.

… understand how the bootloader of the Android OS functions and enhanced mobile security.

… identify key sandboxing and virtualization processes of the Android OS for mobile applications.

… understand how the mobile device kernel provides security in the Android OS.

… determine how code signing of applications enhances security in mobile devices.

## Introduction

Android is a mobile operating system that has predominantly taken over the mobile devices landscape. Developed by Google, this mobile device software has 71.47% of the mobile operating system landscape, with an estimated two billion users worldwide [Statista 2022]. As such, there are several features that have caused the Android operating system (OS) to be widely adopted, one of which is that it was designed and developed to be an open-source system. By design the OS has several key features that are similar to other common open-source Linux-based operating systems for both mobile devices and desktop computer systems. This section introduces the Android OS and provides a deeper understanding of how it is designed and developed, considering the underlying frameworks and all other related software and hardware features of the OS. The Android OS ecosystem has grown over the years due to its ability to be adaptive and thus enable many startups to contribute to this ecosystem by developing mobile applications. As a result, the mobile technology device landscape has changed significantly, and we have witnessed a paradigm shift in mobile computing technology, both in the designs of its hardware architectures and in software applications. The modernization of the mobile space has been in large part due to emerging modern software operating systems and their companies and the accompanying large ecosystems of mobile applications. As such, the Android OS has pioneered advances in mobile technology as this software has become the most used mobile platform in both volume and market share [Statista 2022].

## 3.1 Hardware

The Android OS has a Linux flavor to it, having been developed by an alliance known as the Open Handset Alliance, which comprises companies such as Samsung, Intel, and Sony and is led by Google. Google later developed and deployed the software freely and openly for mobile use. As Linux kernel-based software, the architectural hardware of the OS provides a platform for software developers to develop applications that can be installed on any mobile device that is running Android OS, regardless of the underlying hardware used by the device. These hardware features of the Android OS made it possible for designers to take advantage of its cross-platform features, an approach that is also common across Linux based operating systems [Boyer et al. 2016]. This is possible because the OS created an ecosystem for developers to be able to re-use some of the software components without the need for redevelopment. At the software level, the Android OS source code being open source allows mobile tech companies to modify and reengineer it accordingly to suit their hardware requirements as it is hosted by Google using a public open-source license under the public license [Prateek Singh 2020].

Since the Android OS is mostly available for free, it has some features that companies take advantage of to modify as per their need; the resulting OSs are then mostly installed and run by various mobile companies, including Samsung, Sony, and Huawei. This situation makes it easy for companies to take advantage of its cross-platform features to develop functionalities that are unique to each of the major carriers’ flagships. The Android hardware features create a platform for easy running of apps due to similarities in the hardware architecture. Security is enhanced differently in each such architecture, using application provenance, application permissions, application isolation, and encryption mechanisms [Shuangbao Paul Wang 2021]. However, at the hardware level, the Android OS provides high-level encryption within the kernel to prevent and protect from mobile threats by using mechanisms such as controlling mobile app permissions and access to the device, or the use of high-level isolation, depending on the environment [Mohamed and Patel 2015].

In the Android OS, the software components are mostly integrated with some key hardware components. Hence, at an architectural level all hardware is integrated with a software component that manages the mobile device’s hardware components, such as microphones, speakers, and GPS, to facilitate management and use of these tools. The Android OS also uses embedded hardware to assist technologies that enable high-level graphical user interfaces (GUIs), providing support for development of applications with modern front-end technologies. Mobile devices also come with near field communication (NFC) technology embedded, which allows connectivity between two devices by holding them closer to each other [Chin et al. 2016]. Furthermore, the Android OS includes some communication modules that are designed to support the functions of mobile device communication and network connectivity. All these packages and modules comprise integrated kernel elements and support libraries that make software development possible. The hardware packages also enable these applications to be executed as runtime processes [Collins et al. 2015].

The Android OS has the backing of Google, which manages it as an open-source project and allows for the developers of multiple original equipment manufacturers (OEMs), mobile network operators, and chip manufacturers to contribute to this project. The OS has a diverse ecosystem of mobile applications since most of the mobile applications are developed using Java, which has many libraries making it easy to use. This makes the OS easily compatible with most applications and their hardware driver capabilities, coupled with architectural features that allow easy memory management and general manageable services. However, this aspect of the Android OS varies between the different versions of Android developed by different OEMs. Hence the underlying technology stack is still dependent on using common application libraries that facilitate the creation of both native and hybrid mobile applications that work across platforms [Prateek Singh 2020].

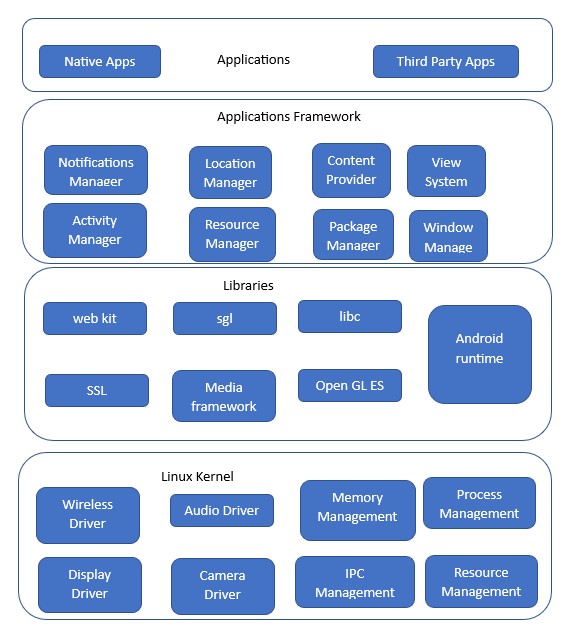
However, due to the various design modes of mobile applications, these apps present various security vulnerabilities that arise due to the mobile companies’ customizations. These vulnerabilities indicate that using a three-fold analysis that considers mobile application permission as a security mechanism, provenance of apps from the various application stores, and vulnerability analysis before apps are deployed is necessary to enhance security and prevent mobile threats. The hardware requires some components of the Android architecture to enhance security; its key role is that, through the software components, the CPU, GPU, and other hardware components use encryption to secure the device. The Android OS also uses hardware features to enhance security, such as the secure element (SE) and the trusted execution Environment (TEE).

Android Architecture

A picture containing text, calculator

Description automatically generated

Source: Panji Harawa [2023], based on Acharya et al. [2022].



The Android architecture as seen from the figure has several components broken down as managers for different services. By design each manager has a unique function, used to manage resources. For example, mobile devices employ notification services to display alerts from applications on the screen. Within the device architecture, content services or providers focus on securely sharing data between different apps. This is achieved through the use of APIs and interprocess sharing, enabling seamless data exchange between various applications. In the Android OS, applications create what are known as activities; these are application activities that are run to perform a specific functionality. For example, if an app would like to access a wireless driver for connectivity, it will initiate an activity that will go through the activity manager to access the kernel and initiate a call for the wireless connection. Through the same application framework, a design app that requires access to a GPS/location feature will run an activity executed and managed by the location manager service, which then generates a request for the app to use location. Depending on the user’s knowledge of mobile threats, they can permit the app to access the service, since by default GPS is normally deactivated for security and privacy reasons.

Every component of the Android OS serves a distinct service or function. The resource manager oversees core services, including graphical user interface components. Additionally, a separate high-level notification manager is responsible for handling all notifications. The sharing of user information across multiple platforms is handled by the content provider. Each running application in an Android phone runs what we call an activity, and each application has various activities running in the background. To determine location there is a separate package responsible for this and it requires setting the necessary permissions for the mobile phone to access location services (GPS) on a mobile device [Cai & Ryder 2017]. To protect the mobile device from threats, all native and third-party mobile applications are developed to request permission to access critical services in a mobile device. These services are managed by controllers, and the OS handles all such requests through user approvals. However, since the end user eventually approves or permits installation and access to services by apps, the requests still present high security threats and leave the OS vulnerable to attacks, as most users do not pay critical attention to most of these app requests [Faruki et al. 2015].

### Self-Check Questions

* How does the Android OS provide security for mobile devices?
* *One way is through application permissions: mobile apps must request*  *permission from the user before they can access certain features, such as the*  *camera or the microphone.*

## 3.2 Bootloader

The Android bootloader is an essential software component that serves as the foundation of the Android operating system. It is responsible for initializing the hardware, loading the operating system (OS) kernel, verifying the OS’s integrity, and starting the OS before it fully launches. Android bootloaders come in two forms, locked and unlocked. A locked bootloader restricts users from modifying the OS or installing custom ROMs, while an unlocked one allows for these actions but may void the phone’s warranty, introduce stability issues, and make it difficult to revert to the original OS.

There are several benefits to having an Android bootloader, such as enhanced security against malware and other threats, optimized performance through boot process streamlining, and increased customization options through custom ROMs or other modifications. However, there are also drawbacks, such as voiding the device’s warranty, causing stability issues, and increasing security risks when the bootloader is unlocked.

The Android bootloader operates in two stages. The first stage loads the device’s boot information into the read-only memory (ROM) at the architectural level, while the second stage involves the primary bootloader (PBL), which is integrated into the original equipment manufacturer (OEM) instances. The PBL is a hardware chipset manufactured by vendors, responsible for initializing the hardware, loading the OS kernel, verifying the OS’s integrity, and starting the OS. In contrast, the fast boot layer refers to the layer where booting interfaces provide security against tampering through secure storage, focusing on PBL integrity verification, OS kernel loading, and OS startup [Borzemski et al. 2019].

The bootloader processes undergo a secure boot process to prevent tampering with the system’s memory. This is achieved by having partitions on the Android device that are built in the kernel, making it read only. As the system boots, it caches as a core process in the boot process together with memory spaces; these are secured to prevent unnecessary modifications. Mappers in the devices assist in dynamic partitions of the memory, done through a component also found in the kernel. Dynamically, this results in metadata of the device memory partition storing blocks of information. The same data is used to vary all other blocks during boot initialization by comparing the block values with the virtual clocks that are created during dynamic partition [Yadav, et al. 2022].

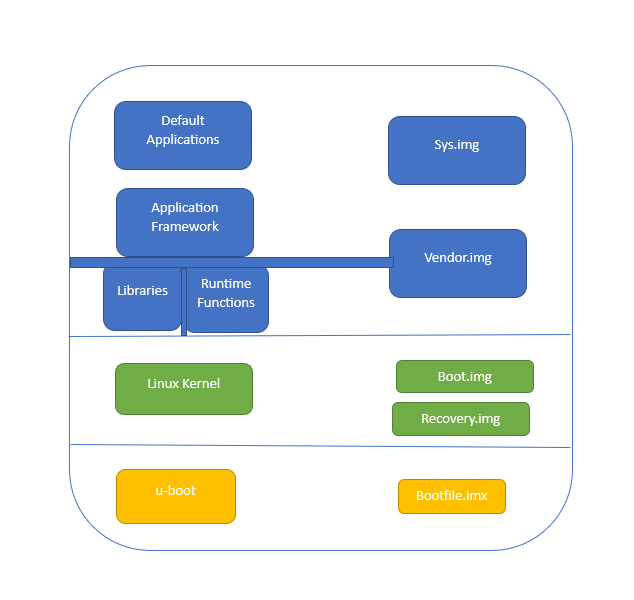
Diagram

Description automatically generated Android OS Bootloader

**Table**

This is a structure that is used to define the layout of the hard drive or other storage media on a computer.

Source: Panji Harawa [2023], based on Acharya et al. [2022].



Android Bootloader Steps

|  |  |
| --- | --- |
| **Step** | **Description** |
| Start |  |
| 1 | Load System Image: Link core libraries and APIs, load default mobile OS application systems |
| 2 | Load Vendor Image: Store device-specific protected binaries, store inaccessible binary files |
| 3 | Load Boot Image (Linux kernel): Mount required partitioning tables |
| 4 | Load Recovery Image (in recovery mode): Perform tasks like Boot Image, install packages and software updates |
| 5 | Load Uboot File.imx: Execute device bootloader (Uboot) |
| 6 | Boot Android Device |
| End |  |

As can be seen in the *Android OS Bootloader* figure and the *Android Bootloader Steps* table, the Android OS bootloader is divided into several components, as per the OS system partitioning table. During the boot process, the following build components are loaded [Acharya et al. 2022]:

1. System img: This image is referred to by system calls to link all the required libraries during runtime. It comprises core libraries and essential application interfaces (APIs) that form part of the default application system of the mobile OS.
2. Vendor img: The Android OS loads a second image known as the vendor img, containing protected binaries specific to certain files. This image also stores binary files that are inaccessible to Android software development teams.
3. Boot img: Under the Linux kernel, the boot img is loaded, containing all the boot files required to mount the necessary partitioning tables.
4. Recovery img: This image is used to load tasks similar to the boot img but is executed in recovery mode. It is a requirement for U-Boot to install components such as packages and software updates.
5. Uboot file.imx: Before the OS starts, this image loads the devices’ bootloader, called U-Boot, which is executed when the mobile device boots.

### Self-Check Questions

* What is the use of mappers in the bootloader?
* *Mappers in the devices assist in dynamic partitioning of the memory, done through a component also found in the kernel.*

## 3.3 Kernel and Hardware Abstraction Layer

**Abstraction layer**

This is a set of software components in the mobile operating system that provide a standard interface between the underlying hardware and the higher-level software that runs on top of it.

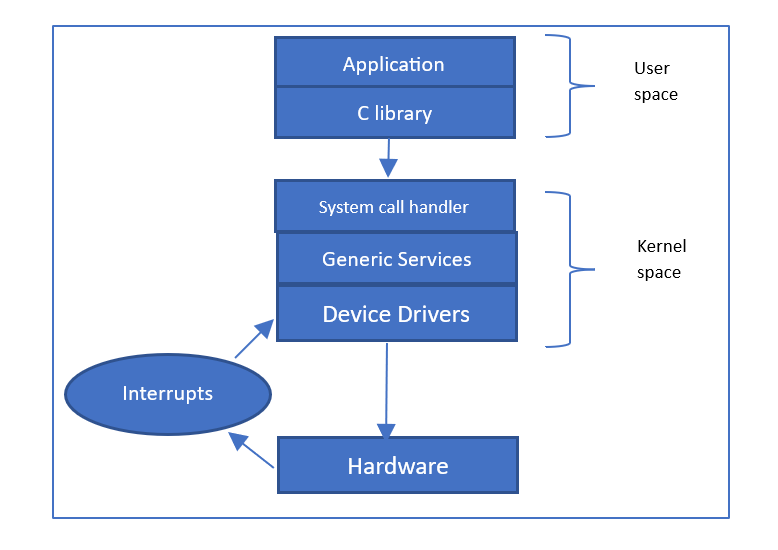
Every mobile device comprises various hardware components that require efficient management, including device drivers necessary for camera, wireless, Bluetooth connectivity, and memory storage. These hardware components are crucial during runtime. When the mobile device boots, the kernel establishes an abstraction layer to manage mobile memory and connected peripheral devices. This abstraction layer consists of software components within the mobile operating system, offering a standardized interface between the underlying hardware and the higher-level software running on top of it.

To achieve this abstraction, specific features are utilized, such as security measures implemented by the Android OS component, managing security-related matters between the OS and mobile apps. The networking hardware stack handles all communication related to networks, while the driver manager is responsible for managing hardware manufacturers’ drivers, ensuring seamless integration of their applications with the inbuilt drivers [Jacobson et al. 2019].

The Linux kernel in mobile devices serves multiple functions including resource management, which involves creating an interface with the mobile device hardware and establishing a hardware abstraction layer (HAL) for users’ applications. Additionally, it performs other functions that are summarized in the figure below.

Diagram

Description automatically generatedCPU Privileges



Source: Panji Harawa [2023], based on Simmonds [2015].

The figure *CPU Privileges* highlights how applications that are running in the user spaces are given low CPU privileges, unlike applications running in the kernel. This is because these applications normally make library calls to access core app functionalities. However, since there is a primary interface between the user and kernel space, the library executes high-level user functionalities called systems calls. It is these system calls that then create an interface for an action that requires the switch between the two modes – that is, from high- to low-level kernel mode [Simmonds 2015].

In the Android operating system, the hardware abstraction layer (HAL) is designed to provide an ecosystem that enables the creation of certain system functionality without requiring any changes to the OS system code. As a result, each mobile device manufacturer may have different implementations of the HAL depending on the specific device and its hardware structure. The typical structure of the HAL is based on modules that contain specific hardware information and data, as well as pointers stored in a shared library that includes primary metadata about the modules, such as the version and designer of the product.

This information stored in these libraries is unique to each hardware, whereas the other structures typically rely on using system calls to identify and communicate with the core hardware components through Java application interfaces [Doan et al. 2020]. This higher-level isolation is also particularly useful as the system calls are being generated by its active processes, and inter-process communication also constitutes part of the security model. The model creates a multiuser system whose objective is to ensure there is separation of resources. However, since during installation each mobile application is given a unique user ID, this means that all resources required by the application will only be used by that app; it can also be used to uninstall the application. Mobile operating systems, like all software-based systems, face various security vulnerabilities due to the increasing complexity of software, evolving attack techniques, and the interconnected nature of devices. These vulnerabilities can arise from programming errors, design flaws, or configuration issues. Some common security concerns include unauthorized access, data leakage, malware infections, and man-in-the-middle attacks. The widespread use of mobile devices and the sensitivity of user data stored on them make addressing these security vulnerabilities crucial. To mitigate these risks, developers and manufacturers need to implement robust security measures, such as regular updates, encryption, and access controls, while users should be mindful of their digital habits and follow best practices for securing their devices and personal information [Ayala 2021].

3.3.1 Usage of HAL: A real-world example

As we described previously, the hardware abstraction layer (HAL) abstracts the low-level hardware details of a hardware component (camera, communication interface, microphone, etc.) by providing a standardized interface for the applications to interact with hardware components. In the case of the Android OS, the HAL is accessed through the Android framework API [Arch 2023]. To illustrate this interaction, we provide the steps for how an application can access the camera in a mobile device through HAL:

* Requesting Camera Functionality: When the camera application is launched, it makes a request to the Android operating system to access camera functionality. The operating system then communicates with the HAL to initialize the camera hardware.
* Initialization of Camera Hardware: The HAL is responsible for initializing the camera hardware, such as the camera sensor, lens, and image processing components. It ensures that the camera hardware is properly initialized and ready to capture media.
* Setting Camera Parameters: The camera application may require specific settings, such as resolution, focus mode, exposure, and white balance, for capturing photos or videos. It communicates these requirements to the HAL, which then configures the camera hardware accordingly.
* Capturing Media: When the user clicks the capture button, the camera application requests the HAL to start capturing media. The HAL interacts with the camera hardware to begin capturing photos or videos based on the specified parameters.
* Closing Camera Session: Once the camera application finishes capturing media, it requests the HAL to close the camera session. The HAL ensures that the camera hardware is released, and any necessary cleanup is performed.

By utilizing the HAL, the camera application can work seamlessly across various Android devices with different hardware configurations. The HAL shields the application from the complexity of interacting directly with the hardware, providing a standardized and simplified interface for accessing camera functionality. This abstraction improves the portability and compatibility of the camera application across a wide range of devices.

### Self-Check Questions

* What role does the HAL play in the Android OS?
* *The HAL connects hardware and software, enabling device-specific implementations.*
* Name two security vulnerabilities in mobile operating systems related to user data.

**Sandbox**

This is a mechanism that provides a controlled environment for applications as it only allows an application to run without having absolute control to write or read information that they are not permitted to.

* *Privacy threats and mismanaged data protection are two user data-related vulnerabilities.*

## 3.4 Sandboxing and Virtualization

The Android operating system employs a sandbox mechanism, providing a controlled environment for applications. This restricted environment ensures that applications run without absolute control over reading or writing data they are not authorized to access. To utilize network services and access hardware at lower levels of the hardware stack, applications must be granted specific permissions. The use of sandboxes has also been instrumental in identifying bugs in mobile applications, as it isolates unverified modules within the device and ensures that only trusted application modules are executed [Yao et al. 2020]. The security mechanisms in sandboxes are similar to those found in virtual boxes.

Virtualization, on the other hand, involves a collection of techniques that create resources and divide them into separate executional environments. When designing a VirtualBox, the environment can be tailored to suit user requirements, effectively hiding hardware details. Virtualization operates through virtual machines, which are isolated software systems independent of hardware, enabling them to run applications as if on a physical computer system. Most virtual machines are configured to provide an isolation layer for hardware requirements and resources, allowing users to scale and work with flexibility. The proximity of the virtual machine’s configuration to the physical hardware affects its scalability and flexibility: configurations closer to the physical hardware make it less scalable, while more distant configurations enhance scalability and flexibility [Vokorokos et al. 2015].

**Virtualization**

This is a collection of techniques and steps that create resources and divide them into several executional environments.

Position of Sandbox Virtualization Layer Within OS

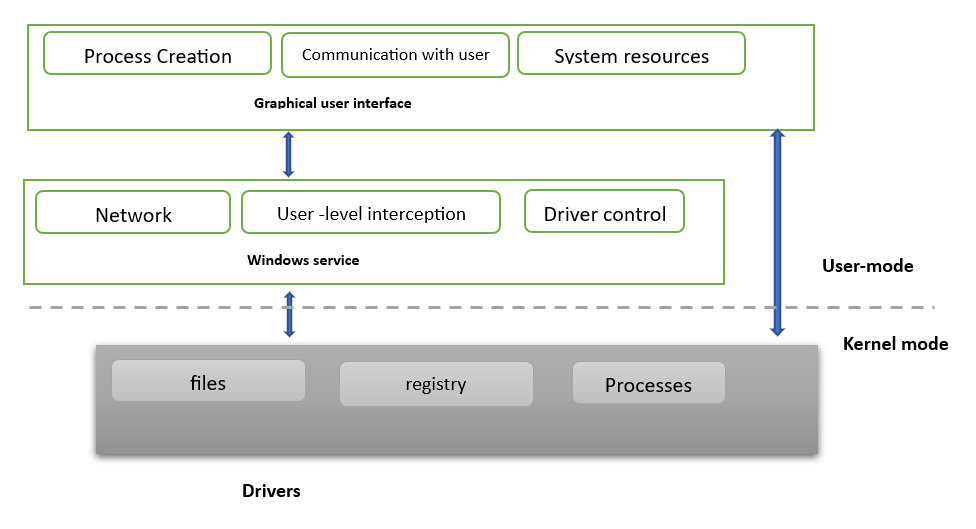
Graphical user interface, text, application

Description automatically generated with medium confidence

|  |  |
| --- | --- |
| Process 1 in Sandbox | Process 2 in Sandbox |
| Virtualization layer | |
| Host Os | |
| Hardware | |

Source: Panji Harawa [2023], based on Vokorokos et al. [2015].

Sandboxing refers to a security mechanism that allows devices to utilize limited resources while separating all executed programs. The sandbox provides an easily accessible platform to run code that has been developed and has not yet been tested. This technique prohibits access to critical mobile device resources, therefore enhancing security. Software developers use sandboxing to roll out new releases of software that is yet to be verified and is untested from all security vulnerabilities, such as viruses and malware, while protecting the host device from being infected. Sandboxes are also used to control mobile applications’ access to critical resources through setting up permissions, as the program can only be allowed to access the device after being granted permission by the device.

Basic Sandbox Architecture

Diagram

Description automatically generated

Source: Panji Harawa [2023], based on Vokorokos et al. [2015].

The basic sandbox architecture depicted in the figure above comprises three distinct components at the architecture level. The first component is responsible for the core execution of the Android operating system kernel, housing the main software modules and drivers. It handles process operations and filters the device registry and files from the user mode. The device’s window services, once loaded, implement functions such as network and driver controls, also performed in the user mode. User interaction with the services is facilitated through a graphical user interface (GUI), which triggers the activation of drivers and service calls, providing feedback to the user via the same GUI. Interactions from the kernel create an abstraction layer when programs wish to access files and processes. The Android OS is designed to safeguard user data by ensuring that mobile applications maintain data integrity. However, the level of protection can vary among original equipment manufacturer (OEM) vendors, as it depends on their adherence to security standards. These protocols subject mobile applications to rigorous malware checks and backdoor prevention embedded within the OS security protocol. As the Android OS is based on the Linux kernel, it inherits the discretionary access control (DAC) security model from it. The DAC is a security model that controls access to resources based on user identities and permissions. When a file is created, the permission is defined as read, write, or execute. The owner of the files can give certain access rights to different users; this can be the owner itself, the group where the resource belongs, or all users.

**Discretionary Access Control (DAC**)

This is a model that secures each mobile application with a unique identification number that it uses to access critical features in the sandbox.

The Android OS extends the DAC model, securing each mobile application with a unique identification number that grants access to critical features within the sandbox. Through sandboxing, untested applications are secured by preventing app core frameworks from interacting or interfering with other running mobile applications. This includes regulating app access to networking modules, Bluetooth, and wireless communication, which is implemented using complex network security protocols accessible only within the sandboxes [Goonasekera et al. 2015].

#### 3.4.1 Apk sandboxing

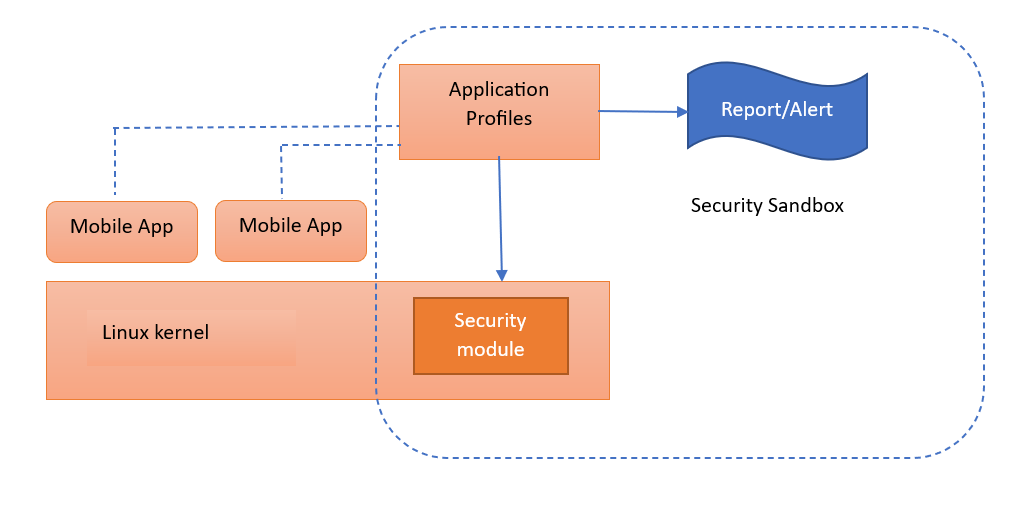
The integrity of mobile data is maintained using application signatures, which also ensures security of the data. It does this by signing each application with a unique ID that is used by the app in the virtual environment. In the figure above, three applications access various hardware components and are grouped depending on the component required by the app. This is done during the installation process. If any tampering has been done that results in invalid certificates, the application installation will be aborted and terminated by the Android OS. [Tejinder S. Randhawa 2022].

Setting up a sandbox involves configuring a secure space with specific rules and safety measures to ensure a safe environment for applications to operate within. This process requires adhering to an architectural design that dictates how applications can access and utilize mobile device resources. Key components of a sandbox can be containers or virtual machines, serving as modules that facilitate this control mechanism. Defined policies govern the data flow within the sandbox, and its operation takes place at the user level [Vokorokos et al. 2015]. These modules also track unverified and untrusted applications, enabling the management of insecure system calls, ensuring that applications can safely function and execute within the same confined space.

Security Sandbox

Diagram

Description automatically generated



Source: Panji Harawa [2023], based on Vokorokos et al. [2015].

The implementation of sandbox modules comes with its challenges, as depicted in the figure *Security Sandbox*. These modules depend on the kernel to efficiently manage and grant permissions required by applications at an advanced level [Spreitzenbarth 2013]. Consequently, a robust standalone security sandbox is established, enabling applications to operate securely while accessing all essential system resources on the device. Prior to running an application, it generates an authenticated profile using the security module in the kernel. If any suspicion of tampering arises, an alert is promptly generated.

### Self-Check Questions

* How does APK sandboxing function?
* *The APK sandboxing mechanism creates a cryptographic hash value of the device’s UID, which is then cross-referenced by the hash value of a verified app certificate.*

**Code signing**

This is the process of using a digital certificate to authenticate and verify that an application has been developed by a designated software designer.

## 3.5 Code Signing

The process of using a digital certificate to authenticate and verify that an application was developed by a specific software designer is referred to as **code signing**. The process uses a cryptographic hashing mechanism to provide proof that there have been no changes made to a piece of software since its release into production. Two main cryptographic hashes are used for this, SHA1 and SHA256. These cryptographic hashes, once invoked, create a combination of a private and a public key pair, which formulate a digital signature that can then be used to verify an application package through its **manifest file**. These signatures are used to check for any tampering or suspected modification that could exist. This is done by confirming the pairs of the public keys and checking if they are the same as the package contents [Horn 2018].

As determined, the Android operating systems must digitally sign each mobile application during installation. This allows the OS to provide a mechanism to verify the authentication of a mobile application as well as checking the security features of the app. This is done to ensure that only approved apps are run, which then creates trust in the app as it is being run. Since most applications will require permission to access some resources, code signing helps in establishing a trusted environment between these applications. As such, as part of dealing with mobile threats, if an application is not digitally signed and authenticated, the Android OS emulators will not accept/allow its installation on the device. The signing of each application is usually done during deployment of an application, and tools such as Key Tool and Jar Signer can be used to generate unique signatures for applications’ APK files. Code signing can enhance the security of mobile applications by enabling the establishment of unique permissions for each application. These permissions can be tailored to either allow or restrict access to specific resources, bolstering the overall security of the applications. The Android security component by default does not grant automated permissions to applications; each application must request permissions that allow use of APIs. For example, the user must grant an application access to the APIs for the camera and GPS, as well as the telephony subsystem to access calls, SMSs, and contacts [Boyer & Mew 2016].

**Manifest file**

In computing, this is a file that includes metadata about a group of accompanying files that are intended to be used together as a unit.

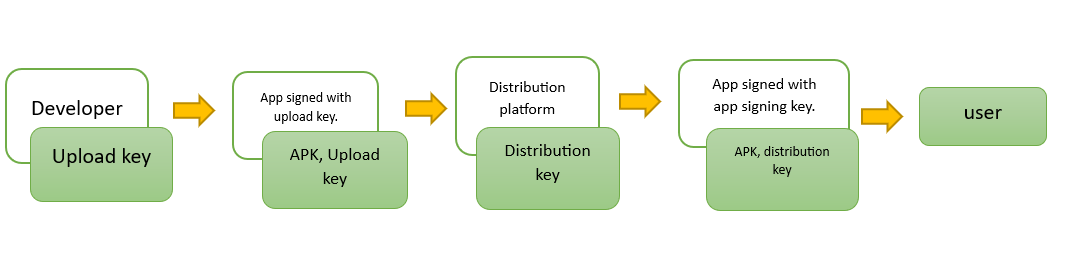
However, like other mobile security mechanisms, employing cryptographic hashes presents certain challenges. Cooper et al. [2018] contend that verifying the authenticity of an application using cryptographic hashes can be problematic, primarily due to their reliance on public keys. It is often difficult for mobile devices to ascertain whether a generated public key is owned by an individual or an organization, which poses a challenge in ensuring the security and integrity of the application. This also creates a challenge in identifying and determining if the public key was modified or completely changed. To resolve this, the Android OS employs a public key infrastructure (PKI). This mechanism uses third parties in the verification and authentication of applications, referred to certificate authorities (CA), which are entities responsible for issuing, managing, and revoking digital certificates in a public key infrastructure (PKI) system. The idea was first initiated through the development of security standards for web applications as a key requirement for the HTTPS internet protocol. The same model is implemented in various mobile software application distribution platforms, such as the Google Play store for Android apps and other Android vendor-based app markets including the Samsung Store. As such, mobile application developers must choose the CA they would like to use for the app. To enhance security and reduce mobile threats, the developer must be registered and vetted by the CA. This, however, can create challenges if the developer of the application and the publisher are different. This is addressed by requesting issuance of a signed certificate from the developers that the CA verifies and grants, allowing their application to be published by the software distributor – e.g., the Google Play store [Kwon et al. 2021].

The Android OS resolves this by implementing two strategies that use a decentralized public key infrastructure. This model allows developers to generate the cryptographic keys themselves as seen below.

Upload Key Resolution Strategy

Diagram

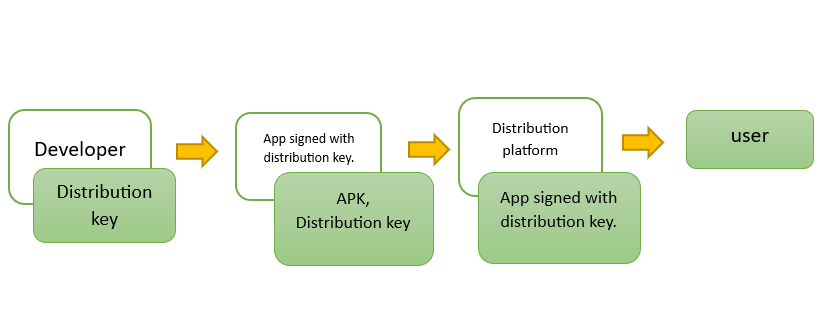
Description automatically generated



Source: Panji Harawa [2023], based on Horn, K. [2018].

In the context of the figure *Upload Key Resolution Strategy*, the software distributor, such as Google for Android mobile apps, generates two pairs of keys. The first pair consists of the distribution key and the upload key. For app authentication purposes, the upload key is utilized. Afterwards, Google proceeds to re-sign the APK package.

Distribution Key Resolution Strategy

Diagram

Description automatically generated

Source: Panji Harawa [2023], based on Horn, K. [2018].

Referring to the figure *Distribution Key Resolution Strategy*, the responsibility for signing of the application lies with the developer, who needs to sign both for the distribution and upload keys.

#### 2.5.1 Permission management in Android

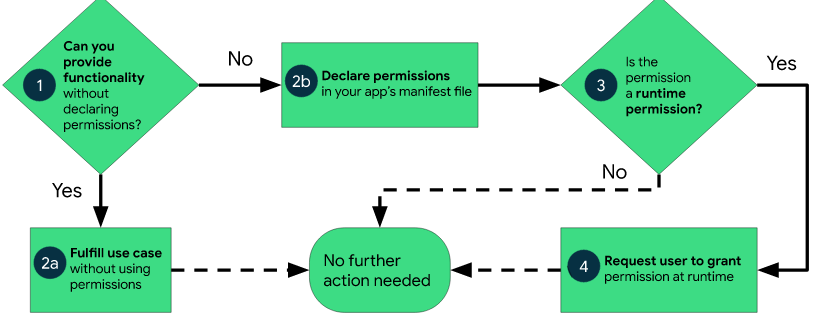
To support user privacy, app permissions are needed to protect access to restricted data and restricted functions. Restricted data relates to private data, such as, for example, the user’s contact list, while restricted functions deal with, for example, connecting a device to another BLE device or giving access to read functions for the user’s directories [PERM 2023].

Android application permissions are categories as follows [PERM 2023]:

* Install-time permissions are permissions that are granted to the app at installation time. The application store presents these permission requirements at the time when installation is performed by the user, who has to accept them to continue with installation.
* Run-time permissions are permissions that are presented during the runtime of the application. Such runtime permissions can affect the system as well as the work of other applications; thus, each permission is requested every time before accessing restricted data or restricted functions.
* Special permissions relate to particular apps and can be defined only by the platform and the OEMs.

The high-level architecture of Android permission management is shown in the figure *Permission Management in Android*.

Permission Management in Android



### Self-Check Questions

* What is the purpose of code signing in the Android OS, and how does it enhance mobile application security?

*The purpose of code signing in the Android operating system is to verify the authenticity and integrity of mobile applications.*

Summary

This Unit gives background on how the Android operating system is based on Linux and was developed by the Open Handset Alliance led by Google. Being open source, it allows for easy software development and modification, making it easy for companies to develop unique functionalities. Android has hardware components that enable easy running of apps, provide high-level encryption, and manage mobile devices’ hardware components. The unit critically discusses the diversity of mobile applications in the Android ecosystem and the how the underlying technology stack is dependent on common application libraries, facilitating the development of both native and hybrid mobile applications that are cross-platform. The Android architecture has several components broken down as managers for different services, with each component used for a specific function. However, the access requests made by native and third-party mobile applications present high security threats, as most users do not pay critical attention to them. OEMs use bootloaders to customize their own operating systems and ensure that these customizations also consider all the peripherals controlled and managed by the software on the chip. At the hardware level, we see that the Android kernel creates an abstraction layer that manages all the connected peripheral devices and the mobile device memory. The hardware abstraction layer (HAL) provides an ecosystem that enables the creation of certain system functionality without requiring any changes to the OS system code, and each mobile device manufacturer may have different implementations of the HAL depending on the specific device and its hardware structure.

The unit summarizes how the Android OS uses sandboxing to provide a controlled environment for applications to run in, with limited access to resources. The unit further explains how virtualization is used to create isolation and flexibility for users. The sandboxing mechanism allows for untested code to be rolled out while protecting the device from security vulnerabilities. The sandbox architecture consists of three components: core execution of the kernel, modules for process operations and filtering, and user interface for device interaction. The sandbox uses modules such as containers or virtual machines to gain access to device limitations and ensure secure execution. When it comes to mobile application security, this is maintained using the APK sandboxing mechanism, which uses cryptographic hash values and verified app certificates to ensure data integrity. The implementation of modules in the kernel can create an effective standalone security sandbox but can also present challenges. The same principle is applied when code signing – a process that uses a digital certificate to authenticate and verify that an application has been developed by a software designer. The unit details how code signing uses cryptographic hashes to provide proof that there have been no changes to a piece of software since its release into production. The Android OS uses code signing to verify the authentication and security features of an app during installation. Code signing enhances mobile application security by setting up unique permissions for each application that can allow or restrict access to some resources. Briefly, the unit explains how the Android OS employs a public key infrastructure (PKI) that uses certificate authorities (CA) to verify and authenticate applications. In addition, the Android OS implements two strategies that use a decentralized public key infrastructure to generate cryptographic keys for developers.

# Unit 4 – Apple iOS Operating System

**Study Goals**

On completion of this unit, you will be able to …

… determine the iOS hardware architecture.

… analyze how the bootloader of iOS functions.

… identify key sandboxing and virtualization processes of the iOS operating system.

… understand how the mobile device kernel provides security in iOS.

… explain how code signing of applications enhances security in iOS mobile devices.

## Introduction

MacOS has been at the core of Apple computer products since the late 1990s. It was during this era that the Macintosh platform was born, after which came a mobile operating system (OS) that was developed called iOS. iOS is a mobile operating system developed by Apple Inc. It is the operating system that powers many of Apple’s mobile devices, including iPhone and iPad. iOS is known for its user-friendly interface and sleek design, as well as its many features and capabilities, such as access to the App Store and the ability to connect to iCloud. Most Apple mobile devices use iOS as the core software that creates an ecosystem to manage and control software resources, hardware, and all peripherals. It provides support for mobile applications that are used to act and create interfaces that users and software developers can use interact with hardware. Just like all other types of operating system, it has evolved over the years, with an ecosystem of mobile apps that are used on a daily basis and implementation of standard OS services, such as storage, process scheduling, and management of boot processes. The easy implementation of its application programming interfaces (APIs) on low-level personal computers made it easy for software engineers to write and execute programs in a convenient way. In today’s modern technological trends, Apple platforms have grown and gained more popularity through their iPhone technology [Statistica 2022], which has escalated with the emergence of cloud computing, the use of web applications, and mobile applications.

## 4.1 Hardware

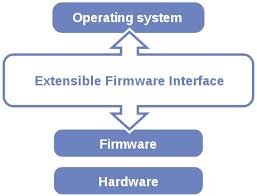
As a mobile operating system, iOS is designed to run on a wide range of hardware devices, including smartphones, tablets, and other mobile devices. These devices typically have several hardware components, such as a processor, memory, storage, and various sensors, necessary for the device to function and perform the tasks it is designed for. The specific hardware components of a given Apple device will vary depending on the model and the manufacturer of the device’s internal components. However, at the core of the Apples iOS there is critical infrastructure hardware. For many years it has been a widespread practice for system engineers to have a standard that can be used for all common OS versions across different hardware. However, it has been noted that most mobile devices by Apple have several components that are integrated; hence, for software engineers to use macOS, they also must consider recent changes to hardware that typically occur with the latest version of the operating system. This is part of how Apple deals with mobile threats; hence, there is no backward compatibility for most of their older versions of iOS or macOS [Drew Smith 2020].

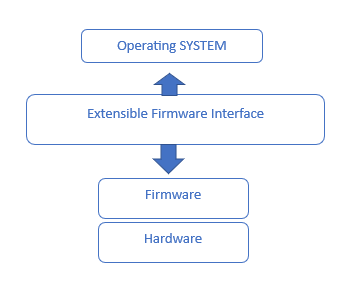
**Extensible firmware interface (EFI)**

a specification that defines a software interface between an operating system (OS) and platform firmware.

As part of the hardware components, every Apple device features an **extensible firmware interface** (EFI) which is a specification that defines a software interface between the operating system (OS) and the platform firmware. The EFI is one of the core hardware components of the iOS security boot and uses a pre-boot password feature. This feature works at a low level, like the firmware, to operationalize the password utility, also called the startup security utility. This iOS password utility feature’s main function is to allow system administrators to lock down the booting of the mobile device without the need to enter a password. The setting up of a password on EFI usually does not require showing a password prompt on startup when the system is used normally. This feature is only activated when the user tries to access the device in recovery mode, using a recovery medium booting process, or via network boot. These features provide security and prevent mobile threats on iOS devices by allowing locking of the devices so that they can only boot from a legitimate, trusted operating system. Afterward, the pre-boot password feature is employed to verify the integrity of the bootloader and the operating system, preventing any unauthorized code execution or tampering.

**UEFI Secure boot IOS**





Source: Panji Harawa [2023], based on <https://www.linux.com/training-tutorials/uefi-secure-boot-big-hassle-questionable-benefit/>

As can be seen in the figure *UEFI secure boot in iOS,* the EFI encompasses two categories of service: boot-time services and runtime services. Boot services are accessible while the EFI firmware governs the system by loading the OS while making sure that runtime services remain available during firmware control and after the operating system assumes control. After this, several extensions are loaded from hardware, which may include closed-source drivers, support for legacy operating systems, and various other proprietary enhancements.

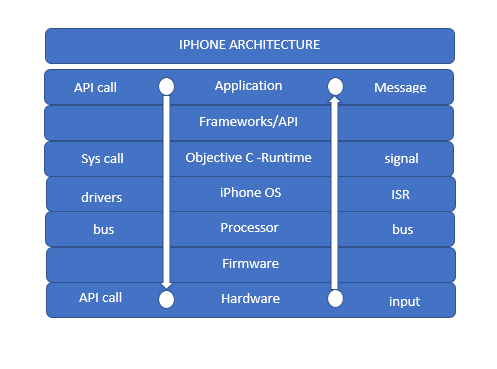
#### 4.1.1 iPhone architecture

The hardware components of an iOS device have broad applicability to several Apple devices, including Apple TV and iPads, since we are in technological era where devices need to connect to each other. The iOS architecture has the main OS layer at the bottom of the software stack; after this layer there is a pre-occupational layer, followed by a media, Cocoa Touch layer, whose main function is to provide the main framework for app development. In addition, there is the core services layer of the software stack at the bottom whose function is to provide high level networking, security, and multimedia. As an operating system, iOS relies heavily on schedulers, a framework that offers a programming model used for task scheduling, state management, and the creation of asynchronous workflows and integrated file systems. These schedulers work in conjunction with the Mach kernel, a vital component responsible for interfacing with memory management systems. The primary roles of the Mach kernel include providing essential low-level services, especially in process scheduling and handling drivers. It is also responsible for configuration and control of hardware drivers for all peripheral devices. In addition, it is responsible for the iOS telephony subsystems that control all inter-process communication with the device [Gronli et al. 2014].

iPhone Architecture

Graphical user interface

Description automatically generated with medium confidence



Source: Panji Harawa [2023], based on Tiwari [2017].

The iPhone architecture is structured in multiple layers, as shown in the figure *iPhone Architecture*. Starting from the bottom layer, the hardware layer refers to physical chips that are embedded within the mobile device circuitry. Processors and system on a chip (SoC) are integrated on this chip to manage all computing and scheduling of all instruction sets, which are done in the memory descriptor but executed at processor level. The next layer is firmware, which has unique code that has been developed to work with the device’s memory and is stored there as well on this chip. After storing and execution, the processor issues a specific ARM instruction set that formulates the booting of the iOS device and drivers. This is a direct linkage between the two layers’ processor and iOS. As the processor is executing it normally sets up an interrupts descriptor table (IDT), a data structure used by the x86 architecture to implement an interrupt vector table. The IDT is used by the processor to determine the correct response to interrupts and exceptions that are managed by the iPhone OS. For the iPhone OS to fully function, it needs applications that are developed by software developers who prefer working with Objective C as a programming tool to allow easy scaling up and support by other libraries. The iPhone mobile application technology stack also uses several frameworks/APIs that are mostly accessible through the Apple software development toolkit (SDK), which is responsible for incorporating Apple-distributed headers, such as App kit and UI kit, as they already provide access to Apples APIs. It is essential to emphasize that all mobile applications in Apple products are compiled at the hardware level using native code and a distributed compiler feature of Xcode. This feature enables developers to distribute the compilation of their code across multiple platforms. This allows apps to run within a predefine user environment [Khan et al. 2022].

iOS Architecture

Diagram

Description automatically generated

Source: Panji Harawa [2023], Based on Atkar [2020] <https://www.fosstechnix.com/ios-architecture-and-concepts/>

Title iOS Architecture

Graphical user interface

Description automatically generated

Source: Panji Harawa [2023].

The iPhone hardware stack as introduced above has five core components. The top layer, referred to as the Cocoa Touch layer, contains all the software frameworks and libraries that are used in mobile applications. Software engineers develop apps that reference the Cocoa API, within the iOS framework that is used to access libraries. The second layer, media services for iOS devices, is responsible for the management of audio, video, and graphics capabilities. It also includes various frameworks used to develop applications that require using audio, video, or graphics. The third layer is the core services layer in iOS, which serves as a foundation for all software components to integrate and function. This layer provides essential services for the other layers to build upon, including features like the address book and core data frameworks. Lastly, the core OS is for managing the user interface and supporting touch-based interactions. All these components are fully embedded in the core iPhone hardware, which also helps in the management of these native applications [Pršeš 2020].

### Self-Check Questions

* 1. What are the key components of Apple hardware, and how do they enable the functionality of an iPhone?
* *Processor: The processor, or central processing unit (CPU), is the “brain” of the device, responsible for executing instructions and performing calculations*.
* *Memory, or random-access memory (RAM), is used to store data that the device is currently using or processing.*
* *Storage: Storage is where data are stored on a device.*

## 4.2 Bootloader

The bootloader is software that loads the operating system on a computer or mobile device. On Apple iOS devices, the bootloader is called “iOS Boot Loader” or “iBoot.” This low-level program verifies the integrity of the iOS operating system and checks for proper sign-in before loading the OS into memory for execution. The iBoot boot loader is an important part of the iOS operating system as it ensures the security and integrity of the system by only allowing signed and trusted operating system software to run on the device.

**Bootloader**

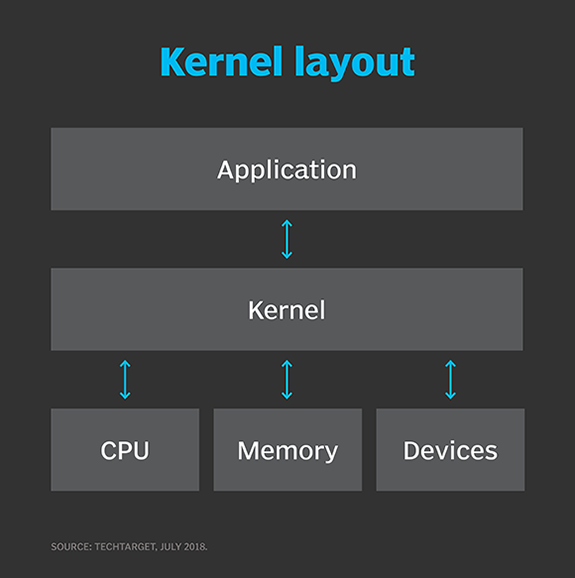
This is software that loads the operating system on a computer or mobile device.

When an iOS device is turned on, the iBoot boot loader is the first software to run. It checks the integrity of the operating system by computing a cryptographic hash and comparing it to the expected value stored in the device’s ROM. If the hashes match, iBoot verifies that the operating system has been signed by a trusted source, such as Apple. This prevents any mobile threats on the device. After the operating system has been verified, iBoot loads it into memory and transfers control to it. The operating system then initializes the device’s hardware and starts the user interface. The top layer of the iOS architecture, referred to as cocoa touch layer (see above), is a layer that contains all the software frameworks and libraries that are used in mobile applications. Software engineers develop apps that reference the Cocoa API within the iOS architecture, which is used for building user interfaces. Apple provides specific frameworks to help with app development in iOS as follows:

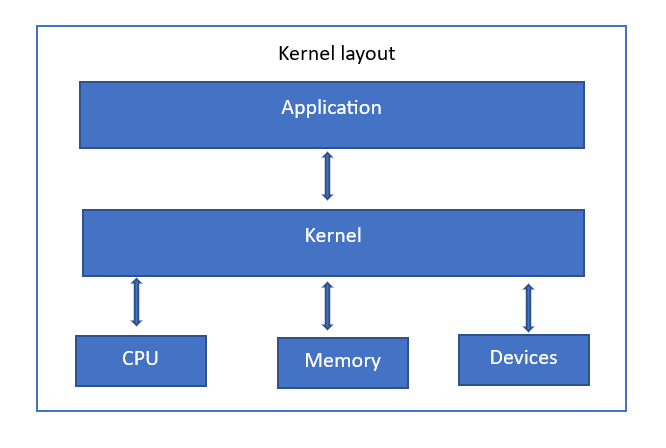
* Core Graphics framework: provides support for 2D graphics, i.e., drawing shapes, text rendering
* Core Data framework: provides support for managing structured data
* Core Location framework: provides support for accessing the device’s location and heading information, i.e., GPS
* Core Bluetooth framework: provides support for communicating with Bluetooth devices

Together, the iOS kernel and its frameworks provide the foundation upon which the rest of the operating system and its applications are built and are essential for the proper functioning of the device [Garcia et al. 2016].

User Mode, Kernel Mode, and Hardware



Source: <https://www.techtarget.com/searchdatacenter/definition/kernel>



Source: Panji Harawa [2023], based on <https://www.techtarget.com/searchdatacenter/definition/kernel>

Kernel designs vary to cater to different system requirements. Monolithic kernels operate entirely within a single address space and prioritize speed, functioning in supervisor mode. In contrast, microkernels allocate most services to the user and application areas, focusing on modularity and resilience. The microkernel itself offers only fundamental functionality, while separate applications and devices assume previous kernel roles such as system drivers [Bigelow et al. 2022]. Mobile devices, including smartphones and tablets, require a mobile operating system to enable various communication interfaces and services like voice calls, messaging, and internet browsing. In the context of iOS, Apple’s proprietary mobile operating system, the kernel plays a crucial role in managing system resources and serving as an intermediary between applications and hardware devices [Bigelow et al. 2022].

### Self-Check Questions

Please complete the following sentence:

* Discuss the three key responsibilities of the kernel in managing system resources.

*The kernel of an operating system is responsible for managing system*  *resources, such as process and thread management, memory management,*  *device management, file system management, and I/O management.*

## 4.4 Sandboxing and Virtualization

iOS sandboxing is a security feature of the iOS operating system that restricts applications’ access to the device’s resources and data. This means that each app is isolated from other apps and is only allowed to access the resources and data that it has been specifically granted access to. Whereas when we talk about **virtualization**, we’re referring to a technology that allows multiple instances of an operating system to run on a single device. This allows iOS to run multiple instances of the operating system, each with its own set of applications and resources, on a single device. Together, these technologies help to improve the security and stability of the iOS operating system. The sandbox and virtual machine do this by limiting app access to the device’s resources and data; security in iOS is strengthened by employing these techniques, which are a combination of hardware and software features. These measures enable the operating system to run multiple instances of itself on a single device, effectively isolating individual apps from each other. As a result, one app is prevented from interfering with or accessing the data of another app, enhancing overall security and ensuring a safer user experience.

**Virtualization**

This is a technology that allows multiple instances of an operating system to run on a single device.

At the software level, iOS uses a security model called “application signing” to ensure that only trusted apps can run on the device, which in return adds an extra layer of security during virtualization. This is also particularly useful when using containerized applications running in a sandbox and on virtual machines. This model requires developers to sign their apps with a unique certificate issued by Apple, which will be verified by the operating system before the app can run. Furthermore, iOS incorporates a technology known as containers, which operate as an abstraction at the app layer. These containers bundle the source code and dependencies of an application together, effectively implementing virtualization. Each app is assigned its own container, which then creates a virtualized environment that provides the app with its own private set of resources and data [Talal et al. 2019].

Overall, sandboxing and virtualization depend on software and hardware features such as isolation and resource sharing that help to improve the security and stability of the operating system, making iOS one of the most secure platforms when it comes to mobile threats. iOS uses sandboxing and virtualization to improve the security and stability of the iOS operating system from attacks. Sandboxing achieves this by preventing apps from being able to interfere with or access the data of other apps, while virtualization allows the operating system to run multiple instances of itself on a single device, which helps to prevent one instance from crashing and affecting other instances. The figure below shows the iOS security structure.

iOS Security Structure

Diagram

Description automatically generated

Diagram

Description automatically generated

Source: Android v/s iOS – The Unceasing Battle; Aditya Sahan [2017].

For Based on graphics:

Source [Author], [2023] based on Aditya Sahan [2017].

**Root certificates**

Also called trusted root certificates, these are used to establish a chain of trust that’s used to verify other certificates signed by the trusted roots.

As seen in the figure above, iOS has two key components – hardware and software – that make up iOS’s core security features. The secure enclave as part of the iOS kernel helps provide security at low level. iOS also employs use of cryptographic hashes using group keys and **root certificates**, which are stored at the hardware level. The software creates an abstraction through the iOS App sandbox, hence ensuring that there is no external tampering with apps. One of the main security mechanisms is that iOS does not make its source code available to the public, making iOS less vulnerable to being “jailbroken,” or hacked to gain access to unauthorized features. iOS also includes various hardware and software features that enhance security by default, such as data encryption and the use of touch ID and face ID. These features not only improve security, but also enhance the user experience. In addition, iOS devices use encryption to protect data stored on the device and transmitted over the network. Data is encrypted using a unique key that is generated when the device is first set up, and the key is stored in the secure enclave that accesses the required key through the iOS crypto engine at the hardware level.

### Self-Check Questions

* What is iOS virtualization and how does it enable the execution of multiple operating systems on a single device?
* *iOS virtualization is the process of running multiple operating systems on an iOS device through virtual machines. It enables the execution of multiple OS’s by abstracting hardware resources and providing isolated environments for each OS.*

## 4.5 Code Signing

Operating system code signing is a security measure that ensures only trusted apps can run on the device. It is managed by the operating system and involves developers signing their app with a unique certificate issued by Apple. This certificate acts as a digital signature that verifies the developer’s identity and the app’s integrity. When an app is installed on an iOS device, the operating system will verify the signature to ensure it is valid and has not been tampered with. If the signature is valid, the app will be allowed to run. If the signature is invalid or has been tampered with, the app will not be allowed to run, and the user will be notified. In addition, iOS periodically checks the signatures of installed apps to ensure they are still valid. To further improve security, iOS devices use hardware-enforced memory protection to prevent apps from accessing unauthorized memory, and the operating system uses a security model called “application signing” to verify the authenticity and integrity of apps, achieved by signing the application with a unique developer certificate issued by Apple. Overall, code signing helps to prevent malicious or tampered-with apps from being installed and executed on the device, which is achieved by signing the compiled code or binary of the app with the developer’s certificate, hence improving the security of the operating system. To be able to achieve and execute this, the following steps shown in the figure below are necessary in iOS devices [Apple, *Application Code Signing* 2018].

Code Signing

(original)

Diagram

Description automatically generated

Diagram

Description automatically generated

Source: Apple, *Application Code Signing* [2018].

The signing process is as follows:

* Develop the app: The first step in the code signing process is to develop the app. This involves writing the code, testing it, and making any necessary updates or changes.
* Obtain a developer account: To sign an app for iOS, you must have a developer account with Apple. You can sign up for a developer account through the Apple Developer Program website.
* Create a signing certificate: Next, you will need to create a signing certificate to sign your app. This is done through the Apple Developer website or using the Xcode development tool.
* Sign the app: Once you have a signing certificate, you can use it to sign your app. This is typically done using the Xcode development tool, which will generate a signed version of your app that is ready to be submitted to the App Store.
* Submit the app to the App Store: The last step in the code signing process is to submit your signed app to the App Store. This is done through the Apple Developer website, and it involves providing information about your app, such as its name, description, and screenshots.

Concisely, the process of iOS code signing involves developing the app, obtaining a developer account, creating a signing certificate, signing the app, and submitting it to the App Store. This helps ensure that only trusted, signed apps can run on iOS devices, improving the operating system security [Apple, *Application Code Signing* 2018].

### Self-Check Questions

* What are the two key components of the iOS code signing process?
* *Development Certificate: A development certificate is a digital certificate that is issued by Apple to a developer*.
* *Signing Key: A signing key is a unique digital key that is used to sign software.*

Summary

This unit describes Apple’s mobile operating system, iOS, which has been the core of its mobile products since the late 1990s, powering devices like iPhones and iPads. It is known for its user-friendly interface and sleek design, and has evolved over the years, with an ecosystem of mobile apps used in various industries. The OS has gained popularity through iPhone technology, enhanced by cloud computing, web applications, and mobile applications. The hardware components of iOS devices mostly include a processor, memory, storage, and sensors, just like other mobile devices. Apple features an extensible firmware interface (EFI) as a core hardware component for iOS boot security. Its architecture at the core consists of five main layers: Cocoa Touch, media, core services, pre-occupational, and main OS layers, and has evolved over the years, applying also to Apple TV and iPads. The Mach kernel interfaces with memory management systems and hardware drivers, and controls the iOS telephony subsystems. iOS also requires a bootloader, which is referred to as “iBoot” and which is responsible for loading the iOS operating system on a device. It verifies the integrity of the OS by computing a cryptographic hash and comparing it to the expected value stored in the device’s ROM. In addition, the secure boot chain ensures that the lowest levels of software on the device have not been tampered with. Apple devices have a dedicated processor called the secure enclave, which performs a secure boot process to verify the software’s signature. Next the unit highlights how the iOS kernel, a core component of the operating system, is responsible for managing resources and providing low-level services to the operating system and apps. iOS also includes frameworks that offer specialized functionality, such as Core Foundation, Core Graphics, Core Data, Core Location, and Core Bluetooth. These components form the foundation for the operating system and its applications. To achieve all these, iOS employs sandboxing and virtualization to improve security and stability. Sandboxing isolates apps from each other, limiting their access to resources and data, while virtualization allows multiple instances of the operating system to run on a single device. These features are implemented through a combination of hardware and software, including application signing, containers, and the secure enclave, which stores cryptographic keys at the hardware level. This comprehensive approach to security makes iOS one of the most secure mobile platforms. The unit also details how iOS handles and manages code signing, a security measure employed by the iOS operating system to ensure that only trusted apps run on devices. Mobile app developers sign their apps using a unique certificate issued by Apple, which verifies the developer’s identity and the app’s integrity. The operating system checks the app’s signature upon installation and periodically afterward to maintain security. To achieve this, developers must follow a process that includes developing the app, obtaining a developer account, creating a signing certificate, signing the app, and submitting it to the App Store. This process enhances the security of the iOS operating system by preventing malicious or tampered-with apps from being installed and executed on devices.

# Unit 5 – Mobile Devices

**Study Goals**

In this unit, you will …

...learn about Linux distributions and their use for mobile devices.

... learn about the internet of things (IoT) and how mobile devices can be integrated into IoT systems.

... understand the architectures of real-time operating systems (RTOS) and how they can be used on mobile devices.

... understand how embedded systems function, including issues related to security, performance, and power management.

## Introduction

Mobile devices, specifically smartphones, have become a crucial aspect of our society due to the diverse range of functions and capabilities they possess. These handheld electronic devices are incredibly convenient and useful for both personal and professional use. With a smartphone, individuals can make phone calls, send texts and emails, access the internet, and utilize various apps to stay organized, entertained, and connected to their surroundings. Moreover, smartphones are equipped with cameras and other features that allow users to capture and share memories, record video, and much more. No matter whether a smartphone is being used for work or leisure, it offers numerous possibilities for staying connected and completing tasks while on the go. However, when it comes to cyber threats, mobile devices are not spared. As such, this section tackles how the use of these devices also carries with it certain risks, such as the potential for malware, hacking, and phishing attacks, which can compromise the security and privacy of the device and its users. It is therefore important for individuals and organizations to be aware of these risks and to take measures to protect against them. This may include regularly updating security patches, implementing strong password protocols, and exercising caution when downloading apps or accessing websites. By acknowledging the potential for mobile threats and taking appropriate precautions, it is possible to maintain the security and integrity of mobile devices as platforms for communication and productivity

## 5.1 The Internet of things

**The Internet of Things**

This is a network of electronic devices and objects that can communicate and exchange data with each other and the internet, enabling remote control and monitoring.

The **internet of things** (IoT) is a network of electronic devices and objects that can communicate and exchange data with each other and the internet, enabling remote control and monitoring. These devices, such as smart thermostats, security cameras, and wearable fitness trackers, use sensors, actuators, and communication capabilities to interact with their surroundings and one another. The goal of the IoT is to create a network of connected devices that can be easily accessed and controlled, automating many everyday tasks and processes. The potential uses of the IoT are vast, with the possibility to transform industries such as healthcare, transportation, agriculture, and energy management. By gathering and analyzing data from connected devices, businesses and individuals can make more informed decisions and optimize their operations. For example, a smart home system can use IoT technology to adjust the temperature, lighting, and security based on the occupants’ preferences and behavior. In healthcare, IoT devices can monitor patients’ vital signs and alert medical personnel of any abnormalities. The IoT has the potential to enhance our daily lives and increase the efficiency of various industries. [Anand et al. 2021].

The structure of the IoT in mobile devices is typically made up of several layers, each of which serves a specific function in the system. These layers may include [Jenky 2019]:

Sensors and Actuators

Graphical user interface, application

Description automatically generated

A picture containing graphical user interface

Description automatically generated

Source: Panji Harawa [2023], based on Janky et al. [2019].

* Sensors and actuators layer: These components of IoT collect data from the physical world and allow the system to interact with it. In mobile devices, these may include sensors like accelerometers, gyroscopes, and GPS, and actuators like vibration motors and speakers.
* Device layer: This refers to the hardware and software components that make up the mobile device itself, such as the processor, memory, storage, and operating system.
* Networking layer: This part is responsible for enabling communication between the device and the internet, and between the device and other connected devices. This may include WiFi, cellular network connection, Bluetooth, near-field communications (NFC), and other communication technologies.
* Platform layer: This refers to the software and services that enable the device to connect to and interact with the IoT. This may include cloud services, middleware, and application programming interfaces (APIs) that allow the device to send and receive data to/from the internet and other connected devices. In addition, data can be processed and formatted based on the needs of the data sinks.
* Application layer: This is the layer that users interact with, consisting of apps and other user-facing features of the device. These may include apps for tracking fitness, managing smart home devices, or accessing location-based services.

Furthermore, the architecture of the IoT in mobile devices involves the integration of various hardware, software, and communication technologies to enable the device to collect and exchange data with the internet and other connected devices, such as smart security home systems and smart appliances Due to this multifaceted integration, IoT faces an immense number of mobile threats, such as malware. We now discuss how it manages and prevents mobile threats [Rathod et al. 2022] .

#### 5.1.1 IoT Security Requirements

IoT device security requirements can be grouped into three main levels [Meneghello 2019]: the information level, access level, and functional level. At the information level, security mechanisms should ensure integrity, anonymity, confidentiality, and privacy of information passing through or stored in the IoT device. This is the general security aspect for every internet device.

At the access level, security mechanisms should specify certain features to control access of the IoT device to the network. These include [Meneghello 2019]

* Access control mechanisms that guarantee access only for legitimate users to the IoT device and network administration functions.
* An IoT device authentication mechanism to check whether the IoT device is allowed to access the network. This is the first step before establishing communication between the device and the network.
* An authorization mechanism to ensure that the IoT device can access only the services in the network it is permitted to.

At a functional level, security mechanisms should ensure IoT device functionality throughout its operation. This includes [Meneghello 2019]:

* Resilience to ensure that the IoT devices stay operational even under security attacks. This is the hardest function to achieve in an IoT network.
* A self-organization mechanism to ensure that the IoT network remains operational based on automatic re-adjustment after a security attack in the network.

#### 5.1.2 IoT Security Vulnerabilities

An IoT system can be separated into edge, access/middleware, and application parts [Pielli 2015]. While the edge part implements the physical and medium access control (MAC) layer to provide local communication between IoT devices and gateways, the middleware layer acts as the intermediary between the IoT world and the internet. An example of a middleware layer is 6LoWPAN, which provides a mapping between the IPv6 layer on the internet side and the low-level layers of LoWPAN devices. Similarly, in the case of Bluetooth low energy (BLE), we have IPv6-over-BLE [RFC7668], which makes use of 6LoWPAN over BLE by considering the BLE frame formats and features. An application layer implements the IoT device functionality and offers service-level communication with the network as well as with other IoT devices.

Due to IoT device constraints, several security vulnerabilities exist that we will distinguish here based on the layer at which they occur.

At the physical and MAC layers, the main threats for IoT devices are side-channel attacks and hardware trojans. Side-channel attacks utilize analysis of side information, such as communication timing, electromagnetic radiation, and power consumption, to leak information from encrypted packets. For example, the power consumption of the IoT device can be exploited to recover and guess encryption keys based on statistical analysis of large numbers of traces.

IoT devices are vulnerable to hardware trojan attacks as well. Such attacks can make the IoT device unavailable for the user, achieving a denial of service (DoS) attack by, for example, battery power drain or jamming of communication channels for other devices without the hardware trojan.

At the middleware layer, there are several vulnerabilities that can be exploited by attackers, such as address resolution protocol (ARP) spoofing (making nodes think that the next hop is some other node rather than the legitimate one), DHCP server spoofing (a rouge DHCP server that replies to the IoT device requests and reroutes the traffic), packet dropping, and fraudulent packet injection.

Application layer vulnerabilities relate to the software stack running in the IoT device and directly expose the functionality of the IoT device to risk. Application layer threats can be divided into four main categories [Ronen 2016]: (1) ignoring IoT device functionalities, (2) reducing IoT device functionality, (3) misusing IoT device functionalities, and (4) extending IoT device functionalities for malicious purposes.

#### 5.1.3 IoT Security mechanisms

Encryption is one of the main security mechanisms to provide information security in IoT devices. Encryption is a process of changing clear information into an obscured form using a function that can be inverted using a known shared secret. In IoT systems, the communication is usually broadcasted or multicasted in downlink. In this type of mass communication, a single gateway or access point will serve multiple end devices. In this scenario, symmetric key encryption is not possible; hence, public key encryption is preferred. Due to limitations and power constraints, the decryption takes time and resources at the end node. As such, some approaches allow for the message to be sent in plaintext accompanied by an encrypted tag that saves the authentication of the sender. Then each device can immediately access the plaintext while only one selected device per group of devices will authenticate with the tag attached to the message [Stallings 2017]. If not correct, it will inform the other devices. To further improve the encryption process for low-end IoT devices, other lightweight encryption mechanisms can be specified, such as PRESENT, CLEIFA, and PRINCE [Fremantle 2017].

After encryption, being able to generate random numbers is an important feature for security protocols in order to, for example, generate nonces, avoid reply-to attacks, or generate asymmetric keys [Meneghello 2019]. Random number generators exploit physical characteristics of the device. The two main types of random number generators are true random number generators (TRNGs), which exploit the randomness of the white noise from the communication channel or other noise sources in the device, and pseudo random number generators (PRNGs), which create a long random sequence of bits from a short key using a deterministic algorithm. Again, due to power limitations, TRNG is not feasible for implementation in IoT devices, but lightweight PRNG algorithms are used instead.

IoT devices are often deployed in remote areas where they can physically come in contact with possible attackers who can carry out side channel attacks. One way to prevent this is by adopting physically unclonable functions (PUFs) to improve hardware security [Meneghello 2019]. PUFs exploit small variabilities in the chip manufacturing process to generate a unique signature for each device. However, this approach increases power usage of the IoT device and is only feasible for devices where power consumption is not a problem.

### Self-Check Questions

How do mobile devices, such as smartphones and tablets, support the connectivity and communication of IoT devices?

* *Mobile devices can be used to access and control IoT devices remotely, either through dedicated apps or through web-based interfaces.*

## 5.2 Linux

“***Linux is a Unix-based free and open-source operating system”*** [Top 10 Best Linux l 2019]. First released in 1991, it has since become a popular choice for servers and high-performance systems, known for its stability, security, and flexibility. Linux can be customized to fit the specific needs of an organization or individual and can run on various hardware, including smartphones, tablets, and supercomputers. Its open-source nature allows users to access and modify its source code, fostering collaboration and innovation within the community. Linux has strong security features, making it less vulnerable to malware and other threats compared to other operating systems.

Linux plays a significant role in the mobile market, particularly in smartphones, where it can serve as an operating system, support apps, provide terminal emulation, and function as the basis for Android. As the kernel used in most mobile device operating systems, Linux is lightweight, customizable, and secure, making it a popular choice for mobile devices handling sensitive information.

While Android is built on a modified version of the Linux kernel, iOS is based on Darwin, an open-source Unix-like operating system that shares some similarities with Linux. Both iOS and Android benefit from the robustness and security features of Unix-based systems.

In the mobile market, Linux has become an important player and is expected to continue growing in popularity in the future [Amit et al. 2019].

Mobile Device Hybrid Kernel

Diagram

Description automatically generatedTimeline

Description automatically generated

Source: Panji Harawa [2023], based on Krasov et al. [2020].

As can be seen in the figure *Mobile Device Hybrid Kernel*, the hybrid kernel combines features of both microkernel and monolithic kernel architectures to interact with file servers, device drivers, hardware, and applications in a balanced and efficient manner. In a hybrid kernel, some essential services like device drivers and file systems run in kernel space to reduce performance overhead, while others, such as user interface components and application services, run in user space for better fault isolation and modularity. The microkernel provides essential services such as inter-process communication (IPC), whereas the monolithic kernel provides high performance and efficiency for the devices.

In addition, to enhance performance, a protected structure of the Linux kernel is implemented in the same hybrid kernel. This ensures that mobile device operating systems maintain a high level of security and stability. Key security features like process isolation, access controls, and address space layout randomization help safeguard mobile devices against unauthorized access, malware, and other threats. Additionally, kernel hardening techniques and secure boot mechanisms further enhance the security posture of mobile operating systems built on Linux [Kerrisk 2018].

The kernel provides a tool useful in preventing mobile threats, which is accessible through a command line interface (CLI). On a mobile device running Linux, you can access the CLI using a terminal emulator app or by connecting to the device over a network using SSH. Common tasks that can be performed using the CLI on a mobile device include managing files and directories, installing and updating software packages, and customizing the system settings.

Mobile devices running Linux can be used to develop and deploy a wide range of applications, including native apps that run on the device itself and web-based apps that run in a browser. To develop apps for a mobile device running Linux, you need to choose a programming language and development environment and may need to install additional libraries or frameworks [Leonard et al. 2019]. Once you have developed the app, you can deploy it to the device by installing it from a package manager or by building and installing it from source. One of the benefits of using Linux on a mobile device is the ability to customize the user interface and system settings to meet your specific needs and preferences. This can include changing the desktop environment, installing custom themes and icons, and modifying system-wide settings such as network configuration and power management options. To customize the user interface and system settings on a mobile device running Linux, you can use the CLI and edit configuration files or use a GUI-based tool like a desktop settings app [Takahiro et al. 2022].

### Self-Check Questions

* Name one of the key components of Linux and how it enables the functionality of a mobile device.
* ***One key component of Linux that enables the functionality of a mobile device is the kernel, as it interacts directly with the hardware and manages system resources.***

## 5.3 RTOS

Real-time operating systems (RTOSs) are specialized operating systems designed to handle time-sensitive tasks and provide predictable and reliable performance for real-time applications. These systems are commonly used in industries such as aerospace, automotive, and industrial automation and are found in systems requiring precise, timely control, such as robotics, medical devices, and control systems. RTOSs prioritize and quickly execute time-critical tasks using fast interrupt handling and minimal overhead to ensure accurate, timely completion. They also offer deterministic performance, meaning they can guarantee a certain level of performance under specific conditions, enabling predictable and reliable performance in the face of external events or disruptions. This therefore makes them ideal for use in IoT applications and embedded systems. RTOSs are an essential component of many real-time systems, and their importance is expected to continue growing in the future. Some of the key security measures that can be implemented for RTOSs that apply to other types of operating system include

• Regular updates and patches: An RTOS should be regularly updated and patched to fix any known vulnerabilities and ensure the latest security features are in place.

• Security policies and procedures: Organizations should develop and implement security policies and procedures to ensure that the RTOS is used securely and in accordance with best practices.

• Access controls: RTOSs are often restricted to authorized users only, and access controls should be in place to prevent unauthorized access.

• Encryption: An RTOS should ensure that all data is encrypted to protect it from unauthorized access. Overall, an RTOS can be secure if proper security measures are implemented and best practices are followed [Furrer 2022].

Due to their varied use cases and applications, there are several types of RTOS, some of which are explained below:

* Preemptive RTOS: These systems allow high-priority tasks to interrupt lower-priority tasks, ensuring that time-critical tasks are completed in a timely manner. The main advantages of this are that it guarantees that critical tasks are executed within time limits and offers high performance, whereas a major drawback is that they are very expensive as they require more hardware.
* Non-preemptive RTOS: These systems do not allow high-priority tasks to interrupt lower-priority tasks. Instead, they use time slicing to allow each task to run for a set amount of time before allowing another task to run. These have several advantages, one of them being that they are simple to design since they do not track all tasks. However, consequently they can also be very unpredictable since it is not guaranteed that all tasks will be run.
* Hybrid RTOS: These systems combine the elements of both preemptive and non-preemptive RTOSs. This allows them to scale easily since they can handle more tasks.

Real Time Operating System

Diagram

Description automatically generated

Diagram

Description automatically generated

Source: Panji Harawa [2023], based <https://www.javatpoint.com/hard-and-soft-real-time-operating-system>

As can be seen from the figure *Real Time Operating System*, an RTOS is just like other OSs, consisting of a kernel and system services that manage resources, tasks, and memory. It also includes device drivers to interact with external hardware. The RTOS connects the system operator, the real-time system, and the object being controlled, processing inputs in real time to ensure timely and predictable performance. This allows the system to meet strict deadlines and efficiently manage time-sensitive tasks.

RTOSs are used in a variety of applications, including

* Embedded systems: these are small, specialized computer systems that are integrated into other devices.
* Industrial automation: used to control and monitor production processes.
* Aerospace and defense: used to control and monitor aircraft and military equipment.
* Medical devices: used in devices such as defibrillators and ventilators to provide precise, timely control.

RTOSs are also used in mobile devices, such as portable gaming consoles and smartwatches. In these devices, RTOSs are used to manage and prioritize tasks, ensuring that the device can respond quickly and accurately to user input. They may also be used to manage the device’s power consumption, allowing the device to run longer on a single charge.

In summary, RTOSs are essential components of many real-time systems, and their importance is expected to continue growing in the future [Hermann et al. 2022].

### Self-Check Questions

* Name one of the key components of Linux and how they enable the functionality of a mobile device.

***The kernel is one of the central parts of the Linux operating system; it manages the system’s resources and communicates with the hardware****.*

## 5.4 Android on Devices

By volume, Android is the most used mobile operating system; it is based on the Linux kernel and is developed by Google [Statista 2022]. It is used on a wide range of devices, including smartphones, tablets, and other mobile devices. Android has gained popularity due to its open-source nature, user-friendly interface, and seamless integration with Google services such as Gmail and Google Maps. It also has a rich ecosystem of applications and services and is supported by a wide range of hardware manufacturers. Considering security, Android is one of the most secure mobile operating systems. It has built-in security features, like sandboxing, which isolates applications from each other and the operating system, and encryption, which protects sensitive data. Google also releases regular security updates to fix known vulnerabilities and provide the latest security features. These updates are automatically pushed to devices, and it is important that they are installed promptly to ensure the security of the operating system. However, there are still some security risks to consider with Android, such as the possibility of installing malicious applications from untrusted sources or not keeping devices up to date with the latest security patches. To maintain the security of Android devices, it is recommended to only install applications from trusted sources and to keep the operating system up to date with the latest security patches. In summary, Android manages security through a combination of built-in security features, regular security updates, and user-implemented security measures [Collins & Ellis 2015].

Android Platform Architecture

Graphical user interface, chart, treemap chart

Description automatically generated

Source: Panji Harawa [2023], based on <https://developer.android.com/guide/platform> (Creative Commons Attribution license.)

The figure *Android Platform Architecture* shows the main building blocks of the Android Platform. Looking top down, the Android platform is composed of system apps, the Java API framework, native C/C++ libraries, and the Android runtime environment.

System apps include the core apps for email, web-browsing, calendar, dialing, and more, which come bundled together with the Android device. The Java API framework is composed of modular systems and includes the following [GUIDE 2023]:

* Content providers that enable apps to share data between each other.
* The view system, which enables developers to build apps’ user interfaces.
* Various managers, including the activity manager (manages the lifecycle of an app), resource manager (manages resource access of the apps), and notification manager (manages notifications of apps in the display).

Below this, Android provides native libraries that are built in C or C++. Some examples include the surface manager (which handles graphics), the media framework (which handles audio and video), and a SQLite database (which is used for storing data). Finally, at the core we have the Android runtime, which consists of two main components: the Dalvik virtual machine and the core libraries. The Dalvik VM is a custom virtual machine that is designed specifically for Android, and it is responsible for running Android applications. The core libraries provide the classes used by those applications during development and compilation. An additional layer known as the Java API framework provides mobile devices with access to a set of APIs that provide access to the various features and services of the Android operating system. It includes classes for things like networking, databases, and user interface elements. These APIs are mostly used for content providers, the view system, are accessed and used by developers to create apps that run on Android devices. The Android OS has a large free ecosystem of mobile apps that can be downloaded for free on Google Play. Most of these apps run on Android devices, such as the home screen, the phone app, and the web browser, which uses many libraries and APIs. These apps are built using the APIs provided by the Java application framework, and they provide the core functionality of the device. Android enjoys its leading market share because of its open structure that has made the architecture of the Android operating system to be highly modular and flexible, so that it can be used on a wide range of devices with different hardware configurations [Shaheen et al 2017].

### Self-Check Questions

* List at least three common features of Android devices that make them popular among users.
* *Customization: Android devices offer a high degree of customization.*
* *Wide range of devices: Android devices come in a wide range of sizes, prices, and hardware configurations.*
* *Open ecosystem: Android is an open platform, which means that developers can create and distribute apps.*

## 5.5 Other Common Embedded Operating Systems

Embedded systems are specialized computer systems that are designed and integrated into other devices or systems. They often require a specialized operating system to run, called an embedded operating system. These systems are characterized by their small size and limited resources and are designed to be efficient and reliable. Embedded operating systems are used in many industries, including aerospace, automotive, industrial automation, and healthcare. They can be found in small devices such as microcontrollers, sensors, and medical devices. Embedded systems rely heavily on hardware resources, and both Android and iOS can be categorized as embedded systems with high-performing underlying hardware. However, on low hardware-resource systems, some common examples of embedded operating systems include the following:

• VxWorks, a real-time operating system (RTOS) that is widely used in a variety of industries, including aerospace, automotive, and industrial automation. It is known for its fast interrupt handling, minimal overhead, and deterministic performance, which makes it well-suited for real-time applications.

• ThreadX, another type of RTOS that is designed for high-performance and low-power devices. It is known for its fast interrupt handling and minimal footprint, which makes it a popular choice for embedded systems.

• MicroC/OS-II, mostly known for its simplicity and small footprint. It is often used in small embedded systems, such as microcontrollers and sensors.

• QNX, best known for its reliability and real-time capabilities. It is commonly used in mission-critical systems, such as medical devices and industrial control systems. There are several distinct types of embedded operating systems, including real-time operating systems (RTOS), lightweight operating systems, and specialized operating systems.

• Lightweight operating systems are designed to be small and efficient, with minimal overhead and a small footprint; hence, they are often used in small embedded systems, such as microcontrollers and sensors. These types of systems are essential to enable embedded systems to function properly [Jabeen Q et al. 2016].

Taxonomy of Embedded RTOS

Diagram

Description automatically generated

Diagram

Description automatically generated

Source: Panji Harawa [2023], based on Jabeen et al. [2016].

Embedded systems are specialized computing devices designed to perform dedicated tasks within larger systems. Hence, their operating systems are built on similar architectural principles. These systems often have limited resources, such as processing power, memory, and energy, which require a tailored architecture to efficiently manage resources and provide reliable performance. The core architecture of an embedded OS typically consists of a kernel, task management system, interrupt handling system, memory management system, device drivers, and various services and libraries. The kernel is the central component responsible for managing the system’s resources, such as memory, processors, and devices. It also provides scheduling and synchronization services to ensure tasks are executed quickly and predictably.

Task management is crucial in embedded systems, as it helps allocate the necessary resources and CPU time for each task within the system. Efficient task management ensures that the system can maintain its performance even with limited resources. Interrupt handling is essential in embedded systems because it enables the system to respond to external events or device interrupts promptly and efficiently, ensuring seamless operation and minimal latency. Memory management in embedded systems is designed to optimize resource allocation and deallocation, providing sufficient memory for tasks to execute in devices with limited resources [Jabeen et al. 2016].

Embedded systems include device drivers that grant access to the system’s hardware and devices. In addition, various services and libraries support application development and execution; these include networking, file systems, and communication libraries. Lastly***,*** just like other RTOS, embedded real time systems mostly specialize in computing devices whose focus is on efficient resource management, reliable performance, and seamless operation, even with limited resources. Their architecture is designed to address the unique requirements of performing dedicated tasks within larger systems.

### Self-Check Questions

* What are some common embedded operating systems?
* VxWorks an embedded (RTOS) developed and designed for use in embedded systems that require fast and reliable performance.
* QNX is a widely used OS in embedded systems used for real-time performance, small footprint, and ability to handle multiple requests.

Summary

This unit primarily focuses on the vital role of mobile devices today, especially smartphones, and their significant impact on the internet of things (IoT). It sheds light on the various risks and challenges associated with mobile devices, including malware, hacking, and unauthorized access, while stressing the importance of implementing security measures such as regular updates, robust passwords, and exercising caution when downloading apps or visiting websites. In the context of IoT, mobile devices serve as essential components, as they form the backbone of a network of interconnected devices spanning various layers, encompassing sensors and actuators and the device layer, communication layer, platform layer, and application layer. The unit highlights the necessity to address security challenges faced by IoT devices and adopt protective measures, including encryption, secure boot, access controls, and network security. At the heart of the unit, we find that most IoT devices rely on Linux, an open-source operating system known for its stability, security, and flexibility. The unit delves into the intricate process of configuring Linux on mobile devices and the crucial role played by the command line interface (CLI) in managing these systems. As the development of operating systems progresses, there is growing interest among engineers in the use and development of real-time operating systems (RTOSs). The unit explores RTOSs in-depth, outlining their application across industries such as aerospace, automotive, and industrial automation. Moreover, it discusses distinct types of RTOS (preemptive, non-preemptive, and hybrid) and their usage in various fields, including embedded systems, industrial automation, aerospace and defense, and medical devices.

Based on data from multiple sources, Android emerges as the most popular mobile operating system, owing to its open-source nature, user-friendly interface, and seamless integration with Google services. It is highly secure, offering built-in features like sandboxing, encryption, and regular security updates. The unit provides a comprehensive explanation of Android’s architecture, which comprises the Linux kernel, libraries, Android runtime, and the application framework. Mobile phone users employing Android-based devices are urged to install apps from trusted sources and keep their devices up to date for optimal security. This is because such applications tend to be more secure, reliable, and less vulnerable to mobile threats. Lastly, on the topic of operating systems, the unit investigates the usage of embedded operating systems in specialized computer systems integrated into other devices or systems. Although there are numerous embedded OS options, some common examples include VxWorks, ThreadX, MicroC/OS-II, and QNX. The unit further delineates how an embedded RTOS possess a core architecture that consists of a kernel, a task management system, an interrupt handling system, a memory management system, device drivers, and numerous services and libraries. This architecture is strikingly similar to the core hardware architecture found in mobile devices running either iOS or Android operating systems.

# Unit 6 – Software Ecosystems and Security

Study Goals

On completion of this unit, you will be able to …

… Understand the distinct types of software ecosystems and how they function.

… Learn about the various security threats that can arise in a software ecosystem.

… understand how Google handles security in their ecosystem.

… understand the role of the cloud in mobile security ecosystems.

## Introduction

The mobile software ecosystem refers to cooperation and interplay between mobile users, mobile apps and their developers, mobile device manufacturers, and app markets. There are billions of mobile device users, millions of apps and app developers, thousands of mobile device manufacturers, and several mobile app markets and platforms – with two of them being the most well-known and widely used: Google Play Store and Apple Store. In this unit we will discuss each of the application distribution platforms of Google and Apple and how they interact with both app developers and users. We will show the application development phases from the perspective of ecosystem interaction and how app monetization can be achieved. A part of the revenues goes to the market store distribution platform; each company (in this case Google and Apple) has their own models of contract conditions for app distribution.

Application distribution platforms do not only simply distribute the apps, however; they also play a crucial role in app security by ensuring protection from malware, spyware, and other security threats. Each of the platforms have their security scanning policies even before the app becomes available in the respective app stores. Also, in addition to distribution and security, app stores offer different monetization policies for the app developers to benefit from their work. Of course, part of the profit will go to the maintainer of the app store (in this case Google or Apple). In this unit, we will discuss all these aspects for both main app ecosystems: Google Play and Apple Store.

## 6.1 Google play

Google Play is the official app store for Android OS-based devices; it serves as a digital distribution service that is developed and maintained by Google. It provides applications, games, books, movies, and other digital content for users. It is the primary marketplace for Android apps. Google Play was released in 2012 by merging and centralizing the Android Market (specialized in Android app distribution), Google Music, Google Movies, and Google eBook stores in one distribution platform. The number of available Android applications in the Google Play Store has reached 2.7 million [STA 2023], making Google Play the most widely used app store in the market. Though Android devices allow for other app stores to be used, more than 90% of the apps downloaded in an Android device come from Google Play Store [DC 2023].

Android was introduced in 2007 as a mobile operating system offering an open-source model of software where everyone was invited to contribute to the development. In the beginning this faced a range of resistance from the device manufacturers, which were used to producing devices working with closed-source operating systems such as, for example, Microsoft [Mob 2008]. In the beginning only HTC agreed to produce Android-based devices [Mob 2008]. Despite the resistance, Google remained committed to the principles of open platforms, emphasizing the benefits of collaboration, innovation, and a thriving developer community. The company continued to refine and enhance Android, making it more robust, feature-rich, and user-friendly with each iteration.

Over time Android OS and Google Play (its official app store) came to control more than 70% of the app store market [MARKT 2021].

#### 6.1.1 Application lifecycle in Google Play Store

An application lifecycle refers to various steps and stages that an application development process undergoes until the application is available to be used by the user. Typically, an application lifecycle involves the following steps:

* Application development and testing: This includes the application design and coding using the Android development platform, Android Studio, and using one of the programing languages Kotlin [Kotlin 2023], Java, or C++.
* Application submission: Using the Google Play console, an application developer can submit their app to the Google Play Store once they have created an account. The process includes submitting the application details (app language, title, screenshots, etc.), generation and upload of a signed APK of the application file, and content rating for the targeted audience.
* Application review: This is done by the Google Play side and includes several processes – policy compliance check, application functionality review, and security review. The whole process can take from a few hours to several days, depending on the complexity of the app.
* Application publishing: Once the application has passed all the reviews by Google Play, it will be listed in the app store.

#### 6.1.2 Components of Google Play Ecosystem

The Google Play ecosystem is a dynamic and multifaceted platform that encompasses several key components, each playing a crucial role in shaping the mobile app experience for Android users. These components work together to provide a comprehensive environment for app discovery, installation, security, and overall user satisfaction. The main components of the Google Play ecosystem are [Android 2023]: the Google Play store, Google Play services, and Google Play protect.

As said previously, the Google Play Store serves as the central hub for discovering, downloading, and updating applications and other digital content for Android OS-based devices. It offers an extensive catalog of apps, games, movies, books, music, and more.

The Play Store employs advanced algorithms to facilitate app discovery. Users can search for apps by keywords, categories, or trending topics, enhancing the visibility of both established and emerging applications. Apps are classified into 32 different categories, such as productivity, entertainment, education, and tools [Wang 2019]. In addition to categorization, the Google Play Store features ranking systems that showcase top apps based on factors like number of downloads, ratings, and user engagement. In addition, apps can be categorized as paid apps and free to use apps. In 2019, the percentage of free-to-use apps in the Google Play store amounted to 99.7% of all apps [Wang 2019]. Users can provide reviews and ratings for apps, enabling others to make informed decisions. Positive reviews and high ratings contribute to an app’s credibility and visibility within the store.

Google Play Services is a set of APIs and background services that provide essential functionality to Android apps. They operate behind the scenes, offering developers access to powerful features while maintaining consistency across different Android devices. Google Play Services provide optimization of on-device resources, automatic updates of apps, and backward compatibility with Android 5.0 APIs [GP Service 2023]. Google Play Services offer APIs for tasks such as location services, Google Maps integration, authentication, cloud messaging (Firebase Cloud Messaging), and more. These APIs simplify app development by providing pre-built solutions for common functionalities. For developers, Google Play Services streamline app development by offering a unified platform to access various services. On the other hand, for users, they ensure a consistent experience across devices and facilitate the integration of Google features into third-party apps.

Google Play Protect is a security solution designed to safeguard Android devices from potentially harmful apps and activities. It combines machine learning, app scanning, and real-time threat detection to provide a secure environment for users [GP Protect 2023]. Google Play Protect provides protection at two stages: on-device level and on-cloud level security. On-device security capabilities help to keep the device and data secure. Google Play Protect scans apps on user devices daily for malware, spyware, and other security threats. If a potentially harmful app (PHA) is found in the device, the user will be asked with a notification to remove the app. If the app is not useful or has not been used for a long time by the user, it can be removed automatically and blocked from future installations. The cloud-based security scans the app before it is published on the Google Play Store. Google Play Protect conducts thorough checks to ensure apps’ safety and compliance with policies. This verification process enhances the overall security of the app ecosystem.

#### 2.1.3 Monetization

Every app developer will want to generate revenue out of their intellectual property. Google Play provides developers with means of generating revenue from their apps and content while offering end users a range of pricing models and options. In this section we will describe various monetization strategies and models available within the Google Play ecosystem. The four main monetization strategies are: paid apps, in-app purchase, subscription, and advertising [GP monet 2023].

In-app purchases are a popular monetization model that allows developers to offer additional content, features, services, or digital or physical goods within their apps. Users can make purchases directly from the app, enhancing user engagement and revenue potential. In-app purchases are effective for apps, where the basic version is free, and users can opt to purchase more content or to remove ads. Google Play provides a robust infrastructure for handling in-app purchases securely, ensuring a seamless user experience and reliable revenue stream for developers.

Subscriptions provide a recurring revenue stream for developers and offer users access to premium content or services for a specified period [GP Sub 2023]. This model is suitable for apps that offer ongoing value, such as video streaming apps, news apps, or fitness apps. Google Play supports various subscription types, including monthly, annual, and introductory pricing. Developers can customize subscription plans and offer trial periods or discounts to attract subscribers. The subscription model encourages long-term engagement and establishes a stable income source for the developers, if the number of subscribers is maintained.

As in any other content sharing platform, advertising is another monetization strategy for apps in Google Play. Developers can aim to display ads within their apps to get revenues from the advertising companies. Google Play offers integration with Google AdMob, a platform that enables developers to earn revenue by displaying relevant ads to users. Ad-based monetization is often paired with free apps and is effective when implemented thoughtfully to balance user engagement and ad display.

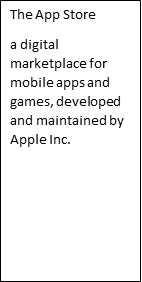
Google Play employs a revenue-sharing model, where developers typically receive 70% of the app’s revenue, while Google retains the remaining 30%. However, certain circumstances, such as subscription revenue and geographical considerations, may lead to adjusted revenue shares.

### Self-Check Questions

* Name the main monetization models in the Google Play Store.
* Paid apps, in-app purchase, subscription, and advertising.

## 6.2 Apple Store

The App Store is the official app store maintained by Apple for its iOS-based products. It serves as a digital distribution service for applications running on iOS for different Apple products: iPhone, iPad, iPod, Apple smartwatch, or Apple TV. The origins of the App Store can be traced back to a pivotal moment in technology history – the unveiling of the iPhone in 2007. Initially Apple was planning to limit the third-party development of apps for iOS. But due to concurrency, Apple agreed to open the app development to third-parties; however, they centralized the distribution of apps for iOS and provided a software development kit (SDK) for development. This laid the foundation for a groundbreaking innovation: a centralized platform where users could seamlessly discover, download, and enjoy a diverse array of applications. A year after introducing the first iPhone, on July 10, 2008, Apple officially launched the App Store, forever altering the digital landscape.

The App Store’s launch marked an unprecedented leap forward, as it democratized software distribution. Prior to its existence, acquiring and installing software was often cumbersome, requiring manual installation from various sources. The App Store introduced a seamless and user-friendly mechanism, enabling users to access a wealth of applications directly from their devices.

At its inception, the App Store offered a selection of just 500 apps – an impressive number at the time. However, this modest number in beginning reached around 1.6 million in 2022 [STA 2023]. Within the first year, the App Store boasted over 65,000 apps, showcasing the power of a platform that empowered developers to highlight their creativity and ingenuity to a global audience.

#### 6.2.1 Application life cycle in The App Store

For an app to be available in the App Store, certain steps need to be followed before it can be downloaded, installed, and used by the user. These steps can be summarized as follows:

* Application development and testing: This includes application design, coding, and testing to ensure it functions as intended and is free of major bugs. Apple provides tools like Xcode and the iOS Simulator [App Sim] to assist in the development and testing process.
* Application submission: Before submission of the app to the App Store, the developer should take care to test for crashes and bugs, enable back-end services if needed, check whether the app follows Apple guidance, and include detailed explanation of non-obvious features of the app [App Sub]. The submission process includes submitting the application details (app language, title, and screenshots), and the application itself using Apple’s App Store Connect portal for the review.
* Application review process: Apple’s App Review team evaluates the app to ensure it adheres to the App Store Review Guidelines. This process includes checking for compliance with safety, security, design, functionality, and content guidelines. The review process time depends on workload, and there are no time frames for the review process to finish [App Sub].
* Application publishing: Once the app is approved during the review process, the app will become available within 24 h in the App Store. However, if the app release date is set in the future, this is delayed until the release date.

There are cases where an app is rejected by the App Store. If the developer thinks that the rejection is unjustified, they can appeal the decision by submitting to the Apple Store Connect platform. They might choose to include a new version of the app with changes/updates in the appeal process.

Once the app is available in the App Store, it can be downloaded and used by users. During the lifetime of an app, new bugs might appear that have to be fixed, or the developer wants to update certain features of the app. The bug fix review process is faster and does not require the whole review procedure to be repeated [App Sub].

#### 6.2.2 App Store privacy

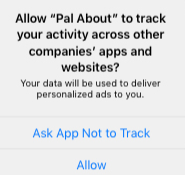
During interaction between users and apps that are downloaded from the App Store, users might come across a situation where they must share their personal data, which will be managed by the app. To protect user privacy and manage user data usage, the App Store provides two mechanisms: description of data usage and permission tracks.

Each developer is asked to make a detailed description of the data that the app tracks and processes, and if such data are related to the identity of the user or the identity of the device [APP Priv]. In addition to this, developers should report if third-party code and SDKs are used in the app and if they collect any data from users for third parties. Based on this, a privacy nutrition label (PNL) is assigned to the app, so the user can determine the privacy level of their data when using the app. Privacy nutrition labels can be categorized based on the data types that apps collect, as follows [App PNL]:

* Contact information PNL: this category includes data like names, email addresses, phone numbers, and other personal contact details.
* Health and fitness PNL: data related to health and fitness, such as exercise routines, medical information, and activity tracking.
* Financial information PNL: information related to financial transactions, payments, credit card details, and banking.
* Location PNL: data about the user’s physical location, often used for services like navigation, mapping, and location-based features.
* User content PNL: any content that the user creates, uploads, or interacts with, such as photos, videos, messages, and other user-generated content.
* Browsing history PNL: information about the websites, pages, and content the user interacts with while using the app.
* Identifiers PNL: unique identifiers associated with the user or device, such as device IDs, advertising IDs, and user account information.
* Usage data PNL: information about how the user interacts with the app, such as app usage patterns, session durations, and interaction behaviors.
* Diagnostic data PNL: technical data that helps diagnose app performance issues and identify bugs.

The second mechanism to ensure privacy is the privacy permission track. Since iOS 14.5, the App Store provides the app tracking transparency (ATT) framework, which requires developers to get explicit permission from users if the app wants to track their data or to access their device advertisement identifier. Users will be prompted with a message to accept the tracking of data or not, as shown in the figure *ATT prompting message.* The user can choose to allow or deny the app’s request. By denying it, users can prevent apps from building a comprehensive profile of their interests and behaviors across different apps and websites, thereby enhancing their online privacy.

ATT prompting message



Source: <https://developer.apple.com/app-store/user-privacy-and-data-use/>

#### 6.2.3 App monetization in The App Store

Developers can generate revenues from their apps based on several business models and monetization techniques that the App Store offers. However, they have to make clear during the review process which is their chosen monetization technique. In the following subsections we will describe each of the payment and monetization methods that developers can use in the App Store.

#### In-App Purchase

In-app purchases can be used to allow app developers to unlock new features and functionalities within the app. Also, in-app purchases can be used to allow users to tip the app developer. In addition, in-app purchase can be used only to sell digital products (digital gift cards, certificates, vouchers, and coupons) and services, rather than physical products [App Sub].

#### Subscriptions

Subscription are another monetization model that is used for apps to offer, for example, new game levels, episodic content, consistent updates, and access to large collections of media content, software as a service (“SaaS”), or cloud support. Subscriptions should be at least seven days long, and the user should be able to upgrade or downgrade their subscription seamlessly [App Sub]. The Apple Store asks developers to make clear the actual subscription information, e.g., how many issues per month will the user get, how much cloud storage will the user get, what kind of access to the service will be provided and how many times per month.

#### Other purchase methods

In some specific cases, other purchase methods than in-app purchases can be used. Some such cases include the following [App sub]:

* “Reader” apps: where content providers (e.g., magazines, newspapers) might ask the user for payment via an external link to their web-based platform.
* Person-to-person services: where apps allow purchases of real-time services.
* Multiplatform services: apps that run on multiple platforms can allow users to access content and subscriptions that they have paid for on other platforms.
* Goods and services outside the app: apps can allow other payment methods, such as credit card payment or Apple-Pay payment, for purchasing goods and services.

#### Apple Pay and cryptocurrencies

Another method of monetization is to offer Apple Pay within an app. In such cases, the developer and owner of the app should provide all purchase information to the user prior to the sale [App Sub]. And the last payment method is cryptocurrency, where apps can support wallets, mining, and exchanges [App Sub].

The App Store uses a 70/30 revenue share model, where 70% of the revenues go to the developer of the app, and 30% go to Apple. In the case of a subscription model, this share will be 85/15 if the subscriber is subscribed for more than a year.

### Self-Check Questions

* How are privacy nutrition labels (PNLs) categorized in the App Store?
* Based on the types of data that are being collected by the app.

## 6.3 Security Providers

In the continuously evolving digital market, app security and user privacy have become important factors. Two of the most used app distribution platforms, the Google Play Store and Apple’s App Store, are committed to safeguarding users’ data and providing a secure environment for app downloads. They provide a number of mechanisms to enhance the security of their users and their data. Such mechanisms play a pivotal role in maintaining the integrity, authenticity, and safety of the apps available on these platforms as well the as security of their users’ data. From pre-screening apps for potential vulnerabilities during reviewing process to actively monitoring and preventing malicious behaviors of different apps, provided security mechanisms within these platforms ensure that users can confidently engage with the apps they download.

In this section, we will look in more detail at mobile application attack vectors and the security mechanisms that each of the platforms provides to overcome such issues.

#### 6.3.1 Security Mechanisms in Google Play Store

To prevent different attack vectors from malicious apps, the Google Play Store has implemented and follows several mechanisms and policies. First, the Google Play Store has a strict review process for all apps that will be included in the store before they appear to the public for downloading. Google Play provides several security mechanisms to protect mobile devices from application malware as well as to protect the integrity of data that applications interact with.

#### Jetpack security

The Android Jetpack suite provides a set of libraries and tools to simplify the implementation of security best practices in Android apps. It leverages the Android KeyStore to create a hardware-backed AES 256 key that is used to protect encryption keys [Android ES 2021]. It also provides the Encrypted File API to support secure storage and the Encrypted Shared Preferences API that ensures encryption of data saved in shared preferences [Muhammad 2023].

#### Application Signing

To ensure the authenticity of an app, verify the source of the app, and enable secure communication between the app and services, Google Play applies application signing. Application signing is a process that involves generating a digital signature for an Android application package (APK) file [Android ES 2021]. Apps signed with the same key can run in the same process and can expose functionality to each other. With updates, to prevent a security breach with a malware application that is not genuine, the corresponding certificates of the new version and the old version of the app are compared. The update is allowed only if the certificates match.

#### SafetyNet

SafetyNet is another set of service APIs offered by Google Play to protect apps against security threats. It can be used to mitigate device tempering, malicious URLs, potentially harmful applications, non-genuine APIs, and fake users. It provides several APIs [Android ES 2021]:

* SafetyNet Device Attestation API: allows the integrity of a device’s hardware and software to be assessed, looking for integrity issues by comparing them with reference data for approved Android devices.
* SafetyNet App Verification API: is used to verify the integrity of installed apps by comparing their cryptographic signatures against known values published on the Google Play Store. If the app has been tampered with or repackaged, it can be detected at this stage.
* SafetyNet reCAPTCHA API: provides CAPTCHA challenges to detect and prevent automated bots or abusive behavior.

#### Network Security

In addition to the security of data at rest, Android also provides secure communication to avoid any man-in-the-middle attack possibilities. Android supports domain name system (DNS) over transport layer security (TLS). Using this, administrators can configure the DNS over TLS as well as prevent users from changing such configurations [Android ES 2021]. Also, by default Android app developers are required to use TLS 1.3 to avoid clear text traffic over the network. The TLS protocol is standardized by IETF under RFC-4346 [RFC4346] for its first version, and the latest version TLS 1.3 is standardized per RFC-8446 [RFC8446]. The TLS is a sublayer located between the application and transport layer. It is used to provide security and privacy for several applications, such as HTTP, FTP, and SMTP. The TLS protocol is based on a three-way handshaking procedure between the device and the server. This procedure is as follows:

* exchanging supported security parameters
* authenticating peer devices
* negotiating and instantiating security key parameters.

Android also supports WiFi protected access version 3 (WPA3) as well as virtual private network (VPN) for protected communications between apps and services in the cloud.

#### 6.3.2 Security mechanism in The App Store

The Apple App Store provides several security mechanisms to prevent any possible attack from malicious apps. Apple devices that run on iOS, by default, don’t allow users to install apps that are from websites or app stores other than the Apple App Store. In addition to this, the App Store requires app developers to sign their apps with a unique certificate ensuring the integrity and authenticity of the app. The security mechanisms can be classified as follows [APS]:

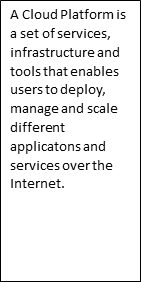
* Mandatory code signing: This mechanism requires developers to sign their apps with a unique certificate, helping to ensure the integrity and authenticity of the app. If code from third parties is used in the app, it is required that that code be validated with an Apple-issued certificate. This helps prevent tampering with the app’s code.
* Verifying proprietary in-house apps: When an organization wants to write proprietary apps, they have to apply to become part of the Apple Developer Enterprise Program (ADEP). After becoming a member of ADEP, an organization obtains a profile that allows for proprietary apps to run on the devices of the organization, but these apps will not be publicly available. Such apps are installed using the mobile device management system and are implicitly trusted.
* Sandboxing: Apps on iOS are sandboxed, meaning they are isolated from each other and the system, reducing the potential impact of security breaches.
* Data protection: Data stored on iOS devices from different apps is encrypted by default, adding an extra layer of security. Data protection is available for file and database APIs, including NSFileManager, CoreData, NSData, and SQLite.

From a network security perspective, the iOS platform uses security mechanisms in the transport and network layers as well as the lower communication layer. In the transport layer it supports TLS up to version 1.3 as well as datagram transport layer security (DTLS). In the network layer it supports VPN tunneling, while in the communication layer it supports WPA3 for WiFi as well as authentication, encryption, and message integrity checks for Bluetooth [APS].

### Self-Check Questions

* What are the ways of introducing malware applications into a mobile device?
* Re-packaging of legitimate apps, bug exploitation of genuine apps, fake applications, or remote installation of malicious apps.

## 6.4. The Role of the Cloud

A cloud platform is a set of services, infrastructure, and tools that enables users to deploy, manage, and scale different applications and services over the internet. They provide virtualized environments where users can run their solutions and services without the need for the user to own infrastructure, processing power, storage, network, or database. The three main service types that are offered by cloud solutions are infrastructure as a service (IaaS), platform as a service (PaaS), and software as a service (SaaS) [Jawale 2016]. While IaaS provides virtualized computing resources over the internet, PaaS offers a higher-level environment that also includes development tools, middleware, and runtime environments. Beyond this, SaaS delivers fully functional software applications over the internet where users do not need to install them locally but can access the applications via a web interface. Major cloud platforms include Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), IBM Cloud, and Oracle Cloud.

#### 6.4.1 Cloud for mobile app ecosystem

In the mobile app development and distribution ecosystem, the cloud plays a crucial role, both for Google Play as well as for the App Store. The cloud enables app distribution and management, as well as enhanced functionalities, features, and services.

#### Google Cloud Platform

Google Cloud Platform (GCP) is provided by Google as a tool for cloud computing services. It offers a wide range of infrastructure and platform services that enable businesses, developers, and organizations to build, deploy, and manage applications and services in the cloud. GCP provides scalable, flexible, and cost-effective solutions for various computing needs. GCP follows a pay-as-you-go model, where users can pay for and utilize cloud resources as needed. Users can create virtual machines, store and analyze data, build machine learning models, and develop applications without the upfront investment in hardware or infrastructure. GCP’s global network of data centers ensures low-latency access and high availability.

Services that GCP provides to users are diverse and can be categorize as follows [Bisong 2019]:

* Compute services: Provides scalable virtual machines (Google Compute Engine) and container orchestration (Google Kubernetes Engine) for, e.g., machine learning algorithms and big data handling.
* Storage services: Offers object storage (Google Cloud Storage), file storage (Google Cloud Filestore), and databases (Google Cloud SQL, Bigtable).
* Networking services: Includes networking services like virtual private clouds (VPCs), load balancing, and content delivery (Google Cloud CDN).
* Big data and analytics services: Offers data storage, processing, and analysis tools such as BigQuery, Dataflow, Dataproc, and Pub/Sub.
* Machine learning and AI services: Provides tools for building and deploying machine learning models, such as Google AI Platform, AutoML, and TensorFlow.

#### iCloud

Apple does not offer a comprehensive cloud platform as Google does; however, it provides a platform named iCloud that offers Apple device users cloud storage service and synchronization of files. iCloud is a cloud-based service that works seamlessly throughput all Apple devices: iPhones, iPads, etc. iCloud offers several services, including the following:

* iCloud Drive service: A file storage and synchronization service that allows users to store files, documents, photos, and other content in the cloud and access them from any supported device.
* iCloud Photos service: Automatically stores photos and videos in the cloud, ensuring that users’ media content is accessible across devices while saving local storage space.
* iCloud Backup service: Enables users to back up their devices to the cloud, ensuring that their data is protected and recoverable in case of device loss or damage.
* ‘Find My’ service that helps users locate lost or stolen devices, as well as share location with friends and family.

Summary

The mobile software ecosystem provides the mechanisms to allow the cooperation and interplay between different stakeholders. Mobile users, mobile apps and their developers, mobile device manufacturers, and app markets all cooperate to enable secure communication and secure usage of mobile apps.

In this unit, we revisited the Google Play and App Store ecosystems as two of the most widely used mobile application development and distribution platforms. For each of the ecosystem platforms we showed the life cycle of the app from the moment it is developed by a third-party developer until it becomes publicly available on the distribution platforms. In addition to this, we showed how each of the two platforms ensures user privacy and the privacy labels of the apps. Since an ecosystem is not only related to the technicalities of app development and distribution, we delved into monetization mechanisms in both platforms, including in-app purchases, subscriptions, and other methods developers can use to monetize their work.

In the third part of the unit, we analyzed the security providers and mechanisms for each platform. First, we looked at possible mobile application attack vectors as well as the different security mechanisms that Google Play and the App Store provide to overcome such attack vectors. In the last part of the unit, we analyzed the role of the cloud in mobile app distribution and analyzed in detail the two cloud platforms provided by Google (Google Cloud) and Apple (iCloud).

**Man-in-the-middle**

A type of cyber-attack where an attacker relays and alters the communication between two targets. By intercepting the communication between two parties, the attacker can eavesdrop on the conversation and potentially steal sensitive information or alter the contents of the communication.

# Unit 7 – Mobile handset threats

Study Goals

On completion of this unit, you will be able to …

* … Analyze and understand the historic development of cyber-attacks against handsets.
* … Understand and analyze the taxonomy of handset threats and different attack vectors against mobile devices.
* … Analyze the mitigation possibilities on the attack vectors against end devices.

## Introduction

With the development of communication technology and its increasing usage by the general public, the number of cyber-attacks in communication networks has increased as well. Attackers try to exploit different network security vulnerabilities by taking different paths to get access to networks or data communicated or saved via/in those networks. Such attacking paths are commonly known as attack vectors.

A network attack can be active or passive. In passive network attacks, attackers get access to the communication network; however, they do not change the data passing through/saved in the network but just steal sensitive data that can be used by them for malicious purposes. In active attacks, attackers get access to the network and change the data passing through/saved in the network by deleting them, encrypting them, or changing them.

Mobile handsets, as one of many devices that are connected to the network, are not exempt from such threats and attacks. Similar threats can also happen to mobile handsets. Attackers can get access to mobile handsets and can change data saved on them or sourced on them onto the network.

To mitigate the vulnerabilities and possible attacks in mobile handsets and in the network, an engineer should understand the common attack vectors used by the attackers. In this unit we will discuss the historic examples of handset attacks, a taxonomy of handset threats and attacks, and “jailbreaking.”

## 7.1 Historic Examples Of Handset Attacks

**Worm**

A worm is a type of malicious software (malware) that is designed to self-replicate and spread across computer networks without requiring any user action.

Viruses, malware, and worms are malicious software that can change the behavior of mobile devices and will try to get access to specific data inside the device. All three types of malicious software have their own characteristics and can be distinguished as follows [Dunham 2008]:

* Malware includes all types of malicious software, such as viruses, worms, and Trojans, and is designed to get access to mobile devices, networks, or data saved/communicated via them.
* Viruses are a subtype of malware that are attached to legitimate programs or files and require users’ interaction to distribute them by sharing infected programs and files.
* The worm is another subtype of malware that is standalone software that can be spread without users’ interaction via communication networks using system vulnerabilities.

While mobile devices are well protected from virus-type malware due to the use of sandboxing (different apps are sandboxed and do not share data), they are vulnerable to of worm-type malware. One of the earliest examples of mobile malware was the Cabir worm, which spread in early 2004 and targeted Symbian OS-based mobile handsets [Dunham 2008]. The attack vector used to spread this worm was Bluetooth communication. Even though the initial Cabir worm was not that risky, as it did not use data from the mobile handset, the source code of Cabir was spread by an attack group called 29A and became the base for a Symbian virus family. The Symbian virus family spread by mid-2005; it stole phone book addresses and distributed them via Bluetooth to the next symbian-infected device [Clooke 2017].

Another malware attack was DroidDream in 2011. It targeted Android devices by appearing as a legitimate third-party application on the Google app store. DroidDream could gain root access to the device, allowing it to steal sensitive information and perform various malicious activities. Once infected, a mobile phone could download other infected apps, degrading the security even further. Google responded by removing the infected apps from the Google Play Store and enhancing security measures.

Another malware was Judy in 2017, which affected Android devices similarly to DroidDream through infected apps on the Google Play Store. It used a technique called “auto-clicking” to generate fraudulent ad clicks, generating revenue for the attackers. Judy was estimated to have infected between 8.5 and 36.5 million devices before being discovered and removed from the Google Play Store [Judy 2017].

Once a possible attack vector is recognized, different approaches are taken to fix the issue by releasing a patch. One such example was the master key vulnerability in Android. Android requires that each app to be signed by the developer to determine its authenticity. However, due to discrepancies in application verification procedures in Android, it was possible for APK code to be modified without breaking the cryptographic signature [AMK 2013]. This allowed attackers to add different malicious functionalities to legitimate apps that stole various data from users. Google reacted to this, and this vulnerability was fixed in Android v4.4.

In the fall of 2021, the Silver Sparrow malware was released, targeting iOS devices running Apple’s new M1 chip. Although its exact purpose remains unknown, it could download and execute malicious payloads [Moses and Morris 2021]. These examples demonstrate the ongoing threat posed by cyber-attacks to mobile devices and highlight the importance of users staying vigilant and taking steps to protect their device and their information [Bhatt & Gupta 2017].

### Self-Check Questions

* What is the difference between a worm and a virus?

*A worm is standalone software and can get spread without the intervention of the user, while a virus is attached to a legitimate application or file and can be spread only with user involvement.*

## 7.2 Taxonomy of Handset Threats

The mobile device technology stack is composed of hardware, firmware, operating system, and applications. While the telephony subsystem uses dedicated hardware and firmware implemented in system on a chip (SoC), mobile devices also include various communication technologies and front ends, such as cellular radio, WiFi, Bluetooth, and near-filed communication (NFC).

Depending on what part of the mobile device technology stack is attacked, we will classify the threats in the following categories, as specified in NIST [2016]:

* Mobile application threats: related to threats and vulnerabilities coming from software for mobile applications running in the general-purpose operating system of the mobile device.
* Communication technology threats: related to vulnerabilities and threats coming from using communication technologies like cellular, WiFi, or GPS.
* Authentication threats: related to vulnerabilities and threats coming from authentication mechanisms between the user and the mobile device, the mobile device and a service in the network, or the mobile device and the network itself.
* Communication stack threats: related to vulnerabilities and threads to different elements of the mobile technology stack (operating system, firmware, SIM card, device drivers, or trusted execution environment (TEE)).
* Physical threats: related to vulnerabilities and threats from physical attacks and changes to the mobile device.

According to NIST [2016], the highest number of threats comes from the communication element (mainly cellular and WiFi communication), with stack threats and application attack threats next.

#### 7.2.1 Mobile application threats

When an app communicates with a server in the network, all unencrypted traffic at the application layer can be eavesdropped on by an attacker by accessing the physical wireless media. Even though the data might not be sensitive, attackers can use such data for other attacks on servers. To mitigate such attacks, a mobile user can use secure connection (HTTPS) to connect to the webserver, if possible; as an application developer, you can use another form of encryption employing a transport layer security protocol; while as an enterprise network manager, you can employ VPN connection to increase security [MTC].

Another form of attack on applications is man-in the-middle (MitM) attack. If a mobile app has a weak authentication mechanism (e.g., does not validate the server certificate), then a man-in-the-middle attack can occur, where the attacker impersonates the back-end server and receives all unencrypted traffic from the end device this server and can modify the data in transit [MTC]. Mitigations of MitM attacks are to use public key encryption in the communication server or to implement fail-safe logic where the communication is stopped if the server certificate validation fails.

While cryptographic mechanisms are used by mobile applications, a cipher that is easily broken by brute force can be exploited by the attackers. Saving the cryptographic key to the source code should always be avoided. Also, in order to avoid unauthorized disclosure, variables that save the encryption keys should be overwritten after each use.

Other mobile application attack threats include usage of third-party libraries that are vulnerable to attacks, or exposure of functions and data to untrusted apps that can be exploited by attackers to get access to sensitive data. To mitigate such threats, app vetting tools exist that identify apps that use vulnerable third-party libraries.

A malicious app is a software program that is designed to harm or exploit a device or its user. This can include stealing personal information, spreading malware, or performing unauthorized actions on the device. Users should exercise caution when downloading and installing apps, and only download from reputable sources, such as Google Play and the Apple App Store. It is also a good idea to regularly check for and remove any suspicious or unwanted apps from your device.

On mobile devices, malicious apps can be introduced in several ways [Sharma 2021]:

• The re-packing of a legitimate app includes the creation of a genuine app that can contain malicious code and distributing that code to users via third-party app stores.

• Bug exploitation refers to a genuine app that is used by an attacker to compromise the user’s data.

• Fake applications are used by attackers to make users think that they are installing the correct app. For example, a fake application of a bank app can be used by attackers to get the account access codes of users. Two-factor authentication is often used to prevent this.

• Remote installation is used by an attacker to remotely install malicious apps to a mobile device.

#### 7.2.2 Communications part threats

A mobile device is composed of multiple communications technologies, and there are several threats for each of them. Jamming of the air interface is one of the common wireless threats that is common to all wireless technologies. It is an attack vector that causes service interruption and decreases network availability for the mobile node. Radio jamming is the easiest attack vector in wireless mobile devices as it can be achieved by decreasing the signal-to-noise ratio for the mobile devices in certain channels by transmitting a dummy signal in the frequency band. Radio jamming can be done on specific channels at specific times to avoid detection and is referred to as smart jamming, while continuous jamming of all the frequency band is easily detected and is referred to as dump jamming [NIST 2018]. While for WiFi networks radio jamming can be easily achieved by a cheap software-defined radio (SDR), for cellular communication more advanced equipment is needed.

Another common threat for all wireless communication technologies is eavesdropping on the air interface of unencrypted traffic or simply encrypted traffic (recall WEP in WiFi networks). While in 5G networks the data traffic is both encrypted and integrity checked, in LTE the integrity of data plane traffic in the air interface is not checked. However, even if the data traffic is not encrypted at all, eavesdropping attacks in cellular radio interfaces (LTE or 5G) are complex. The attacker needs to know the exact channel that the user is using and the time slots that are used for the data traffic to be able to demodulate the captured traffic into data IP packets [NIS 2018].

Another attack vector is via compromised femtocells. Femtocells are used in both LTE and 5G standards for improving indoor cellular connectivity while using the owner’s internet connection to connect to the network. As such the standard requires use of the IP security [REF] protocol between the femtocell and the femtocell gateway to protect the data traffic on the internet outside the mobile operator network [NIST 2018]. However, if the femtocell is in the physical possession of an attacker, it can be compromised by giving access not only to the data traffic but also to the encryption keys used by the end device [DePerry 2013]. Similarly, small cells may be less frequently updated to patch security and configuration, exposing them to possible attacks to eavesdrop on calls and data passing through the cell [NIST 2018].

Other denial of service (DoS) attacks can be achieved by different attack vectors in cellular networks, including message injection, SMS-induced DoS, and silent message DoS [NIST 2018]. In all cases the attacker will send a stream of messages (or SMSs) to the mobile device to render the mobile device incapable of serving the user.

#### 7.2.3 Authentication Attack Threats

Mobile devices use a personal identification number (PIN) or user biometrics to give access to the mobile device. Biometrics such as fingerprinting or face recognition are included in the latest releases of mobile OSs in both Android and iOS.

Certain attack vectors try to gain access to the mobile devices by brute-force cracking of the PIN/password of the mobile device. Other attack vectors on the PIN/password used to access a mobile device include interfering with PIN/password information from the sensors of the device or by using screen smudges [NIST 2018]. Each mobile device has sensor data that can collect information on which part of the screen the user presses. If allowed, certain malicious applications can collect such information, inferring the PIN and passwords of the mobile device. Such attack vectors can be prevented or the required time to brute force the PIN/password can be extended by choosing longer or more complex PINs/passwords. To determine the number of all possible combinations of a subset of *n* elements out of a set with *k* elements, the following formula can be used:

So, the number of combinations of 4 ciphers out of 10 is only 210, which is not that high for the current processing times of modern processors. Further, if you want to find all the permutations a subset of n elements out of a set with k elements, the following formula can be used:

Other vectors attack authentication between the mobile device and services in the network. One such form of stealing authentication credentials is by sending phishing emails or using MitM attacks [MOBILE 2018]. Phishing emails are personalized emails sourced from attackers that appear to come from a trusted source, which direct the user to access a certain website. Then by providing the authentication credentials, the attacker can get access to the main user profile of that site. Similarly, MitM attacks redirect the user to malicious websites to steal their authentication credentials. Such attacks can be prevented by being more attentive to the phishing emails and to the websites we browse.

The last attack vector deals with the possibility of stealing authentication credentials of the mobile device in the network itself. A common attack vector in this regard is insecure credential storage of authentication data by apps in the device [MOBILE 2018].

#### 7.2.4 Physical threats

Physical threats include all attacks carried out in person by attackers. This includes device loss and theft, where the attacker will gain full access to the mobile device. In addition, SIM card swapping with a compromised SIM gives the attacker the possibility to run a malicious java applet in the new SIM [MOBILE 2018]. A new way to mitigate these forms of attack is integrated SIM cards or eSIMs, which cannot be physically swapped. Another physical threat is malicious charging stations that can cause battery overheating and battery damage. To prevent this attack, allow an overheated battery to cool down, use certified charging stations, or charge the mobile device from a USB port.

### Self-Check Questions

1. How many password combinations can be formed with three single-digit natural numbers?

*120*

Graphical user interface, logo, company name

Description automatically generatedDiagram, schematic

Description automatically generated

## 7.3 Jailbreaking

As mobile devices are used for plethora of tasks, including personal and business purposes, compromised devices are a major security issue for the applications running on those devices as well as for data being shared from them. Compromised devices are devices that have been altered to remove the limitations imposed on the device by its manufacturer or the operating system. This is done with the aim of giving low-level access to applications and users that do not possess such rights. In the Android context, such devices are known as rooted devices, while in the iOS context, such devices are known as jailbroken devices [Geist 2016].

In iOS the jailbreak is performed by patching /private/etc./fstab to mount the system partition in “read-write” mode so users can have access to the system partition [AW 2023]. This will allow arbitrary code to run in the kernel of iOS. More recently, jailbreaks can be achieved by patching the kernel code to bypass the code signing and other restrictions to run custom applications or third-party applications.

Types of jailbreaks include [AW 2023]

* Tethered, where the device must be booted using a computer in jailbreaking mode every time, otherwise it will not boot up.
* Semi-tethered, where the device must be booted using a computer in jailbreaking mode every time, otherwise it will boot up in normal mode.
* Untethered, where the device boots every time in jailbroken mode after it is jailbroken once.
* Semi-untethered, where the device needs to run an application to perform jailbreaking.

Apple Jailbreaking Processes

Diagram

Description automatically generatedDiagram

Description automatically generated

Source: Panji Harawa [2023], based on Dana et al. [2016].

The figure *Apple Jailbreaking Process* shows the process of jailbreaking an iOS device through a third-party application. It involves sideloading the said app onto the mobile device using SSH. Subsequently, the third-party application loads a developer certificate within the device’s settings. This certificate is then utilized to exploit vulnerabilities present in iOS. Once root access is obtained, an additional tool is introduced via the debugger, followed by a reboot of the device [Geist 2016]. This sequence of steps grants users the ability to manipulate the execution of the program. The tool provides precise control, allowing for identification and circumvention of potential obstacles. By setting breakpoints, the debugger inspects memory and alters the program’s state. This process effectively facilitates the implementation of the necessary bypassing strategy essential for the jailbreaking procedure.

For mobile devices, jailbreaking carries risks such as reduced security, heightened susceptibility to malware and cyberattacks, and incompatibilities with software and hardware. Once a device is jailbroken, it loses the manufacturer’s security protections, making it more prone to malware and various attacks. Moreover, jailbreaking may lead to compatibility problems, hindering the installation and usage of legitimate applications and services. [Adams 2015].

Jailbreaking also voids the device’s warranty, meaning that if the device is damaged or needs repairs, it will not be covered by the manufacturer’s warranty. In addition, jailbreaking may also cause the device to become unstable, leading to crashes and other issues that can be difficult to resolve. Jailbreaking can offer users increased flexibility and customization options, but it also comes with significant risks and dangers to the device’s security and stability. Before jailbreaking a device, it is important to consider the potential risks and weigh them against the benefits [Lee and Soon 2017].

### Self-Check Questions

1. What is jailbreaking and what are some potential risks associated with it?

*Jailbreaking is the process of removing software restrictions on an Apple device, such as an iPhone, to allow the installation of unauthorized apps and modifications to the device’s operating system.*

Summary

This unit covers mobile handset threats and focuses on understanding the types of threats, common attack vectors, potential consequences of security breaches, and best practices for protection. Mobile handset threats include malware, phishing, and unauthorized access to sensitive information, and have targeted mobile devices like smartphones and tablets since their widespread adoption. Notable historic examples of handset attacks, including the Cabir worm, DroidDream, and Judy, are described along with the measures taken against them. The discussion of mobile threats cannot be done without engaging in the taxonomy of handset threats as this provides a categorization of the security risks affecting mobile devices. Based on NIST, we have categorized handset threats into threats related to mobile applications, threats related to communication technologies used, threats related to authentication breaches, threats related to the communication stack, and physical threats. In addition, for each threat category we have discussed in detail examples of attack vectors that can be used and how they can be overcome. The unit further expounds on how jailbreaking is executed to remove limitations imposed by manufacturers or operating system providers on mobile devices and how it allows users to run unauthorized third-party software. While jailbreaking offers users more flexibility and customization options, it carries several risks, including reduced security, increased susceptibility to malware and cyberattacks, and potential incompatibilities with software and hardware. Moreover, jailbreaking voids the device’s warranty, which means that any damage or repairs will not be covered by the manufacturer. Additionally, jailbreaking can cause device instability, leading to crashes and other issues that may be difficult to resolve. Considering the potential risks associated with jailbreaking, users should carefully weigh the benefits against the drawbacks before proceeding.

# Unit 8 – Mobile Device Management

Study Goals

On completion of this unit, you will be able to …

… understand the benefits of mobile device management (MDM).

… understand the features and capabilities of bring-your-own-device (BYOD) solutions.

…. identify the potential risks and challenges associated with mobile devices.

…. explore best practices for implementing patching policy.

## Introduction

Mobile device management (MDM) refers to the practices, policies, and technologies used to secure and manage mobile devices such as smartphones and tablets, both in the workplace and for personal use. With the increasing popularity of mobile devices, and the increasing amount of sensitive data stored on them, MDM has become a critical issue for organizations and individuals alike. MDM solutions can be used to enforce security policies, monitor device usage, and ensure the integrity of data on mobile devices. This can include remote wiping of lost or stolen devices, monitoring of device activity, and controlling access to corporate data and networks. In this unit, we explore the various components of MDM, including the different types of mobile devices, the challenges posed by mobile device security, and the various MDM solutions available. We will also look at the various approaches to MDM, including the bring-your-own-device (BYOD) model. By the end of this unit, you will have a comprehensive understanding of MDM and the various solutions and approaches that can be used to secure and manage mobile devices in the workplace and for personal use.

## 8.1 The Threats Of BYOD

The bring-your-own-device (BYOD) trend, where employees bring their own personal mobile devices to work and use them for work purposes, has created new security challenges for organizations. In a BYOD scenario, the organization has limited control over the security of the device and its contents, and may be vulnerable to data breaches, theft of sensitive information, and malware attacks. Additionally, the use of personal devices for work purposes may also raise privacy concerns for employees. To mitigate the security risks posed by BYOD, organizations can implement the following strategies [Randhawa 2022]. There are various BYOD approaches [Boon 2015]:

* Full BYOD, where employees can use their own mobile devices (smartphone, tablet, laptop) for work without any restrictions in application used or operating system. While this provides the highest flexibility, it provides the greatest possible security threats.
* Choose your own device (CYOD), where employees can choose between a range of company-approved mobile devices that are configured with specific security settings.
* App-based BYOD, where users can access work-related resources (network and data) via specific apps. This is a highly secure approach as data resides in the network and all the interactions are done via highly secure links (e.g., VPN, IPsec).

In technology most things are tradeoffs. Higher flexibility for employees to use their own mobile devices comes with higher risks of security attacks. The possible threats of BYOD can be summarized as follows [Zambrano 2018]:

* Advanced persistent threat (ATP) is an advanced targeted cyber-attack that tries to detect vulnerabilities in an employee-owned device over a long period of time to access company network and/or data.
* Malware exploits BYOD vulnerabilities to steal confidential information of companies and employees. In cases where companies do not fully control the security policies of BYOD, malware can easily spread between the devices of the employees. Diversity of devices, lack of full control, phishing attacks, and possible unvetted apps pose a major challenge to the security of BYOD.
* Theft or loss of mobile devices is another challenge for corporate security. Since employees can take their own mobile devices anywhere, and such devices contain personal and corporate data, attackers can gain physical access to the corporate network and data via a stolen device. One approach to protect against stolen devices is the ability to erase sensitive data remotely.

#### 8.1.1 Mobile Device Management System

An organization or an enterprise, in addition to protecting its network, should protect its workers’ mobile devices (smart phones, laptops, tablets). A mobile device management (MDM) system is a software solution that is utilized by organizations to control functionalities of mobile devices, in order to prevent any sensitive data leakage by a misused or a lost device [Rhee 2012].

The MDM is composed of two main components: the MDM mobile application or MDM agent and the MDM server [Batool 2020]. The MDM application is installed on the mobile device, while the MDM server is typically hosted in the cloud or on the premises of the organization. The IT administrator of the organization interacts with the MDM server via an MDM console unit to enforce policies and restrictions. Further, based on the security level of the device, certain policies and restrictions are applied from the MDM server to the MDM agents. MDM can be used to manage devices that are company-owned or employee-owned.

Form a functional perspective, MDM includes [Rhee 2012]

* Device enrollment: This involves registering the mobile device with the MDM server and installing the client application.
* Policy management: This involves defining policies and restrictions for the mobile device, such as passcode requirements, app blacklisting, and WiFi settings.
* Device monitoring: This involves monitoring the status of the mobile device, such as battery life, storage capacity, and network connectivity.
* App management: This involves managing the installation and removal of apps on the mobile device, as well as enforcing app-specific policies.
* Data protection: This involves encrypting sensitive data on the mobile device, as well as providing remote wipe capabilities in case the device is lost or stolen.

Compared to other management methods, such as manual configuration or using consumer-grade tools, MDM offers centralized management for all the mobile devices of an organization, supporting consistency in applied rules and policies as well as high scalability. Furthermore, MDM can remotely monitor and manage mobile devices, including the possibility of erasing sensitive data in the event of device theft.

#### 8.1.2 Virtual private network (VPN) connection of mobile devices

Enterprises often restrict their services to only authenticated, authorized devices connected to their local network to avoid security issues such as data leakage and DoS attacks. However, when mobile devices are connected to the internet via public WiFi networks, they are vulnerable to security risks. To securely connect mobile devices to an enterprise network from a remote location, a virtual private network (VPN) is a reliable solution. By creating an encrypted tunnel between the mobile device and the VPN server, VPNs can prevent unauthorized access to sensitive information transmitted over the internet.

A VPN can be defined as a private network that has been constructed within the public network infrastructure, where access is limited to permitted peer connections [Ferguson 1998]. It uses packet encapsulation to wrap the data in an additional layer of encryption. The most common types of packet encapsulation used by VPN are Layer 2 Tunneling Protocol (L2TP) [RFC 3931] and Internet Protocol Security (IPsec).

L2TP connects two end points: the L2TP access concentrator (LAC) and the L2TP network server (LNS). L2TP is a combination of two main procedures that control connection management (including connection establishment and teardown) and session management (session establishment for incoming and outgoing calls) [RFC 393]. The procedure of establishing a connection in L2TP is as follows [RFC 393]:

* The L2TP connection begins with the LAC initiating a connection request by sending a start-control-connection-request (SCCRQ) to the LNS.
* The LNS responds by opening an L2TP tunnel and sending the start-control-connection-reply (SCCRP) to the LAC, including the tunnel ID.
* The LAC then sends a start-control-connection-connected (SCCCN) to the LNS and confirms the tunnel ID.

Once the connection is established, session establishment follows [RFC 393]:

* Once the LAC detects a call, it sends an incoming-call-request (ICRQ) packet to the LNS to establish an incoming call.
* Then the LNS responds with an incoming-call-reply (ICRP).
* The LAC accepts the call and responds with incoming-call-connected (ICCN).

The same procedure happens with outgoing call establishment, by exchanging outgoing-call-request, reply, and connected messages. L2TP provides a tunnelling mechanism to transmit data between two endpoints (e.g., between a mobile end device and the local enterprise network). However, it cannot provide full security because it does not provide any encryption or authentication for the data transmitted over the tunnel. L2TP can be vulnerable to several types of attacks, including eavesdropping, packet sniffing, and data tampering. As such, it is always used together with the IPSec protocol, which provides network-level security by encrypting and authenticating IP packets. Overall, a VPN connection based on L2TP and IPsec is an effective way to secure communication between a mobile device and the local enterprise network over the internet, providing confidentiality, integrity, and authenticity.

### Self-Check Questions

What are some potential security threats associated with BYOD in mobile devices?

*Some potential security threats associated with BYOD in mobile devices include the possibility of users accessing sensitive data on insecure networks.*

## 8.2 Unique Threats To Mobile Devices

Mobile devices, such as smartphones and tablets, have become an integral part of our daily lives. While they offer numerous benefits and conveniences, they also face a myriad of unique threats that can compromise user privacy, security, and device functionality. These threats include mobile malware targeting specific platforms, malicious apps in official app stores, unsecured mobile networks, location-based threats, mobile payment fraud, sensor-based attacks, and mobile botnets, among others. To ensure the safe use of mobile devices, it is crucial for users to be aware of these threats and adopt best practices to mitigate potential risks. In the following paragraphs, we will delve into the details of each of these unique mobile threats, highlighting their potential impacts and offering insights into how users can protect themselves against such risks. In addition, we will also describe SIM swapping attacks, access point name (APN) security threats, and SMS brute-force attacks, which are unique to mobile devices connected to mobile networks.

### 8.2.1 Malware

Mobile malware targeting specific platforms and malicious apps in official app stores are significant concerns for mobile users. Mobile malware is designed to exploit the unique features, vulnerabilities, or app ecosystems of specific platforms like Android or iOS. These malware types may come in the form of viruses, worms, Trojans, or ransomware, causing harm to the user’s data or device. Malicious apps in official app stores pose a unique threat since users trust apps that are available on these platforms. While app stores like the Google Play Store and Apple App Store have stringent security measures, some malicious apps can still bypass these checks, allowing users to inadvertently download and install them, exposing their devices to various security risks [Ezer et al. 2014].

Mobile devices also face another unique threat that targets all unsecured mobile networks, and location-based threats are also unique to mobile devices. Mobile users frequently connect to public WiFi networks or use mobile data, exposing them to network-based threats such as man-in-the-middle (MitM) attacks or WiFi eavesdropping. Attackers can intercept and manipulate network communications, gaining access to sensitive information or injecting malware into the user’s device. Location-based threats arise due to mobile devices’ reliance on GPS and other location services. Attackers can track a user’s location, enabling stalking, or deploy location-aware malware targeting users in specific geographic areas. This can lead to privacy breaches and potential physical harm [Akhtar et al. 2022].

Furthermore, mobile payment fraud, sensor-based attacks, and mobile botnets represent additional unique threats to mobile devices. As mobile payment systems gain popularity, attackers have devised new techniques to exploit these platforms and commit fraud. They may intercept payment information or manipulate transactions, causing financial losses. Sensor-based attacks exploit the various sensors found in mobile devices, such as accelerometers, gyroscopes, and microphones. Attackers can gather sensitive information or infer user activities by accessing these sensors, compromising user privacy. Mobile botnets involve malware-infected devices being remotely controlled by attackers, who can use them to launch distributed denial-of-service (DDoS) attacks or to spread further malware. These threats highlight the need for users to take necessary precautions when using mobile devices and to be aware of potential vulnerabilities [Joshua et al. 2016].

#### 8.2.2 SIM Swapping

Subscriber identity module (SIM) swapping can either be done physically by replacing the SIM card on the user equipment (UE) or by taking over the ISDN of the SIM and assigning it to another SIM card [Lin 2021]. By doing so, an attacker can gain access to the data network of the campus.

An attacker can come into contact with the original SIM either by picking up a crashed device drone on the campus or by having physical access to the campus network devices.

#### 8.2.3 Access point name (APN) security

The Access point name (APN) is configured by the mobile network operator in the UE and provides the identity of the gateway that connects the UE with the internet.

In the case of virtualized 5G campus networks, a mobile network operator can ask for a custom configuration of APN in the campus network UEs to achieve campus network separation and a certain quality of service (QoS).

Once the authentication between the UE and the core network has taken place, the APN identifier is exchanged between the UE and the core network. This message is already encrypted in the air interface, to be decrypted in the base station, and forwarded further in the core network. If the attacker has access to the interface between base station and the core network, then the APN name is noticeable in clear text [Lin 2021]. Alternately, if the attacker has physical access to the UE, it might change the SIM to cause the UE to communicate with the attacker gNB. Then the device will be rejected due to an unknown APN from the network, and the APN is retrieved by the attacker [Ciancaglini 2021].

A custom APN from the mobile network operator does not mean higher security. To avoid APN retrieval by attackers, VPN/internet protocol security (IPsec) between the base station and the core network is a must when using the public internet [Lin 2021].

#### 8.2.4 SMS brute force attacks

Many devices that are connected to a cellular campus network support remote SMS configuration in the event that the link to them is blocked or broken. Via SMS commands, the device can be rebooted, the connection status can be checked, or certain configuration parameters, such as APN, can be changed.

Even though SMS-based remote configuration is password-protected, the default passwords are usually the last four digits of the SIM card identifier [Lin 2021]. An attacker can get the last digits of the SIM card identifier, as they are the same as the international mobile subscriber identity (IMSI) identifier. As such, an IMSI-catcher attack can reveal the default password.

On the other hand, even if the password is changed, it is prone to brute-force attacks. Thus, it is recommended to have a strong password for an SMS-based configuration link. Another possible attack is to send an SMS to the device and make the device respond with the wrong password message to increase the cost of messaging for the company that owns the campus network. In addition, a denial-of-service attack is also possible in such a case.

### Self-Check Questions

Give two examples of unique threats to mobile devices.

*Two examples of unique threats to mobile devices include malicious apps and phishing.*

## 8.3 Patch And Policy Management

One of the key components of MDM is patch and policy management, which involves regularly updating the device’s software and enforcing security policies. Keeping mobile devices up to date with the latest security patches and software updates helps to protect against potential security vulnerabilities. Additionally, organizations can enforce security policies, such as password requirements, device encryption, and data wiping, to further secure the device and protect sensitive information [Kleiner and Disterer 2018].

Patch and policy management are crucial components of a comprehensive mobile device management (MDM) strategy. **Patch management** refers to the process of updating software on mobile devices to address security vulnerabilities and improve device performance. This can include updates to the device’s operating system and updates to the kernel, as well as updates to individual applications. Organizations can use MDM solutions to automate the patch management process and ensure that all devices are up to date with the latest security patches [Pilarski et al. 2015]. As part of an MDM system, patching and policy management include:

**Patch management**

This is the process of updating software on mobile devices to address security vulnerabilities and improve device performance.

* Security policy enforcement, such as encryption, password requirements, and app list usage.
* App control, where lists of apps to be installed and updated are managed.
* Patch management, including automated OS, app, and kernel updates.
* Remote wipe of the whole device, including software and data, in the event of a lost or stolen mobile device.

In addition, mobile devices employ policy management. This refers to the process of creating and enforcing security policies on mobile devices to ensure that they are used in a secure manner. Many software companies such as Microsoft and Google enforce and invoke policies specific to their software platforms. This can include policies on password protection, app usage, and strong password requirements; the last of these entails establishment of a policy that requires the use of strong passwords or passphrases, with a minimum length of 8–12 characters, including a mix of uppercase and lowercase letters, numbers, and special characters. Another potential policy is password expiration, where users’ passwords have to be reset after a specific period. These policies are implemented to govern the use of mobile devices and can help ensure that sensitive information is protected and prevent unauthorized access to corporate data. Organizations can utilize MDM solutions for policy enforcement; tracking device usage, including app utilization and device performance; and confirming policy compliance. According to Batool and Masood [2020], proficiently managing software updates – a process known as effective patch management, which involves applying and installing all necessary updates for critical software applications alongside policy administration – can assist organizations in safeguarding mobile device security and sensitive information. By implementing and enforcing security policies, it becomes possible to proactively avert security breaches and uphold the integrity of corporate data on mobile devices [Charles and Trouton 2020].

#### 8.3.1 Example: Android Device Policy

Android device policy enforces security policies of the organization that mobile device belongs to. This is to protect the organization’s data as well as the organization network. Android device policy works together with MDM solutions for remote control of policies applied in the mobile device [ADP].

Android device policy accesses the following device parameters [ADP]:

* Device’s location: used to check for available WiFi networks and roaming to new network when needed.
* Camera: used by the administrator to scan QR codes during enrollment in security policies.
* Phone number: used for device registration with MDM.

The administrator of the device can remotely apply security policies regarding length and strength of device passwords, number of failed passwords attempts before the device is cleared, time before the device password expires, idle time before the device gets locked, and encryption of device, among others.

### Self-Check Questions

How can patch and policy management be used to mitigate risks?

*Patch and policy management can be used to mitigate the risks associated with software vulnerabilities by ensuring that all software on the organization’s systems is up to date with the latest security patches.*

What does policy and patching management include?

*Patching and policy management includes security policy enforcement, app control, patch management (automated OS and kernel updates), and remote wipe.*

Summary

The unit highlights how the BYOD trend is gaining popularity, and how, as a result, organizations face new security challenges. Notably It is crucial for companies to establish strict security policies and train employees on best practices for using personal devices at work to reduce the risk of data breaches, sensitive information theft, and malware attacks. The module in other subsequent units clearly describes how mobile devices are vulnerable to various unique security threats, including platform-specific malware, malicious apps in official app stores, SIM swapping, APN security, and SMS brute force attacks.

Further, it has been described how users need to be aware of these risks and adopt best practices to mitigate potential vulnerabilities. They also need to play a role in protecting and preventing mobile threats to safeguard against security vulnerabilities. It is crucial to keep mobile devices updated with the latest patches and software updates. The unit explores two of the mechanisms used for this: patch management mostly involves updating the OS and individual applications, while policy management focuses on creating and enforcing security policies, such as password requirements and app usage restrictions. The unit further describes how organizations can use MDM solutions to automate patch management and ensure compliance with the overall security policies of the organization. A real-world example, Android Device Policy, is given.

# References

Agrawal, L. K., Agrawal, D., & Gopalaiyengar, S. K. (2022). Security and privacy in health data storage and its analytics. In Tanwar Sudeep (Ed.), *Blockchain for 5G Healthcare Applications - Security and Privacy Solutions*. Institution of Engineering and Technology. <https://doi.org/10.1049/PBHE035E_ch10>

Al-Qershi, F., Al-Qurishi, Rahman, S. M. M., & Al-Amri, A. (2014). Android vs. iOS: The security battle. *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*, 1–8. <https://doi.org/10.1109/WCCAIS.2014.6916629>

Arp, D., Quiring, E., Wressnegger, C., & Rieck, K. (2017). Privacy threats through ultrasonic side channels on mobile devices. *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, 35–47. <https://doi.org/10.1109/EuroSP.2017.33>

Batool, H., & Masood, A. (2020). Enterprise mobile device management requirements and features. *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Computer Communications Workshops (INFOCOM WKSHPS)*. 109–114. <https://doi.org/10.1109/INFOCOMWKSHPS50562.2020.9162763>

Bhatt, A. J., & Gupta, C. (2017). Comparison of static and dynamic analyzer tools for iOS applications. *Wireless Personal Communications*, *96*(3), 4013–4046. <https://doi.org/10.1007/s11277-017-4366-1>

Bigelow, S. J., & Lulka, J. (2022). kernel. Data Center. <https://www.techtarget.com/searchdatacenter/definition/kernel>

Brunschwig, L., Guerra, E., & de Lara, J. (2022). Modelling on mobile devices: A systematic mapping study. *Software & Systems Modeling*, *21*(1), 179–205. <https://doi.org/10.1007/s10270-021-00897-8>

Chen, C.-M., Liu, Y.-H., Cai, Z.-X., & Lai, G.-H. (2020). A power-efficient approach to detect mobile threats on the emergent network environment. *IEEE Access*, *8*, 199840–199851. <https://doi.org/10.1109/ACCESS.2020.3035192>

Collins, L., & Ellis, S. R. (Eds.). (2015). *Mobile devices: tools and technologies*. Chapman and Hall/CRC.

Edge, C., & Trouton, R. (2020). *Apple Device Management: A unified theory of managing Macs, iPads, iPhones, and AppleTVs*. Apress.

Google Workspace Learning Center. (n.d.). *About Android Device Policy*. <https://support.google.com/a/users/answer/9453213?hl=en>

Hamzenejadi, S., Ghazvini, M., & Hosseini, S. (2022). Mobile botnet detection: a comprehensive survey. *International Journal of Information Security*, *22*, 137–175. <https://doi.org/10.1007/s10207-022-00624-4>

Harris, M. A., & Patten, K. P. (2014). Mobile device security considerations for small- and medium-sized enterprise business mobility. *Information Management & Computer Security*, *22*(1), 97–114. <https://doi-org.pxz.iubh.de:8443/10.1108/IMCS-03-2013-0019>

*UEFI secure boot: Big hassle, questionable benefit*. (2012, June 12). Linux.com. <https://www.linux.com/training-tutorials/uefi-secure-boot-big-hassle-questionable-benefit/>

Jabar, T., & Mahinderjit Singh, M. (2022). Exploration of mobile device behavior for mitigating advanced persistent threats (APT): A Systematic literature review and conceptual framework. *Sensors*, *22*(13), Article 4662. <https://doi.org/10.3390/s22134662>

Kleiner, C., & Disterer, G. (2018). *Bring your own device: Mobile Device Management in Bildungskontexten*. Springer VS. <https://doi.org/10.1007/978-3-658-19123-8_19>

Laayu, M. R., Kurniawan, A., Cahyani, N. D. W., & Satrya, G. B. (2022). Comparison of acquisition results on iPhone 7 Plus (iOS 14.8.1) between jailbreaking vs non-jailbreaking device. *2022 10th International Conference on Information and Communication Technology (ICoICT),* 402–406. <https://doi.org/10.1109/ICoICT55009.2022.9914862>

Lee, M. S., & Soon, I. (2017). Taking a bite out of Apple: Jailbreaking and the confluence of brand loyalty, consumer resistance and the co-creation of value. *Journal of Product & Brand Management*, *26*(4), 351–364. <https://doi.org/10.1108/JPBM-11-2015-1045>

Olaleye, S. A., Ukpabi, D., Karjaluoto, H., & Rizomyliotis, I. (2019). Understanding technology diffusion in emerging markets: The case of Chinese mobile devices in Nigeria. *International Journal of Emerging Markets*, *14*(5), 731–751. <https://doi.org/10.1108/IJOEM-01-2018-0055>

Pilarski, B., Freier, P., & Schumann, M. (2015). Mobile Device Management - Eine strukturierte Marktanalyse. *HMD Praxis Der Wirtschaftsinformatik*, *52*(3), 373–385. <https://doi.org/10.1365/s40702-015-0117-5>

Tanenbaum, A. S., & van Steen, M. (2015). Modern operating systems (4th ed.). Pearson Education

Sahani, A. (12 2017). Android v/s IOS – The Unceasing Battle. *International Journal of Computer Applications*, *180*, 23–26. doi:10.5120/ijca2017915990

References 7 & 8

Saman Mirza Abdullah, Bilal Ahmed, & Musa Ameen. (2018). A New Taxonomy of Mobile Banking Threats, Attacks and User Vulnerabilities. *Eurasian Journal of Science and Engineering*, *3*(3), 12–20. <https://doi.org/10.23918/eajse.v3i3p12>

Tejinder S. Randhawa. (2022). *Mobile applications: Design, development and optimization*. Springer. <https://doi.org/10.1007/978-3-030-02391-1>

Tlachac, M. L., Reisch, M., Lewis, B., Flores, R., Harrison, L., & Rundensteiner, E. (2022). Impact assessment of stereotype threat on mobile depression screening using Bayesian estimation. *Healthcare Analytics*, *2*. <https://doi.org/10.1016/j.health.2022.100088>

Ullah, I., Boreli, R., & Kanhere, S. S. (2022). Privacy in targeted advertising on mobile devices: a survey. *International Journal of Information Security*. <https://doi.org/10.1007/s10207-022-00655-x>

Vashisht, S., Gupta, S., Singh, D., & Mudgal, A. (2016). Emerging threats in mobile communication systems. *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*, 41–44. <https://doi.org/10.1109/ICICCS.2016.7542341>

Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). Operating system concepts (10th ed.). John Wiley & Sons

Xu, S., Xia, Y., & Shen, H. (2022). Cyber Protection for Malware Attack Resistance in Cyber-Physical Power Systems. *IEEE Systems Journal,* *16*(4), 5337–5345. <https://doi.org/10.1109/JSYST.2022.3150576>

Yeboah-Boateng, E. O., & Amanor, P. M. (2014). Phishing, SMiShing & Vishing: An assessment of threats against mobile devices. *Journal of Emerging Trends in Computing and Information Sciences*, *5*(4), 297–307.

Zukarnain, Z. A., Muneer, A., & Ab Aziz, M. K. (2022). Authentication Securing Methods for Mobile Identity: Issues, Solutions and Challenges. *Symmetry*, *14*(4), Article 821. <https://doi.org/10.3390/sym14040821>

[Pawson 2022] Pawson, Richard. “The myth of the Harvard architecture.” *IEEE Annals of the History of Computing* 44.3 (2022): 59-69.

[Tan 2022] Tan, T., & Cao, G. (2022, May). Deep learning on mobile devices through neural processing units and edge computing. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications* (pp. 1209-1218). IEEE.

[Singh 2014] Singh, M. P., & Jain, M. K. (2014). Evolution of processor architecture in mobile phones. *International Journal of Computer Applications*, *90*(4), 34-39.

[Bhunia 2018] Bhunia, S., & Tehranipoor, M. (2018). Hardware security: A hands-on learning approach (1st ed.). Morgan Kaufmann.

[Franklin 2020] Franklin, J., Howell, G., Sritapan, V., Souppaya, M., & Scarfone, K. (2020). *Guidelines for Managing the Security of Mobile Devices in the Enterprise* (No. NIST Special Publication (SP) 800-124 Rev. 2 (Draft)). National Institute of Standards and Technology.

[Zhao 2014] Zhao, S., Zhang, Q., Hu, G., Qin, Y., & Feng, D. (2014, November). Providing root of trust for ARM TrustZone using on-chip SRAM. Proceedings of the 4th International Workshop on Trustworthy Embedded Devices, 25–36. <https://doi.org/10.1145/2666141.2666145>

[Lin 2018] Lin, Ph. Z., Perine, C., Voseler, R., Lin, W. (2018). Attacks from 4G/5G core networks risks of the industrial IoT in compromised campus networks. Trend Micro Research. [https://docu](https://docu/)ments.trendmicro.com/assets/white\_papers/wp-attacks-from-4G-5G-core-networks.pdf

[Stallings 2012] Stallings, W., & Paul, G. K. (2012). *Operating systems: internals and design principles* (Vol. 9). New York: Pearson.

# [Srikanth 2020] Srikanth V. (2020) Operating System(OS) Tutorial

[Android] <https://android.googlesource.com/>

[Marketshare 2023] <https://gs.statcounter.com/os-market-share/mobile/worldwide>

[Kotlin 2023] <https://developer.android.com/kotlin/first>

[A-API 2023] <https://developer.android.com/reference>

[Arch 2023] <https://source.android.com/docs/core/architecture>

[Objective-c 2023] <https://developer.apple.com/documentation/objectivec/objective-c_runtime>

[ART 2023] <https://source.android.com/docs/core/runtime>

[Xu 2016] Xu, M., Song, C., Ji, Y., Shih, M. W., Lu, K., Zheng, C., ... & Kim, T. (2016). Toward engineering a secure android ecosystem: A survey of existing techniques. *ACM Computing Surveys (CSUR)*, *49*(2), 1-47.

[DC 2023] Competition in the Mobile Application Ecosystem, Department of Commerce of the USA, 2023.

[STA 2023] Number of available applications in the Google Play Store from December 2009 to March 2022, Statista, <https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/>

[APP 2023] Apple App Store, <https://www.apple.com/app-store/>

[Egan 2004] Egan, W. F. (2004). *Practical RF system design*. John Wiley & Sons.

[Want 2006] Want, R. (2006). An introduction to RFID technology. IEEE Pervasive Computing, 5(1), 25–33. <https://doi.org/10.1109/MPRV.2006.2>

[Tanenbaum 2021] Tanenbaum, A. S. (2021). Computer networks (5 th ed.). Pearson Deutschland.

[Ekberg 2014] , J. E., Kostiainen, K., & Asokan, N. (2014). The untapped potential of trusted execution environments on mobile devices. *IEEE Security & Privacy*, *12*(4), 29-37.

[Trusty 2023] <https://source.android.com/docs/security/features/trusty>

[Kanonov 2016] Kanonov, U., & Wool, A. (2016, October). Secure containers in Android: the Samsung KNOX case study. In *Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices* (pp. 3-12).

[Samsung 2015] “Android security maximized by Samsung KNOX”, *White Paper, Samsung 2015.*

[Asmitha 2022] Asmitha, P., & Sunitha, T. (2022, July). Student Attendance using Face Recognition Technology. In *2022 International Conference on Intelligent Controller and Computing for Smart Power (ICICCSP)* (pp. 1-4). IEEE.

[Lai 2021] Lai, Xiaojun, and Pei-Luen Patrick Rau. “Has facial recognition technology been misused? A public perception model of facial recognition scenarios.” *Computers in Human Behavior* 124 (2021): 106894.

[Axente 2020] Axente, M. S., Dobre, C., Ciobanu, R. I., & Purnichescu-Purtan, R. (2020). Gait recognition as an authentication method for mobile devices. *Sensors*, *20*(15), 4110.

[Rui 2018] Rui, Z., & Yan, Z. (2018). A survey on biometric authentication: Toward secure and privacy-preserving identification. *IEEE access*, *7*, 5994-6009.

[Pace 2022] Pace, P. E. (2022). *Developing digital RF memories and transceiver technologies for electromagnetic warfare*. Artech House.

[Coppens 2022] Coppens, D., De Poorter, E., Shahid, A., Lemey, S., Van Herbruggen, B., & Marshall, C. (2022). An overview of UWB standards and organizations (IEEE 802.15. 4, FiRa, Apple): Interoperability aspects and future research directions. *IEEE Access*.

[Yeh 2009] Yeh, S. C., Hsu, W. H., Su, M. Y., Chen, C. H., & Liu, K. H. (2009, March). A study on outdoor positioning technology using GPS and WiFi networks. In *2009 International Conference on Networking, Sensing and Control* (pp. 597-601). IEEE.

[Zafari 2019] Zafari, F., Gkelias, A., & Leung, K. K. (2019). A survey of indoor localization systems and technologies. *IEEE Communications Surveys & Tutorials*, *21*(3), 2568-2599.

[Crato 2010] Crato, N., & Crato, N. (2010). How gps works. *Figuring It Out: Entertaining Encounters with Everyday Math*, 49-52.

[Fakhreddine 2016] Fakhreddine, A., Giustiniano, D., & Lenders, V. (2016, May). Evaluation of self-positioning algorithms for time-of-flight based localization. In *2016 14th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)* (pp. 1-8). IEEE.

[GPS 2023] <http://gpsinformation.net/main/gpstime.htm>

[Lenhart 2022] Lenhart, M. (2022). Distributed Relay/Replay Attacks on GNSS Signals.

[Papadimitratos 2008] Papadimitratos, Panagiotis, and Aleksandar Jovanovic. “Protection and fundamental vulnerability of GNSS.” *2008 IEEE International Workshop on Satellite and Space Communications*. IEEE, 2008.

[Dwivedi 2021] Dwivedi, S., Shreevastav, R., Munier, F., Nygren, J., Siomina, I., Lyazidi, Y., ... & Gunnarsson, F. (2021). Positioning in 5G networks. *IEEE Communications Magazine*, *59*(11), 38-44.

[PERM 2023] <https://developer.android.com/guide/topics/permissions/overview>

[Meneghello 2019] Meneghello, F., Calore, M., Zucchetto, D., Polese, M., & Zanella, A. (2019). IoT: Internet of threats? A survey of practical security vulnerabilities in real IoT devices. *IEEE Internet of Things Journal*, *6*(5), 8182-8201.

[Pielli 2015] Pielli, C., Zucchetto, D., Zanella, A., Vangelista, L., and Zorzi, M. (2015) “Platforms and protocols for the Internet of Things,” EAI Endorsed Trans. Internet Things, vol. 15, no. 1, p. e5, Oct. 2015.

[RFC7668] Nieminen, J., Savolainen, T., Isomaki, M., Patil, B., Shelby, Z., & Gomez, C. (2015). *Ipv6 over bluetooth (r) low energy* (No. rfc7668).

[Ronen 2016] Ronen, E., & Shamir, A. (2016, March). Extended functionality attacks on IoT devices: The case of smart lights. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)* (pp. 3-12). IEEE.

[STALLINGS2017] Stallings, W. (2017). *Cryptography and network security, 7/E*. Pearson.

[Fremantle 2017] Fremantle, P., & Scott, P. (2017). A survey of secure middleware for the Internet of Things. *PeerJ Computer Science*, *3*, e114.

[Mob 2008] *<https://www.zdnet.com/article/battle-for-the-mobile-platform-space-in-2008/>*

[MARKT 2021] <https://blog.gitnux.com/google-play-store-statistics/>

[GUIDE 2023] <https://developer.android.com/guide/platform>

[Android 2023] <https://developers.google.com/android>

[Wang 2019] Wang, Haoyu, Hao Li, and Yao Guo. “Understanding the evolution of mobile app ecosystems: A longitudinal measurement study of google play.” *The World Wide Web Conference*. 2019.

[GP Service 2023] <https://developers.google.com/android/guides/overview>

[GP Protect 2023] <https://developers.google.com/android/play-protect>

[GP monet 2023] <https://play.google.com/console/about/guides/play-commerce/>

[GP Sub 2023] <https://play.google.com/console/about/subscriptionsetup/>

[STA 2023] <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>

[App Sim] <https://developer.apple.com/documentation/xcode/running-your-app-in-simulator-or-on-a-device>

[App sub] <https://developer.apple.com/app-store/review/guidelines/>

[APP Priv] <https://developer.apple.com/app-store/user-privacy-and-data-use/>

[App PNL] <https://www.apple.com/privacy/labels/>

[MTC] Mobile Threat Catalog, <https://pages.nist.gov/mobile-threat-catalogue/>

[Sharma 2021] Sharma, T., & Rattan, D. (2021). Malicious application detection in android — A systematic literature review. Computer Science Review, 40, Article 100373. <https://doi.org/10.1016/j.cosrev.2021.100373>

[Android ES 2021] Android Enterprise Security Paper, April 2021

[Muhammad 2023] Muhammad, Z., Amjad, F., Iqbal, Z., Javed, A. R., & Gadekallu, T. R. (2023). Circumventing Google Play vetting policies: A stealthy cyberattack that uses incremental updates to breach privacy. *Journal of Ambient Intelligence and Humanized Computing*, 1-10.

[RFC4346] Dierks, T., & Rescorla, E. (2006). The transport layer security (TLS) protocol version 1.1. (RFC4346). Internet Engineering Task Force (IETF). <http://doi.org/10.17487/RFC4346>

[RFC8446] Rescorla, E. (2018). Rfc 8446: The transport layer security (tls) protocol version 1.3.

[APS] Apple Platform Security, May 2022, Apple Inc.

[Jawale 2016] Jawale, A. S., & Park, J. S. (2016, August). A security analysis on apple pay. In *2016 European Intelligence and Security Informatics Conference (EISIC)* (pp. 160-163). IEEE.

[Bisong 2019] Bisong, E., & Bisong, E. (2019). An overview of google cloud platform services. *Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*, 7-10.

[Dunham 2008] Dunham, K. (2008). *Mobile malware attacks and defense*. Syngress.

[Clooke 2017] Clooke, R. (2017) A brief history of mobile malware (Available online: <https://www.retaildive.com/ex/mobilecommercedaily/a-brief-history-of-mobile-malware>)

[Judy 2017] <https://blog.checkpoint.com/securing-user-and-access/judy-malware-possibly-largest-malware-campaign-found-google-play/>

[AMK 2013] <https://securityintelligence.com/understanding-the-android-master-key-vulnerability/>

[NIST 2016] Franklin, J. M. (2016) “Assessing Threats to Mobile Devices & Infrastructure, The Mobile Threat Catalogue”, National Institute of Standards and Technology (NIST)

[NIST 2018] Cichonski, J., Franklin, J. M., Bartock, M. (2018) “Guide to LTE Security”, National Institute of Standards and Technology (NIST)

[DePerry 2013] D. DePerry, T. Ritter, and A. Rahimis, Traffic Interception & Remote Mobile Phone Cloning with a Compromised CDMA Femtocell, Defcon 2013, Las Vegas, Nevada, August 1-4, 2013.

[MOBILE 2018] “Mobile Threat Catalog Website”, <https://pages.nist.gov/mobile-threat-catalogue/> (Accessed: September 2023)

[Geist 2016] Geist, D., Nigmatullin, M., & Bierens, R. (2016). Jailbreak/root detection evasion study on iOS and Android. *MSc System and Network Engineering*.

[AW 2023] Apple Wiki, Available online: <https://theapplewiki.com/wiki/Jailbreak>

[Boon 2015] Boon, G. L., & Sulaiman, H. (2015). A review on understanding of byod issues, frameworks and policies.

[Zambrano 2018] Zambrano, F. R. R., & Rafael, G. D. R. (2018). Bring your own device: a survey of threats and security management models. *International Journal of Electronic Business*, *14*(2), 146-170.

[Rhee 2012] Rhee, K., Jeon, W., & Won, D. (2012). Security requirements of a mobile device management system. *International Journal of Security and Its Applications*, *6*(2), 353-358.

[Lin 2021] Lin, Ph. Z., Perine, C., Voseler, R., Lin, W. (2018). Attacks From 4G/5G Core Networks Risks of the Industrial IoT in Compromised Campus Networks. *Trend Micro Research*

[Ciancaglini 2021] Ciancaglini, V., Costoya, J., Lin, P., & Reyes, R. (2021) Identified and authorized: Sneaking past edge-based access control devices. Trend Micro Research. [https://documents.tren](https://documents.tren/)dmicro.com/assets/white\_papers/wp-identified-and-authorized-sneaking-past-edge-based-access-control-devices.pdf

[ADP]https://support.google.com/a/users/answer/9453213?hl=en&ref\_topic=9335739&sjid=13490580988519079111-EU