**AVB**:

Android Verified Boot is a security feature of the newer Android versions. Its main purpose is to ensure that all executed code comes from a trusted source, rather than from an attacker or device corruption. It works using a chain of trust model, starting from the bootloader, and moving to the boot partition and other verified partitions like the system, vendor, and OEM partitions. When the device is started, each stage verifies the integrity and authenticity of the next stage before proceeding to it.

The following three pictures are the process of VB 1.0 in A/B and Non-A/B systems and the process of AVB 2.0:

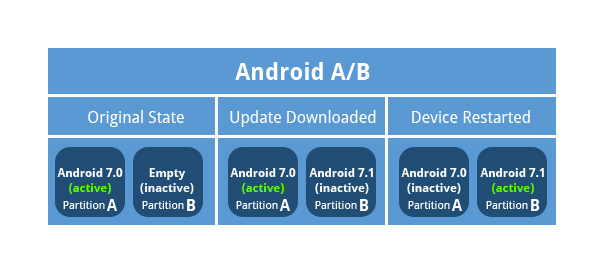
**VB 1.0 A/B Support :**

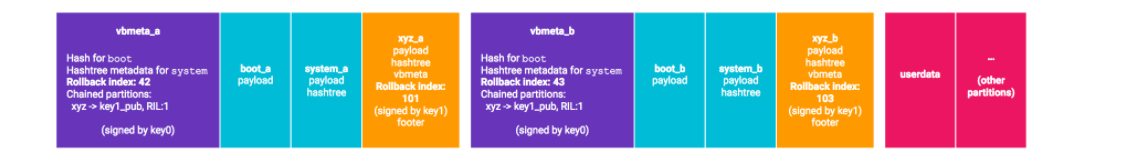
A/B refers to partition names. A is one partition and B is another partition.

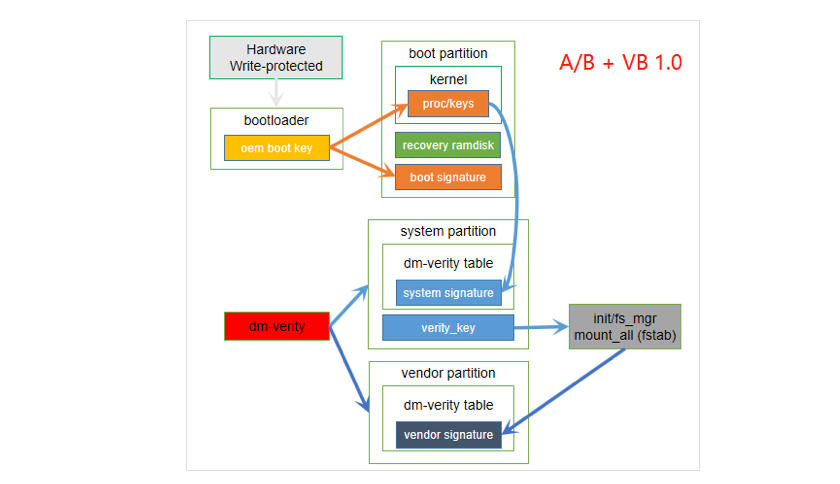
A/B system updates, also known as seamless updates.Seamless updates work by having two different system partitions on a device. When using a device that can perform seamless updates, all the transferring is done while one system partition is running, as things are copied and moved into the second partition.

Seamless updates aren’t a requirement for manufacturers. Our existing device partitions may not be A/B partitioned and won’t be able to perform seamless updates.

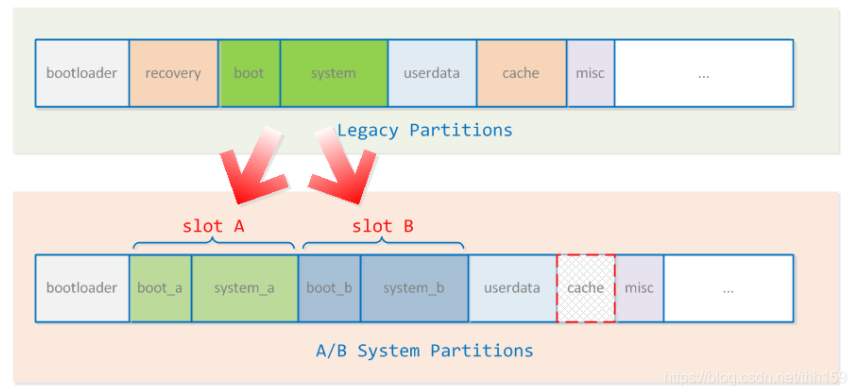
A/B OTA updates use more space on our devices internal storage. It uses a complete second system partition. Expect a couple of gigabytes to be used and reserved for it.







The A/B system update uses a so-called  update\_engine background daemon and two sets of partitions. These two sets of partitions are called slots, usually slot A and slot B. The system runs from one of the slots (the "current slot"), but the running system does not access the partition in the "unused" slot (for normal operation)



**VB 1.0 Non-A/B Support :**

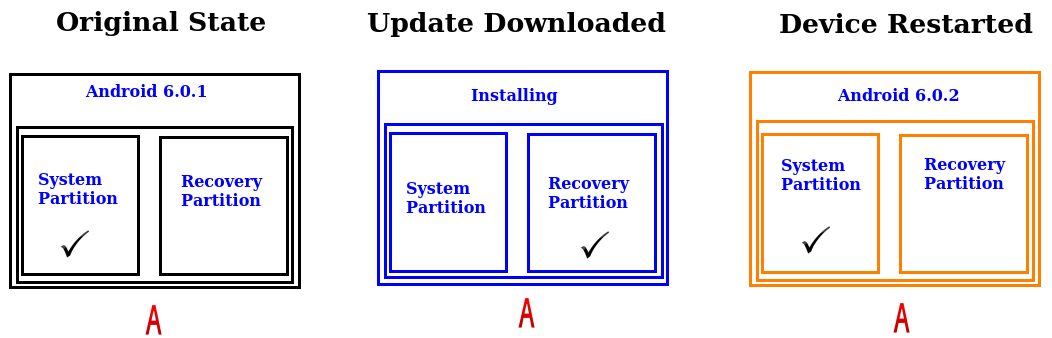
Android devices without A/B partitions have the following partitions, “**boot, system, vendor, userdata, cache, recovery, misc**“.

The life cycle of Non-A/B system update.

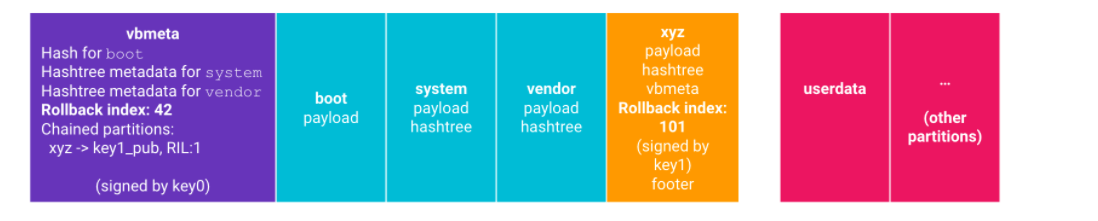
1. Devices perform checks with OTA servers and get notified of the availability of an update.
2. Downloads OTA packages to a cache or data partition.
3. Cryptographic signature is verified against the certificates in /system/etc/security. If everything looks fine, User is prompted to install the update.
4. Device reboots into recovery mode.
5. Recovery verifies the cryptographic signature of the package against the public keys in /res/keys(part of the RAM disk contained in the recovery partition).
6. Extracts the package to update the boot, system, and/or vendor partitions as necessary.
7. The system partition contains the contents for the new recovery partition(optional).
8. Device reboots normally.

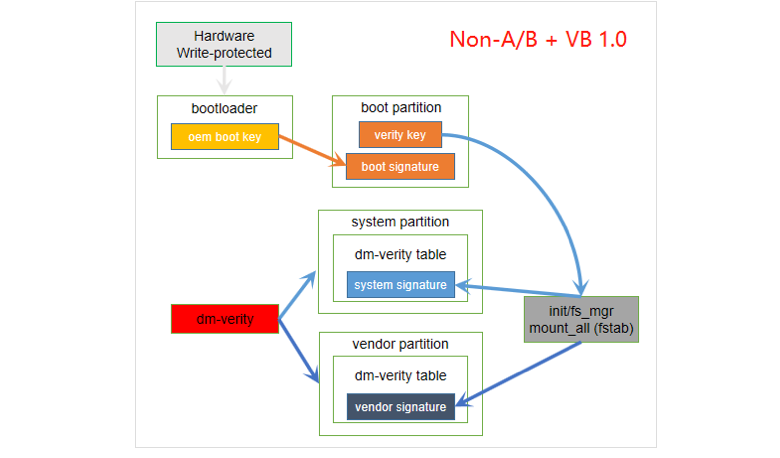
The system update is complete!

When non-A/B devices perform the above 8 steps for OTA upgrade their active partition or boot partition changes as shown in the picture below.



The central data structure used in AVB is the VBMetastruct. This data structure contains a number of descriptors (and other metadata) and all of this data is cryptographically signed. Descriptors are used for image hashes, image hashtree metadata, and so-called chained partitions. A simple example is the following:





A Comparison of A/B and Non A/B update

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.No** | **Components** | **A/B Updates** | **Non A/B Updates** |
| 1 | Recovery Partition available | No | Yes |
| 2 | OS | Partition A and Partition B | Only one Partition |
| 3 | Downtime | Minimal | High |
| 4 | Size | Requires High | Medium |
| 5 | Bootloader changes | Yes | Yes |

**AVB 2.0 :**

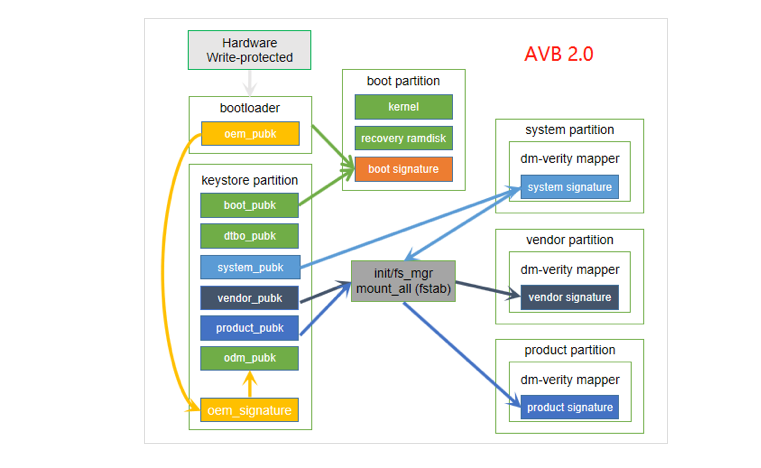
AVB2.0 is used in bootloader, it add an image “vbmeta.img” for this usage. A public Key will be compiled into Bootloader for verify vbmeta data and vbmeta.img include a signature that should be verified by this public key.

Verified Boot cryptographically verifies executable code before it is run. This includes the kernel, device tree, and the system and vendor partitions.

Small partitions, like boot (kernel) and dtbo (device tree) are checked using hashing.The entire partition is loaded into memory, then the hash is calculated and compared against an expected value. If the hash doesn’t match, Android will not load.

Larger partitions that can’t fit into memory (for example, file systems) need to use a hash tree where verification continuously happens as data is loaded into memory. The root hash is compared to an expected root hash value. If the values do not match at any point,Android enters an error state.

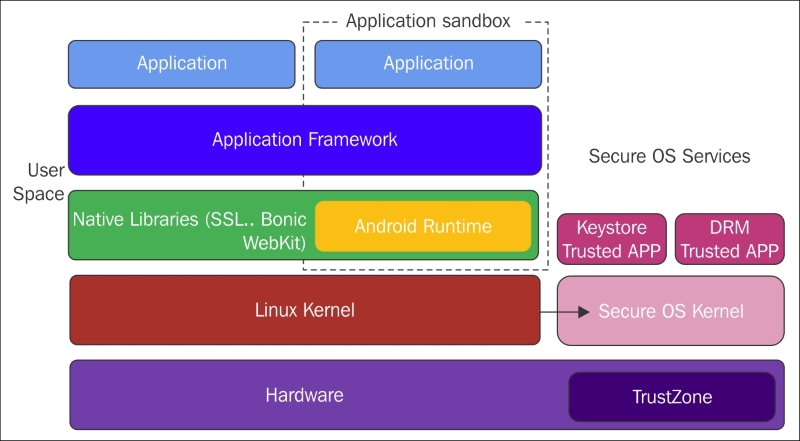
Expected hashes are stored at the beginning or end of each verified partition, or in a dedicated partition, depending on implementation. These hashes must be signed by the root of trust to ensure integrity. Verified Boot checks for the correct version of Android with [rollback protection](https://source.android.com/security/verifiedboot/verified-boot#rollback-protection).



====================================================================

Security

# Android Security Architecture



# Android Security Features

## App sandbox

App signing

Authentication

Biometrics

Encryption

Keystore

Security-Enhanced Linux

Trusty Trusted Execution Environment (TEE)

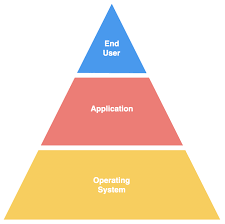
Verified Boot

* Android security model as a **three party consent model.** The security model is based on the consent of the following parties:

**Operating System**

**Application**

**End-User**



 Android is primarily focused on the end user, the system has to be secure by default.

**Operating system security:--**

The **kernel** is the core operating system software that handles the CPU resources, the system memory, the system devices, including the file systems and networking, and is responsible for managing all the processes. It serves as a link between the software and the hardware.

The kernel security determines the overall security of the whole system.

The security of the Android operating system is based around the following key security features of the Linux kernel:

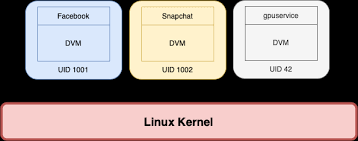
**Process Isolation**

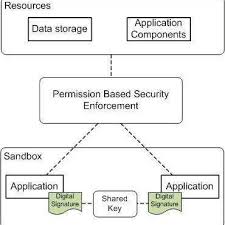
**User-Based Permission Model**

**Inter-Process Communication (IPC)**

## Sandboxing:--

Android platform uses the Linuxuser-based permissions model to isolate application resources. This process in called **application sandbox**.





The aim of sandboxing is to **prevent malicious external programs from interacting with the protected app**.

The secure communication between applications is ensured by the Linux user-based protection.

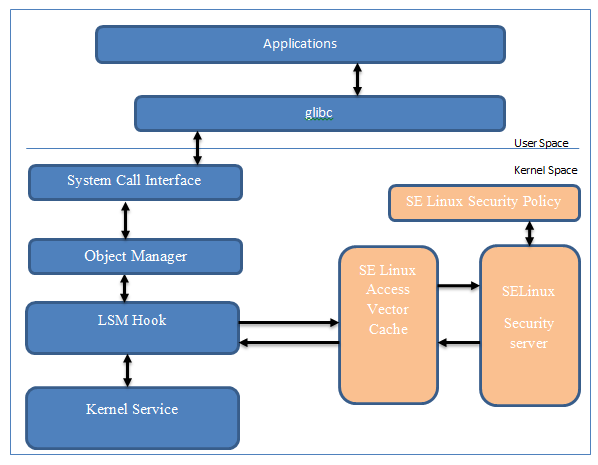
**Applicaton security:--**

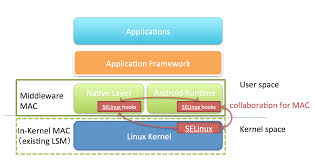
The permissions required by an application are declared in the **AndroidManifest.xml**. Every permission is specified in its own **uses-permission** tag.

Some permissions are granted to the application by default when specified. However, the other category of permissions, called **dangerous permissions**, require special consent from the user.

# Security-Enhanced Linux (SELinux):-

As part of the Android [security model](https://source.android.com/security), Android uses Security-Enhanced Linux (SELinux) to enforce mandatory access control (MAC) over all processes, even processes running with root/superuser privileges (Linux capabilities). With SELinux, Android can better protect and confine system services, control access to application data and system logs, reduce the effects of malicious software, and protect users from potential flaws in code on mobile devices.





SELinux operates on the principle of default denial: Anything not explicitly allowed is denied.

SELinux can operate in 2 modes which are **Enforcing** and **Permissive**. The default mode is Enforcing.

* In Enforcing mode, SELinux actively enforces the given policy which specifies what is allowed (permissions in general). If an initiator wants to perform an action, SELinux will check if it is allowed to do so in the installed policy, and if allowed, it will then permit the requested action to happen. If denied, it will be logged in the kernel log buffer along with logcat on Android.
* In Permissive mode, SELinux will only log actions which are explicitly not allowed in the installed policy, and the initiators of those actions.

Android includes SELinux in enforcing mode and a corresponding security policy that works by default across AOSP. In enforcing mode, disallowed actions are prevented and all attempted violations are logged by the kernel to dmesg and logcat.