Sensor-based Magnetic Covert Channels on Mobile Devices

Documentation

Alban Thaumur Samyak Puri Apurva Jagdale Alexander Ulyanov

March 18, 2020

Contents

1 Theoretical Background

1.1 Side-channel attacks and covert channels

1.1.1 What are side-channel attacks? When are they used?

Side-channel attack is any attack based on information gained from the implementation of a computer system, rather than weaknesses in the implemented algorithm itself. In this attack one process gains information about a second process that it is not supposed to obtain [40], i.e. there is one malicious entity attacking an honest entity.

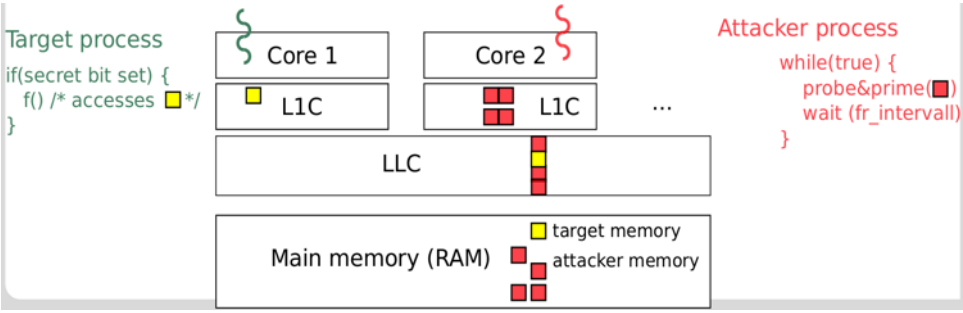


Figure 1.1: Prime and Probe Cache based Side-channel Attack

- *Timing Attack*: attacks based on measuring how much time various computations (such as, say, comparing an attacker's given password with the victim's unknown one) take to perform.
- *Cache Attack*: attacks based on attacker's ability to monitor cache accesses made by the victim in a shared physical system as in virtualized environment or a type of cloud service. Figure 1.1 is most common attack in this category is called the Prime and Probe attack. It measures the time needed to read data from memory pages associated with individual cache sets. If the victim accesses a primed line, data on the line will be evicted and caused a cache miss.
- *Power-Monitoring Attack*: attacks that make use of varying power consumption by the hardware during computation.
- *Electromagnetic Attack*: attacks based on leaked electromagnetic radiation.

- *Acoustic Cryptanalysis*: attacks that exploit sound produced during a computation.

Side-channel attacks are possible when:

1. Physical access to the system on which victim process is running.
2. Technical knowledge about the internal operations of the process is known.
3. Attacker is knowledgeable about hardware architecture of the system.
4. Adversary process can access information without making benign process aware about it.

1.1.2 Passive and active side-channel attacks, hardware- and software-based attacks. What are electromagnetic side-channel attacks? Provide examples. Which countermeasures exist against EM side-channel attacks?

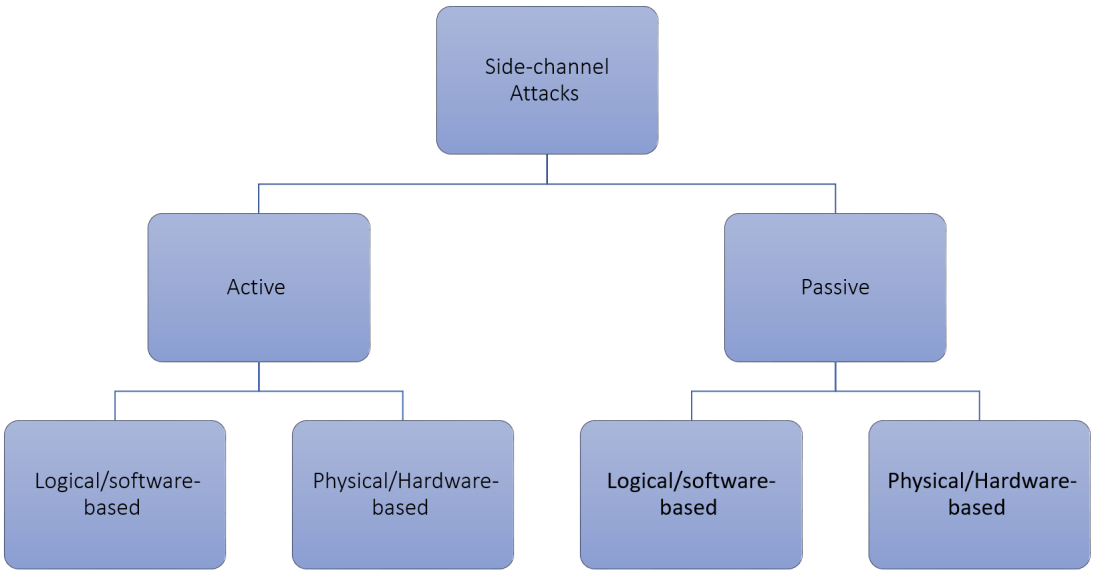


Figure 1.2: Classification of Side-Channel Attacks [37]

Figure 1.2 from Raphael Spreitzer et al. [37] classifies Side-channel attacks broadly into active and passive side-channel classes based on initiation of attack. Which further splits into Physical (Hardware-based) and Logical (Software-based) attacks.

Passive side-channel attacks In figure 1.3, A target represents anything of interest to possible attackers. During the computation or operation of the target, it influences a side channel (physical or logical properties) and thereby emits potential sensitive information. An attacker who is able to observe these side channels potentially learns useful information related to the actual computations or operations performed by the target. Therefore, an attacker models possible effects of specific causes. Later on, careful investigations of observed effects can then be used to learn information about possible causes.

For example, Cache based side-channel attacks most of the times can be classified as passive side-channel attacks.

Active side-channel attacks An active attacker tampers with the device or modifies/influences the targeted device via a side channel, e.g., via an external interface or environmental conditions. Thereby, from figure 1.3 the attacker influences the computation/operation performed by the device in a way that allows to bypass specific security mechanisms directly or that leads to malfunctioning, which in turn enables possible attacks.

For example, physically operating a smartphone via the touchscreen, i.e., the touchscreen input represents the target, causes the smartphone to undergo specific movements and accelerations in all three dimensions. In this case, one possible side channel is the acceleration of the device (a physical property), which can be observed via the embedded accelerometer sensor and accessed by an app via the official Sensor API.

Software and Hardware based attacks This category from figure 1.2 classifies side-channel attacks according to the exploited information, i.e. depending on whether the

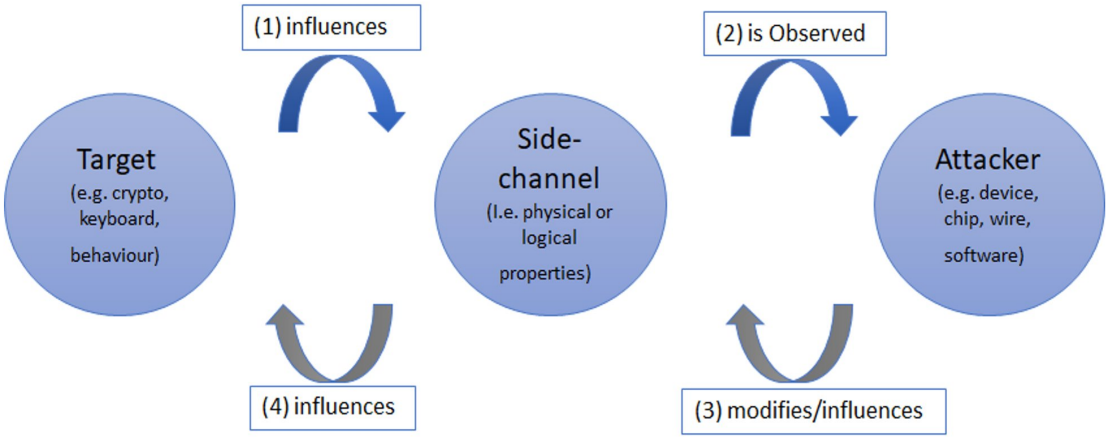


Figure 1.3: Flow of Side-Channel Attacks [37]

attack exploits physical properties (hardware) or logical properties (software features). Physical properties include the power consumption, the electromagnetic emanation, or the physical movements of a smartphone during the operation. Logical properties include usage statistics provided by the operating system, such as the data-usage statistics or the memory footprint of an application.

Electromagnetic side-channel attacks The Wikipedia defines electromagnetic side-channels in the following way:

> ... Electromagnetic attacks are side-channel attacks performed by measuring the electromagnetic radiation emitted from a device and performing signal analysis on it. [39]

The EM analysis attack is typically performed in two phases. In the first phase, the attacker collects the EM emanations using an EM probe optionally connected to a low-noise amplifier (LNA) placed in the vicinity of the encryption device under attack. In the second phase, the collected EM traces are subjected to simple (SEMA) or differential EM analysis (DEMA) to extract the secret key of the encryption device. [21]

- *Smart cards*: By measuring the variations of the electromagnetic field over the chip surface or by obtaining power measurements.
- *Smartphones*: The magnetometer sensor on mobile devices can be disturbed by electromagnetic activity caused by the smartphone's CPU.
- *Personal Computers*: Using near-field magnetic probes attacks EM attacks on personal computers are possible under certain conditions.

Countermeasures against EM side-channel attacks [8]

- *Logical Level*: The focus is on designing DPA (Differential Power Analysis) resistant logic styles which consume equal power in each clock cycle. As EM side-channel attacks are mostly used along with the power-monitoring attacks this can be one of the countermeasures.
  Example: Sense Amplifier Based Logic (SABL), Wave Dynamic Differential Logic (WDDL), Bridge Boost Logic (BBL), Masked Dual-Rail Pre-charge Logic (MDPL)
- *Architecture Level*: The power consumption characteristics is defined by the underlying hardware.
  - *Introducing Time Distortion*: Can be done only by random insertion of dummy operations or by shuffling of operations.
  - *Introducing Amplitude Distortion*: By choosing instructions with lowest leakage, avoiding conditional jumps or usage of memory addresses depending on key, and thus reducing amplitude of leakage.

- Physical Level:
  - *Noise Injection*: Generally using Attenuated Signature Noise Injection (ASNI)
  - *Two layered STELLAR*: Signature aTtenuation Embedded CRYPTO with Low-Level metAl Routing. With Electromagnetic field Suppression and Signature Suppression. [10]

1.1.3 What are covert channels? Side channels as a means for covert channels. Examples of electromagnetic covert channels.

Covert Channels are attacks in which two processes that are not supposed to communicate gain the capability to transfer information between them, i.e. there are two collaborating entities.

Side channels as a means for covert channels Every side channel can also be used as covert channel. As per definition of the covert channel inference is Active Side-channels can also be called Covert-Channels.

Electromagnetic Covert-channel

- *CPU emissions and a dedicated receiver*: Recording the EM signals emitted by the activities on CPU.
- *CPU-RAM emissions and a mobile phone with patched firmware*: Rogue software on an infected target computer modulates and transmits electromagnetic signals at cellular frequencies by invoking specific memory-related instructions and utilizing the multi-channel memory architecture to amplify the transmission. [14]
- *Hard drive side-channel attack*: Head movements lead to EM fluctuations. Magnetic sensors can detect hard drive activity. writing random data to the hard drive produces a single peak.

1.1.4 Use cases of Covert Channels.

Ex-filtration of sensitive data from isolated computers. What is an air-gap? Ex-filtration from isolated systems The data can be extracted from isolated systems using the data generated by the system being leaked in the environment in two ways.

1. Active method
   The isolated systems are infected using some method which create covert channel by manipulating system components to leak data via the environment.

2. Passive method
   In this method the data being leaked unintentionally by the system is accessed and the sensitive data recovered.

What are Air Gaped systems?

Air gaped systems are isolated systems which are removed from the network, either standalone machines with no networking or a private network not connected to any external networks whose access is restricted normally.

They can also have total isolation with external systems by removing access to hardware allowing access to external systems/data to these isolated system i.e. no access to CD drive, floppy drive, USB, ethernet port, parallel port or any other such port.

These can either be disabled in software or physically removed from the device to provide no attack vector.

Methods of getting the Data:

1. Audio

Using the speakers of the system to generate ultrasonic information channel by a malware. Other sources of audio noise inside the system can be used to transmit sensitive data as shown by:

- Fans in case of Fansmitter [16]
  - Frame format 16bits (4-bit preamble – 16-bit data)
  - Bandwidth 10 bpm to 15 bpm (at 8 m to 1 m)
  - SNR, capacity and bandwidth is heavily dependent on environment noise.
- Hard drive as used by DiskFiltration [15]
  - Capacity 40bits (4-bit preamble – 26-bit data)
  - Bandwidth 180 bpm
  - SNR was 1.5 at specific frequency as compared to 12 at full frequency range

2. Seismic

The vibrations generated by different systematic components while running different operations can provide a lot of information about what the system is doing and even leak a lot of sensitive information. Examples:

- Keystroke detection:
  - Cell phone accelerometer can be used to detect keys being pressed. It needs to be near 1 inch to 2 inch of the keyboard [24].

   - Laser pointed at a laptop to detect keystrokes as they would make laptop vibrate [5]

3. Electromagnetic

Making use of Electromagnetic emissions generated by different components of a system to infer sensitive data or what the system is doing while the emissions are being monitored. Also, different methods can be employed to specifically generate EM emissions to transfer data.

Examples:

- Airhopper [18]
  - Screen images can be captured via EM radiation of cable
  - EM radiation can be manipulated by transmitting specially crafted images
  - In FM range 88 MHz to 108 MHz
  - Bandwidth 13 bps to 60 bps (at 1 m to 7 m)
- USBee [11]
  - Use D+/D- lines as an antenna making use of seek commands
  - Simple I/O(read/write) commands, no permissions required
  - Bandwidth of 20 bps to 810 bps
- GSMem [13]
  - Uses multichannel memory to amplify signal generated by read write instructions to transmit at cellular frequency
  - Bandwidth 100 bps to 1000 bps
  - SNR as low as 0.5 dB can be achieved
- Wired keyboard sniffing [38]
  - Log keystrokes by analyzing EM radiation from keyboard wire
  - SNR under 6

4. Magnetic

The magnetic fields generated by system components can be exploited to exfilterate data from air gaped systems. Examples:

- ODINI [17]

  - based on low frequency magnetic fields generated by CPU
  - can bypass shielding and Faraday cage
  - Maximum SNR 36 dB and Minimum SNR 5 dB under various conditions (different magnetic fields and workloads)
  - Bandwidth 1 bps to 40 bps (100 cm to 5 cm)

- MAGNETO [12]
  - Makes use of smartphone magnetic sensor to receive data from nearby computer even in Faraday cage
  - Can work with virtual machines as Host CPU magnetic field corresponds with that of VM
  - SNR 17 dB to 10 dB (5 cm to 15 cm)
  - Frame format 32-bits payload with 4-bit preamble and 4-bit CRC

Communication between isolated processes. Covert channels in cloud environments.
Communication between isolated processes

In a system with process isolation, limited (controlled) interaction between processes may still be allowed over inter-process communication (IPC) channels such as shared memory, local sockets or Internet sockets.

In this scheme, all of the process' memory is isolated from other processes except where the process is allowing input from collaborating processes.

System policies may disallow IPC in some circumstances. For example, in mandatory access control (MAC) systems, subjects with different sensitivity levels may not be allowed to communicate with each other.[32]

The isolation can be provided by different methods like Apparmor which gives granular control over what resources the process can access which can limit or stop communication with other processes.

Covert channels in cloud environments

The cloud environment isolates the virtual machines from each other for providing security. This aims to prevent information leak to other user as he does not have access to the same resources. So, a process on different VM cannot access any resource assigned to other VM.

There is a common management system (hypervisor, Virtual Machine Monitor) which communicates with all VMs to manage resources between them and provide a layer to translate instructions run on the virtual process to the processor on host machine.

In public cloud networks the VMs migrated in the Data Center they are located in to better manage resources as and when required by the user, also, for resource management so that least amount resources are under utilized.

A covert channel can be created by using the method of Round Trip Time(RTT), where, it defines the '1', or '0' as the RTT changes if the location of the VM is changed from one DC to other. The user can at any time change the location of VM which can take few seconds to hours depending on data in the VM. If the size of VM is low it can be achieved to use it to send data by shifting VM from DC to DC. The bandwidth is very limited as the migrations take time and are not instantaneous. [35]

Some cloud providers make use of shared libraries between VMs to better utilize the resources available. This made the VMs vulnerable to covert channels via method of CPU cache. Now a days most of the cloud providers have removed shared libraries between Vms.

Cross-device tracking In this age of multiple devices per user it becomes important from the prospect of companies to track user activity across all devices for analytics and advertisements. The advertisement companies especially want to track users and the activities related to the advertisements shown to them to better target the user.

There are multiple methods to track user:

- Unique id/code dependent on the user being a registered user on the website being visited.
- Creating unique ids based on the device,browser, etc. information and track the activities.

Hardware methods for cross device tracking methods are becoming more common. Some TVs transmit an ultrasonic sound signal embedded in the advertisement being broadcast which can be listened by the smart phones (Ultrasonic cross device tracking [26]).

This is then used to check if the user searches for it or buys the item and can be correlated with the tracking id created by the software methods to know after how many times the user bought the item or if they should stop showing the ad now on the phone/desktop.

The tracking can also be done via the method of RF beacons if physical advertisements or locations are equipped with these the signals can be detected by the user devices and can be used to correlate the user data and perform analytics. The same thing as used for ultrasonic tracking can be used by using RF if the systems are RF equipped.

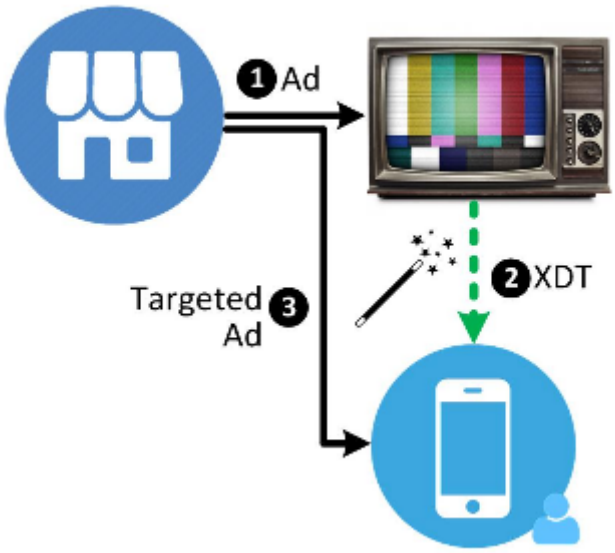


Figure 1.4: Illustration of how advertisers use ultra-sound cross-device tracking to target users across multiple devices.

Can electromagnetic emission be used as a covert channel for each category?

- Isolated Systems: EM emission can be used as a method for getting sensitive data or for creating covert channels
- Cloud Environments: As the VM/environment is stored in far of center it is difficult to use EM emissions to gather data. If physical access is possible the attacks are feasible as the CPU activity is correlated between VM and host.
- Cross-device tracking: The use of EM emissions is possible specially if the TVs have RF parts integrated. RF beacons can be used as the ultrasonic is used.

1.2 What are capacity and bandwidth of covert channels? What is Signal to Noise ratio?

1.2.1 Channel Capacity

Channel capacity, is the maximum or the tight upper bound rate at which information can be transmitted reliably over a communication channel.

It can also be defined as the maximum transfer rate of information (unit information per unit time) of the communication channel which can be achieved with arbitrary small error rate.

Cover channels have limited or low to very low channel capacity due to use of unconventional and slow (rate of change of method is slow) communication channels.

Bandwidth

Bandwidth has two different meaning:

1. Computing The bandwidth is the maximum data transfer rate over a path. It can be referred to as *Network Bandwidth* or *Channel Capacity*. It is the maximum amount of information that can be transmitted per unit time over a particular method or path.
2. Signals and Systems The bandwidth is the difference between the upper frequency and the lower frequency that can be carried with acceptable losses over a transmitting medium. It is used to specify filters, transmitters and receivers.

In covert channels it is widely varying depending upon the method being used as the channel. It may refer to as both definitions in these cases. The methods utilizing optical or electromagnetic properties for communication will have the Channel capacity for the bit rate and bandwidth for the frequencies. In these cases multiple frequencies can be utilized to increase the channel capacity if the bandwidth is wide enough to not have large symbol errors. The methods utilizing other methods where multiple signal cannot be used for simultaneous transmission the channel capacity and bandwidth refer to the same thing the maximum bit rate.

1.2.2 Signal to Noise Ratio

The Signal to noise ratio (SNR or S/N) is the measure of desired signal to the background noise. It is defined as the ratio of signal power to the noise power. It is given in the decimals. Its general formula is

$$SNR = \frac{P_{signal}}{P_{noise}}\text{dB}$$

The cover channels have varying SNR depending upon the environment conditions and how sensitive the method being used it to the environmental changes.

1.3 How can attackers use covert channels on smartphones? Describe the application sandboxing and permission model on modern mobile OSes.

1.3.1 Covert Channels on Smartphones

There a number of covert channels which have been researched a lot on smartphones. These channels are either:

- Storage based
- Timing based

Some unique channels that can be used in smartphones:

- Call/Phone:
  An app can make calls and the phone numbers are stored in ASCII of arbitrary length. Two clouding apps can communicate using this method.
- Battery:
  The phone broadcasts change of every 1% to apps subscribed, one app can run heavy operations to drain battery and encode data in this method.
- Screen:
  Making use of the screen brightness can be used to exchange data by app reading the screen brightness when a specific screen variable is set or unset.
- Volume:
  There are a number of volume settings which can be manipulated by the apps to set specific volume numbers to covertly send data.

There are problems with these methods as parallel apps can affect these and even user may change these values and disrupt communication. Use of sequence to designate data can be used to bypass this. [7]

Application Sandboxing [3] The Android platform takes advantage of the Linux user-based protection to identify and isolate app resources. It uses the UID to set up a kernel-level Application Sandbox. The kernel enforces security between apps and the system at the process level through standard Linux facilities such as user and group IDs that are assigned to apps. By default, apps can't interact with each other and have limited access to the OS.

- SELinux provided mandatory access control (MAC)
- Isolate apps across the per-physical-user boundary.

- All apps are set to run with a *seccomp-bpf* filter that limited the syscalls
- All non-privileged apps with *targetSdkVersion >= 28* must run in individual SELinux sandboxes, providing MAC on a per-app basis.
- Apps have a limited raw view of the filesystem, with no direct access to paths like /sdcard/DCIM

Permission Model [31] Android permission model allows the apps access to various resources and data. The apps have to ask for permission from the user at runtime for access to the resources.

The permissions are divided as:

- **Zero-Permission** is the access to the resources provided by the android system without any restrictions. Every app can access these without requiring any specific permissions.

  The access to the sensors such as accelerometer, magnetometer and gyroscope comes under this [28].

- **Normal** permissions cover areas where your app needs to access data or resources outside the app's sandbox, but where there's very little risk to the user's privacy or the operation of other apps.

  If an app declares in its manifest that it needs a normal permission, the system automatically grants the app that permission at install time.

- **Signature** permissions, The system grants these app permissions at install time, but only when the app that attempts to use a permission is signed by the same certificate as the app that defines the permission.

- **Dangerous** permissions cover areas where the app wants data or resources that involve the user's private information, or could potentially affect the user's stored data or the operation of other apps.

  If an app declares that it needs a dangerous permission, the user has to explicitly grant the permission to the app. Until the user approves the permission, app cannot provide functionality that depends on that permission.

1.4 Sensor-based side-channel attacks and covert channels

1.4.1 Motion sensors in mobile devices: accelerometer, gyroscope, magnetometer. Their purposes and applications.

Accelerometer An accelerometer is a device which measures proper acceleration. It means that it takes the gravity acceleration in account so it is the same acceleration that we can feel as a human. There are several accelerometers, some single-axis model and some multi-axis model. The second model allows to measure the magnitude and the direction of the acceleration.

When an accelerometer experiences an acceleration then a mass moves in a certain direction and with a certain magnitude. Some piezoelectric, piezoresistive and capacitive components are able to convert this movement into an electric signal which can then be read by a device. [1]

Applications:

- It can be used to measure acceleration of a vehicle and even vibration of a system.
- It allows to measure some animals behavior when they are out of sight in the wild (how much they are using energy by moving for instance).
- It is used for machinery health monitoring by measuring vibration (turbines, compressors, etc).
- It is used to determine the position that the screen should be on a mobile phone.
- It is used in inertial navigation systems combined with gyroscope.

1.4.2 Gyroscope

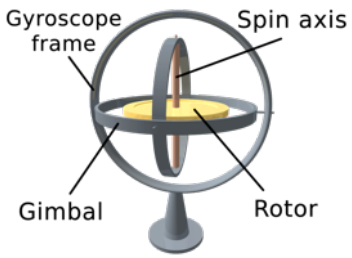
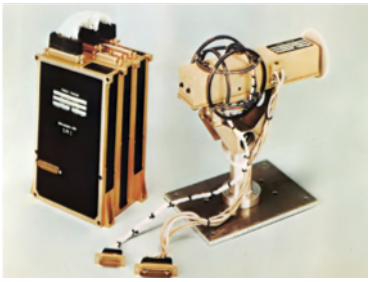


Figure 1.5: Gyroscope

A gyroscope is a device that uses Earth's gravity to help determine orientation and angular velocity.

Figure 1.6: Magnetometer



Its design consists of a freely-rotating disk called a rotor, mounted onto a spinning axis in the center of a larger and more stable wheel. As the axis turns, the rotor remains stationary to indicate the central gravitational pull, and thus which way is "down". [2]

Applications: [19]

- It is used in inertial navigation systems combined with accelerometer (Hubble Telescope for instance).
- It is used to construct gyrocompasses which indicates the "true" north by using the Earth rotation and not the "magnetic" north (in ships or aircrafts for example).
- MEMS gyroscopes are used in mobile phone with accelerometer and MEMS magnetic field sensors for orientation.

1.4.3 Magnetometer

A Magnetometer measures magnetic fields (direction, strength and variation). It allows to know where the "magnetic" north is and thus where we are going to. It outputs an electrical quantity that the device can interpret as a value for the magnetic field. [22]

A Magnetometer can output something relevant thanks to Hall effect, which is a voltage difference across an electrical conductor.

Applications:

- It is used to measure the Earth's magnetic field.
- It is used in inertial navigation systems to give a reference (in aircraft for instance).
- It is used to detect submarines in the military.
- It can detect magnetic metal (ferrous).
- MEMS magnetic field sensors are used in mobile phone.

1.4.4 What is sensor fusion? Why do we need output from multiple sensors to compute device orientation or motion?

Sensor fusion consists of using several information from different sensors to get an information which is more accurate, more complete.

An accelerometer can measure a horizontal or vertical movement (or vibrations). A gyroscope can measure a rotation. A magnetometer can measure magnetic fields and thus allows to know where the north is.

Without one of these sensors, it would be impossible to determine accurately where we are and where we are going to go. Indeed the Earth is moving constantly and we need these sensors to determine our movements relatively to the Earth in order to compute an orientation application on a mobile phone for example.

1.4.5 Sensor-based attacks on smartphones. Provide examples.

When one thinks about the word "attack" in the security context, usually the information leakages first comes into mind. However, with people's growing dependency on mobile devices other forms of attacks such as denial of service or interferences also can pose danger. And the sensors provided inside any modern smartphone open a lot of opportunities for attackers to exploit. We will discuss below which possibilities the sensors open by category of attacks.

Information Leakage Whenever the user types on the keyboard the device will slightly tilt and move when the keys are pressed causing deviations in sensors like accelerometer, gyroscope, magnetometer. This effect can be exploited to learn which of these deviation map to which keys and perform *Keystroke Interference*.

Examples:

- Raphael Spreitzer developed a method named PIN Skimming to use the data from ambient light sensor and RGBW (red, green, blue and white) sensor to extract PIN input of the smartphone. The sensor is activated (permissions are not needed), and the model is trained on several samples providing the user to enter similar to PIN values. As the user moves the fingers the values of the ambient light sensor will change. A trained model can then be used to run in the background and observe what PIN the victim enters. [36]
- Owusu et al. developed an app named *ACCessory* which can identify the area of the touchscreen by analyzing accelerometer data of smart devices. [29]
- Some smartphones provide vibrations to confirm that the user has pressed the key. These vibrations could be accessed by gyroscope, analyzed, which can lead to detecting which keys the user has pressed. [20]

- Asonov et al. proposed an experiment to record the sound of key tapping and infer the correct key from it. [4] Zhuang et al. showed that it is even possible to do even without training data by simply analyzing the acoustic emanation. [43]
- Chhetri et al. introduced a method to reconstruct the design source code sent to a 3-D printer, and later reconstruct it using acoustic covert channel. [9]
- Simon et al. developed a malware named PIN skimmer which uses the front camera of a smartphone and microphone to infer PIN input in a smartphone. The tone would determine the area of the keyboard, and the recording from the camera can help to determine where the key was moved. [34]
- A rumored attack by observing magnetometer when a user uses moves hand with a magnetic field deflecting element (ex. a metal ring)

Location Interference Location detection using acoustic side-channels: secure communication application may produce foreground-audio disseminated which can be detected in a close environment, and the location of the victim may be estimated (ex. in the conference room).

Evesdropping

- Using gyroscope to analyze vibrational noise and eavesdrop on the conversation, instead of a microphone. [27]
- Building acoustic covert channels using gyroscope and accelerometer. [25]

Smartphone Fingerprinting Due to manufacturing process, motion sensors will have deterministic error, which will need to be corrected from device to device to ensure proper operation. After the sensors are calibrated, the resulting calibration matrix can uniquely identify the device. Both Android and iOS devices allow access to calibrated and uncalibrated sensor values (both through the apps and through JavaScript), thus allowing the adversary to calculate a unique fingerprint for the device and later use it for identification. [42]

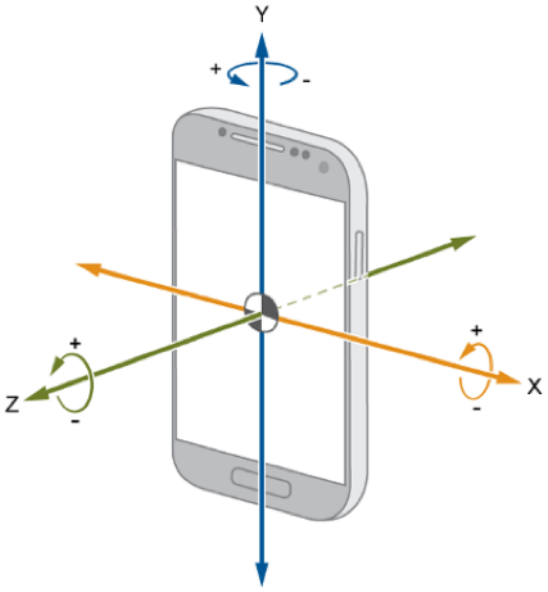
False Sensor Data Injection Since we the people rely on the smartphones, and the smartphones often rely on sensors, by fooling a sensor with values the adversary can possibly control the behavior of the device. [33]

Denial of Service In a similar fashion, if the victim's device is relying on sensor data to perform a task, the adversary may take measure that these sensor reading will not be accurate thus possibly preventing the victim from performing a task. [33]

1.4.6 What is the working principle of the magnetometer sensor? Why does magnetometer need calibration, what are hard-iron and soft-iron distortions? Reaction of magnetometer to electromagnetic noise.

The working principle The magnetometer sensor inside an Android device measures the strength of the surrounding magnetic fields among the three axes (X, Y, Z) in microtesla (µT). The unit "Tesla" indicates the amount of magnetic flux (ex: measured in Weber) per area (ex: square meters). To be precise: $1\,\mathrm{T} = \frac{1\,\mathrm{Wb}}{1\,\mathrm{m}^2}$

Figure 1.7: Magnetometer axes



There are different approaches to measure the magnetic fields. Each of the approaches has its benefits and drawbacks. The most popular is based on the "Hall Effect". A thin plate is introduced which is used to send current. If the magnetic field is present, it will deflect electrons (by Lorenz Force) to a side of the plate. By measuring the voltage on the sides of the plate we can see by how much the electrons were deflected thus how much magnetic field interference we detect. Voltage is then converted into digital signal to indicate the field intensity. [23] The approach repeats on each of the three axes and produces a 1-by-3 vector as a result. This approach is known for lower sensitivity and temperature stability. However, it benefits from a low implementation price, and the fact that it does not produce a lot of electromagnetic noise from its components. The tendency to use magnetometers in Android devices primarily for detection of Earth's magnetic field (i.e. the implementation of eCompass) may justify why this method is the most currently used one. [6]

Other approached that could be found in mobile devices are Giant Magnetoresistance (GMR) sensor, which observes the spin states of the electors provided through the quan-

tum nature. Another approach attempts to build magnetic tunneling junction (MTJ) utilizing a quantum mechanical phenomenon of tunnel magnetoresistance. Anisotropic magnetoresistance (AMR) approach utilizes materials that change resistance with introduced magnetic fields. Another competitive approach tends to work on using only Lorenz force by introducing a micro bar magnet and measuring the deflection. [6]

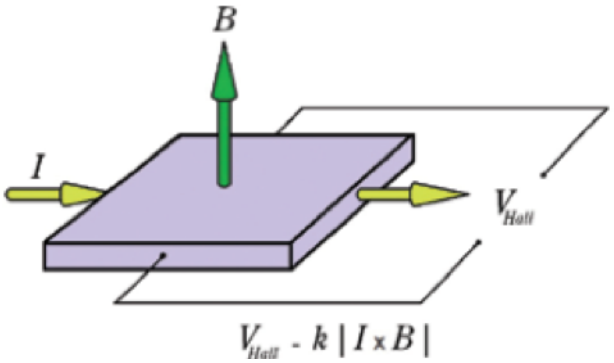
Why the magnetometer needs to be calibrated, what are hard-iron and soft-iron distortions? Since the magnetometer is first produced as a standalone sensor, and then integrated into a smartphone, it will immediately be introduced to additional electromagnetic fields that will distort the pre-set characteristics. We should remember, that along with the Earth's magnetic field there will be other interfering magnetic fields that we will have to account for, in order to read the right measurements. The sources for these interfering magnetic fields (distortions) come from two categories: hard iron and soft iron.

Hard iron distortions are the permanently interfering magnetic fields coming from objects that produce these fields (for example, an electromagnet in the speaker of the phone). Such distortions are always additive to the to the earth's magnetic field and therefore could be mitigated by simply calculating and subtracting the offset from the measurements.

Soft iron distortions come from the materials that distort the magnet field, but do not produce additional field. Example could be the metal chases of the phone. Such distortions are more computationally expensive to account for. The approaches to mitigate this distortion may include 3x3 matrix transformation or scale biases [41] which first calculates the averages for each axes, and one average for all, then normalizes each axis average, and uses it as a factor for correcting the readings.

Reaction of magnetometer to electromagnetic noise There are many sources of electromagnetic noise in daily lives. In fact, any device carrying current will produce mag-

Figure 1.8: The Hall Effect approach



netic waves and thus can distort the magnetometer. The approaches discussed in the previous paragraphs suggest some ways of dealing with expected electromagnetic noise. However, anything that is not observed in the "normal" environment of the sensor, when it is calibrated likely to distort it. For example, one could notice significant sensor deviation by bringing it close to the running CPU of the computer. How to increase the resistance of magnetomoters to the electromagnetic noise while maintaining their sensitivity has been an ongoing research question. Patonis et al. introduced a way of fusing magnetometer with other sensors like accelerometer and gyroscope to reduce the noise when used in mobile devices. [30] However, currently magnetometers are still very sensitive to the electromagnetic interferences, which could serve as an additional opportunity for adversaries to carry out attacks on the smartphones.

2 Proof of Concept

Our work consists of two parts : A transmitter which transmits series of data (*payload*) and a receiver which is used to decode these bits of data. This communication between the transmitter and the receiver should be as discrete as possible in order not to allow anyone to detect it. Then it is relevant to use magnetic field created by the CPU to send this payload.

Transmitter implements the concept to raise the CPU Utilization close to 100% when payload bit is a '1', raising CPU to 100% is achieved using *busy loops*. Similarly, for payload bit '0' we add sleep to achieve 0% CPU utilization.

Figure 2.1 shows transmitter feature allowing to raise the CPU load up to 100%. In order to reach this 100% value for the CPU, it is necessary to use all busy threads as there are available logical cores while the application is running.

However, it should also be possible to configure the *payload size* (number of bits transmitted), *bitrate* in bps(bits per second), *number of threads or CPU cores* and *preamble* specially to evaluate the implementation better.

2.1 Setup the development environment (Android Studio), enable USB debugging on a smartphone, learn how to build and run Android applications on a device

Upon the start of the this phase, each one of us open, configured, and ran a sample project in the Android Studio. We also ensured that we can easily push and pull the code from different branch without issues. We learned the initial building blocks of Android development such UI controls, activity life cycle, and got familiar with Java

Figure 2.1: Illustration of an approach to generate CPU loads in Javascript

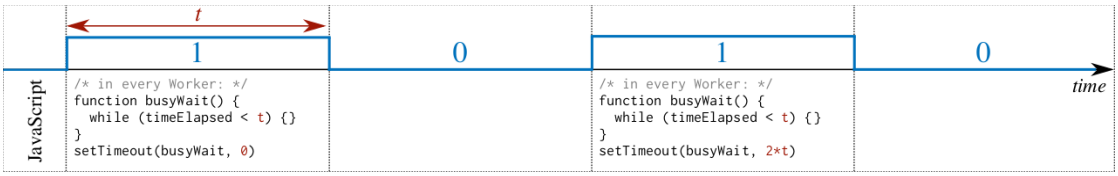
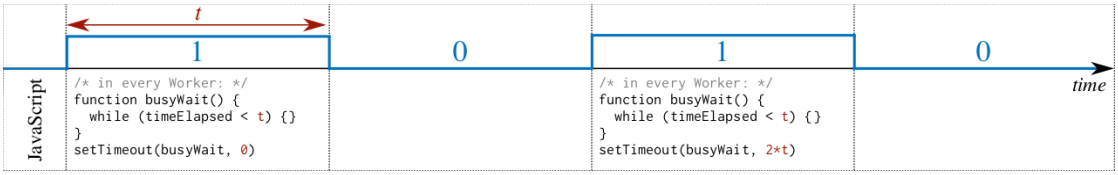


Figure 2.2: Illustration of an approach to generate CPU loads in Javascript



and OOP(Object Oriented Programming). The main challenges we faced was during integration of version control system in Android Studio IDE.

2.2 Implement the recording of sensors. The Android application should be able to constantly record magnetometer measurements in the background and save them to a file on the phone.

2.2.1 Solution

We implemented this task as our initial (version 1) *SensorRecorder* app. We used foreground service, to read the magnetometer, and used *FileOutputStream* to write the values to file. Initially, we only saved X, Y, Z values, but we later decided to add the sensor reading timestamps to be available for processes in the upcoming tasks.

2.2.2 Problems

We did not encounter any problems with this task. Since we built it from scratch, we did not need to refactor the code.

2.3 Implement the CPU signal generator. The Android application should be able to generate up to 100% CPU activity by running so-called busy loops in multiple threads. Run the signal generator and observe the resulting peaks in the sensor recordings.

2.3.1 Solution

Transmitter application contains a class called *Busy*. It implements busy loops in threads. First it stores the number of logical cores that the application is going to use

while running. It creates also threads based on another class for the CPU : *CPUThread*. This second class is necessary, for the application to start the threads. it runs the *run()* function of this second class.

2.3.2 Problems

First, the threads created allowed the signal to increase CPU load up to 100% but when no signal was sent, the CPU load didn't get back close to 0% instantly. Moreover, when we tried to raise the CPU load at 100% for the second time then either nothing happened or the application crashed.

We corrected this by understanding how Thread's life cycle works and how it can be controlled. In more details, instead of trying to kill threads which is done automatically by the java garbage collector, we use the method *sleep* when a '0' bit is sent and we run a busyloop which is checking whether the time spent is greater than what we expect as per bitrate.

2.4 Implement the transmitter based on the signal generator. Implement user input of the payload (e.g. "1011010") and the bitrate (bit/s). On request, the transmitter encodes the payload into CPU loads, by generating 100% CPU activity to encode 1 and by passively waiting to encode 0. Prepend the signal with a preamble to allow the detection of the signal.

2.4.1 Solution

In the *CPUThread* class are stored both the message to send to the receiver and a delay. This delay is related to the bitrate the user wants to use to send the payload, it is the time threads should wait before sending the next bit of the message.

The data sent from the transmitter app is done by using the Busy Class which creates the threads and starts it with the bit rate, message, number of threads to be used.

The message is the final data to be sent. It is in the frame format, as illustrated by *Table 2.1*

Preamble	**Payload**
001101	10101010

Table 2.1: Frame format

The data of the frame is of variable length, but, it is fixed before starting the transmission. We are using the same data length for full transmission once the transmitter is sending a message.

The user interface is really easy to understand. Basically, on this interface a user is able to configure some variables such as : a preamble, a bitrate, number of threads used, the payload, etc.

This part is dealing with how to organize the screen to be easy and clear to use. The user provides some inputs through this interface and then the application uses them in a suitable way. Thereby, busy threads are created with all the right values.

An important part of work was about the synchronization between the transmitter and the receiver. Indeed it is really important that the receiver is able to detect the message when the transmitter is sending it. To do so, the message was not sent alone but also with a preamble in order to tell the receiver that a signal is going to arrive. When the preamble is read by the receiver then it knows the signal comes just after. So the transmitter is first sending this preamble (again a message of 1's and 0's bits) by running the busy threads as it would do for the payload.

This preamble is a way to make both transmitter and receiver synchronized to communicate efficiently.

2.4.2 Problems

The First method we tried for sending data involved creating Busy class object, thus, threads on '1' bit while parsing the payload and waiting in the main UI thread for bit '0'. This caused the CPU usage to remain high and in some cases not exit.

New strategy was designed to solve this issue which involved creating objects once and threads as *HIGH* bits came, still, the parsing of the payload was done at the UI end. On *LOW* bits the main UI thread was sleeping for required time as that of bit rate used. This required the use of synchronized/volatile variables. It resolved the high CPU load even on *LOW* bits.

This method had unexpected side effect of **low** *rise* and *fall* times for the CPU loading. To counteract this and make the transmitting of data as fast as possible we had to dive a bit deeper into threads and parallel processing. Finally, we arrived on the method to solve this issue by passing the frame to the threads where the parsed it and ran at full load or went to sleep for the bit rate. To make this process as fast as possible we moved from using Lists for number of threads needed to arrays which have faster access times, this further improved the roaming of CPU load. Since this preamble is also using some 0's and 1's bit, the first issue which happened was to manage the fact that the preamble could also appear in the payload making the synchronization much harder.

This problem was solved on the receiver side.

Furthermore, it was also better to create a time synchronization between the receiver and the transmitter. In fact, the receiver could start to measure a signal after the beginning of the preamble and then never detect any preamble. So the message was completely lost.

This issue was solved on both side. For the transmitter, it consisted of waiting the minute after the button "send payload" was clicked so that we had a precise time to send the whole message with preamble. This is not necessary anymore (but quite relevant) since a solution was also found on the receiver part.

2.5 Implement the receiver based on the sensor recorder. First, the receiver convert 3-axis magnetometer measurements into a one-dimensional trace. Then, the receiver detects the preamble signal using the cross-correlation. Afterwards, it decodes the payload by analyzing the disturbance, and shows the result.

2.5.1 Solution

Receiver is broken down into multiple steps, first sensor data from recorder is fed into the *SensorWindow* object. *SensorWindow* handles storage of data, and notifies when enough data is collected for the decoding. When the window is full, it is passed to PCA (Principal Component Analysis) which projects the data into single principal component with largest variance using linear transformation. This terminology is also called as Dimensionality Reduction. Algorithm is explained further in detail. Secondly, Decoding function is used to regenerate the bits send from transmitter application. As part of decoding function. First preamble is detected using cross-correlation method and further signal is decoded by taking mean of averages for logical 0 and logical 1.

Dimesionality Reduction
Magnetometer sensor gives sensor values along X, Y and Z axis. But some sensor values recorded by the magnetometer are not relevent, therefore Dimensionality Reduction is applied on X,Y and Z axis data. To achieve this we are using PCA (Principal Component Analysis) provided by SMILE (Statistical Machine Intelligence and Learning Engine) library for applying dimensionality reduction on the sensor data.

PCA Algorithm
1. First we perform mean-normalization of the datasets – transform the dataset such that the mean values of each axis becomes zero.
2. Then compute the co-variance or correlation matrix.
3. After that perform EVD Eigenvelue Deomposition or SVD Singular value decomposition of the covariance matrix from step 2.

4. The result is a set of principal components each of which explains a part of the variance.
5. The principal components are typically ordered such that the first components explain most of the variance, and last components explain very little of it.
6. In the last step, we discard components with the least variance, and keep only first principal component with the largest variance.To select the number of componenets to keep we typically use the cumulated ratio of explained variance to the total variance.
7. We use the components to compress the original dataset by performing the projection of the original data on the basis formed by these components. Basis vectors are internally calculated using Eigenvectors concept explained in foundations section.
8. After doing these steps, we have a dataaset with smaller number of features, but most of the information of the original dataset is preserved.

In SMILE, PCA class already performs mean-normalization, then computes the covariance matrix and automatically decides whether to use EVD or SVD. All we need to do is to give it a data matrix.

Typically PCA is performed on the co-variance matrix but sometimes when some of the original features are on a different scale, the ratio of the explained variance can become misleading.

For example: if one of the feature we have is distance in kilometers and the other one is time in milliseconds, then the second feature will have larger variance simply because the numbers are a lot higher in the second feature. Thus this feature will have the dominant presence in the resulting components.

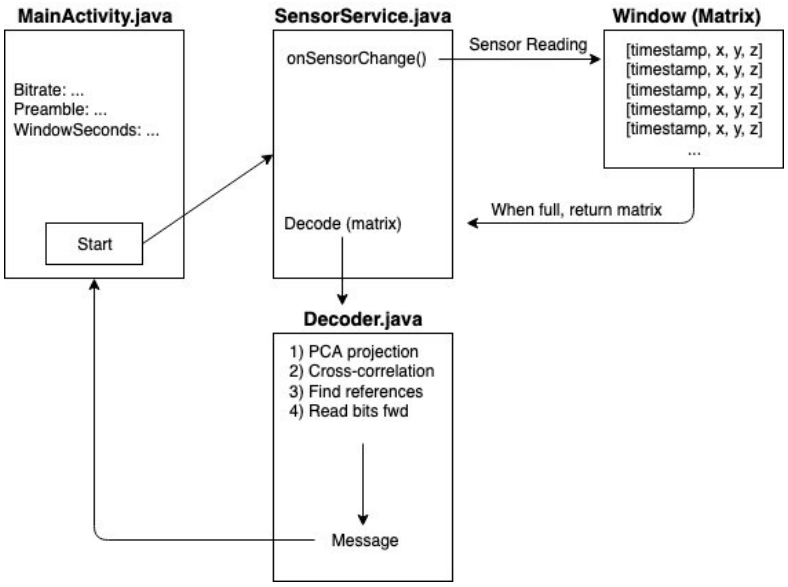
To overcome this problem we can use the correlation matrix instead of the covariance matrix. Since the correlation coeffcient is unitless, the PCA results will not be affected by different scales.

The Figure2.5.1 described the main components of the Receiver app as the foreground SensorSerive, the Window (matrix), and the Decoders, along with the key steps for each of the components.

Decoder

1. Take in matrix of sensor values S
2. Project the values to one dimension using PCA: $S_{projected} = PCA(S)$
3. Read the preamble signature file values: P
4. Project the preamble signature values using PCA: $P_{projected} = PCA(P)$
5. Cross-correlate $S_{projected}$ with $P_{projected}$:

Figure 2.3: The diagram of components of the Receiver



a) Take part of the signal of length l, at delay d, where $l = |P_{projected}|$ and $|P_{projected}| <= d <= |S|$

b) Signal slice $S'_{projected}$ is now the same length, as the preamble signature $P_{projected}$. The correlation score is computer as the dot-product between the vectors $Score = S'_{projected} \cdot P_{projected}$

c) Repeat for each delay d, and record the max score, and the corresponding delay d_{best}

d) Return $d_{best} + |P_{projected}|$ as the delay at which the signal starts

6. Estimate the number of nanoseconds t_n are used for the transmission of 1 bit
7. Read back t_n to get the representation of "1" (since the last bit of the preamble is "1")
8. Read back t_n once again to get the representation of "0" (since the second last bit of the preamble is "0")
9. Return to the signal start index, and start reading bit-by-bit forward with interval of t_n and compare the representations to the references of 1 and 0. Note: we just average the vector values for comparison, and calculate whether the average of a new bit is closer to the average of 1 or the average of 0. This way we decode the bits in the signal.

2.5.2 Problems

Once we implemented the cross-correlation function, we noticed that if the signal contains bits that are same as the preamble, the cross-correlation return a higher score in for the bits in the signal. We speculated that the reason for that could be, that by controlling the CPU activity we are slightly reducing the noise when the message is being transmitted, and less noise results in higher correlation score. We, however, were interested to find the first "good enough" preamble correlation. To implement this, we first find all the peaks for the correlation scores. We define a pick a point, if its preceding and the following point is less than the point itself. Once the peaks are identified, we introduce a threshold of 75%. The threshold is used as a measure to consider the next peak or not. If the next peak value is 25% better than the previous peak, we consider the new peak as the best peak. This approach helps us to neglect the slightly better peaks that result in the signal transmission (compared to the first preamble instance).

3 Evaluation

3.1 Automatize the PoC to transmit multiple short messages (e.g., 20 random 20-bit messages). Compute the average Bit Error Rate (BER) of such transmission.

3.1.1 Solution

The algorithm for transmitter automation to 20 messages of 20 bits was implemented by simply repeating the transmission process the necessary amount of times. On the receiver side however, we had to develop a mechanism with the following requirements:

1. capable of continuously receive messages until ”stop” command is issued
2. the decoding processes must not affect the recording process (in terms on EMF emissions)
3. the message length, preamble, and bitrate would be provided before the service starts

To accomplish this, we had the following two strategies in mind:

1. create a continuous flow of sensor data, and re-calculate the cross correlation with the preamble signature every time the new sensor reading is introduced.
2. create a ”window juggler” which records one window of the specified length, and passes it down to decoding.

We decided that the second approach would be more suitable, given the requirements above. Although in Approach 1, we could implement a very efficient way to calculate the cross-correlation (for example, by keeping the sum of dot-products in the memory and for each new sensor reading simply add a new dot-product to the sum, and remove the first one), this approach would still require new computations every 1/100th of a second and would rely heavily on preamble threshold, instead of finding the best match for preamble in the signal. To implement the second approach, we re-wrote our original implementation of *SensorWindow*, to *SimpleSensorWindow*. The initial implementation used *ArrayList* with estimated size to record the sensor readings. The new implementation aims to reduce the run time complexity by switching from the *ArrayList* to simple *Array*. We estimate the size for the arrays, and create three empty arrays in the memory before the service starts. We refer to them as *Window 1*, *Window*

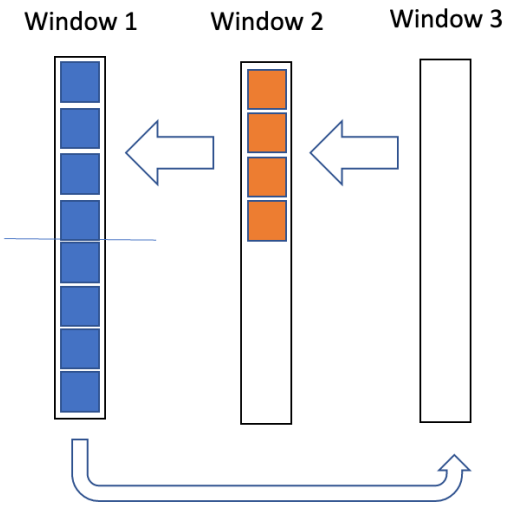


Figure 3.1: Continuous recording and decoding of messages

2, *Window 3*. The size for the windows L_w is estimated from the signal length L_s. In particular, we set the window size to be double the signal length $L_w = 2 \times L_s$, and overlap the windows by 50%. This approach guarantees us that a full signal will be observed in one window in one of the two windows recorded. We are running the risk of duplicating the signal in the event when the signals are sent without interval. To compensate for this, we simply introduce a rule that there should be some delay between signal transmission, which will result, that the transmission time of two continuous signals will always be larger than L_w. The following algorithm is used for recording:

1. A new sensor value s is added to *Window 1*
2. If the number of elements in *Window 1* is larger than half of it's size, then also add the same value to *Window 2*
3. If *Window 1* is full, swap windows:
   a) $Window_3 = Window_1$
   b) $Window_1 = Window_2$
   c) Clear *Window 2*
   d) Send *Window 3* for decoding in the background using *AsyncTask*

The extension of the decoding algorithm:

1. Cross-correlate preamble signature and the signal in the window (as before), however, this time we are considering the minimum correlation score, specified under

preamblePickMin, because we are not sure if the signal in the window has preamble. Lowering this threshold introduces more false positives, but by increasing it, we are also increasing the risk that the preamble will not be detected (since the presence of noise in the channel will lower the correlation scores).

2. Once the preamble is detected via cross-correlation, we receive the signal starting index back. For continuous recording, we are accounting for the situation when the signal transmission occurs in the second part of the window. This leads to the signal being cut off. To solve this, we introduce a function *fullSignalPresent* which takes in the recorded window, the signal start time, and estimated number of nanoseconds per bit transmitted. This function then evaluates whether the the time from the signal start index is large enough to fit the whole signal. If it's not, that means that the signal was chopped off, and we will observe in the next window.
3. If the full signal is present in the window, we decode it with the same algorithm we used previously.

3.1.2 Problems

Although automating the receiver to continuously receive and decode messaged proved successful on shorter messages (ex. 8bit) with lower bitrate (ex. 1bps) we noticed that higher BER as the number of bits in a message and the bitrate grew. We speculate that this discrepancy could have been introduces with two factors. First, since we have to specify the min and max coefficients for preamble cross-correlation. Depending on the scenario, the environment in which the device is located, and status of internal processes the noise could step out of prescribed thresholds, thus skipping a message. If one message is skipped in the queue, the BER will be affected, as it was not clear which of the message was skipped, and which ones were not detected correctly. Secondly, we assume that introducing additional computation online reduces the efficiency of the covert channel. In one of our experiments, we were able to raise SNR for by 42% by first storing sensor values in memory, and only writing them to the file *after* the signal was fully received.

Therefore, in order to capture the maximum potential of the magnetic covert channel we will use the most lean version of the Receiver, but running it in the manual one-message mode.

3.2 Compute the Signal-to-Noise (SNR) ratio of the covert signal. Evaluate the SNR depending on the CPU load by adjusting the number of threads in the busy loop.

3.2.1 Solution

In order to computer the SNR of the covert channel, we send a repeating message of "10" x 5 times. This allows us to measure the SNR for each of the five segments, and compute the average the score. Both, the Transmitter and the Receiver apps, have dedicated SNR screens. We decided to keep SNR activities apart, to keep the important assumptions constant between the two apps. In particular, we agree on a constant message, bitrate and start time. We refer to start time as *Time Sync* in our apps, because it allows us to synchronize the receiver and the transmitter without the use of preambles.

To measure the SNR, we use the following procedure:

1. The user navigate to SNR screens in both apps.
2. The user select which transmission method to evaluate (on the Transmitter side).
3. The user selects the transmission method and options (on the Receiver side). Note: the Receiver app computes the SNR from all the signal methods in the same way, buy simply analyzing the magnetometer deflections. The options selected on the receiver side, will simply be used to label the SNR score and save the sensor values in the descriptive file name, which could be easily found later.
4. Receiver must be started first, because it uses the background service, which the transmitter will hold the UI thread.
5. Once started, the receiver will start observing the magnetometer values at the beginning of the new minute and record them.
6. Once the exacted number of values is recorded, we assume that the signal was transferred in full, we begin the SNR computation.

SNR Calculation

1. Compute the number of elements per segments: $NumberofElementsperWindow = \frac{Length_{Signal}}{Numberof"10"signalrepeats}$ Use this number to define borders of each segment.
2. Split the segments in half. The first half will be magnetometer values at "high", and the second half - the magnetometer values at "low".
3. For each of the two parts of the segment, remove the first third of values, and the last third. This allows us to avoid the transition noise.
4. Find the distance between the average of "high" and the average of "low" values.

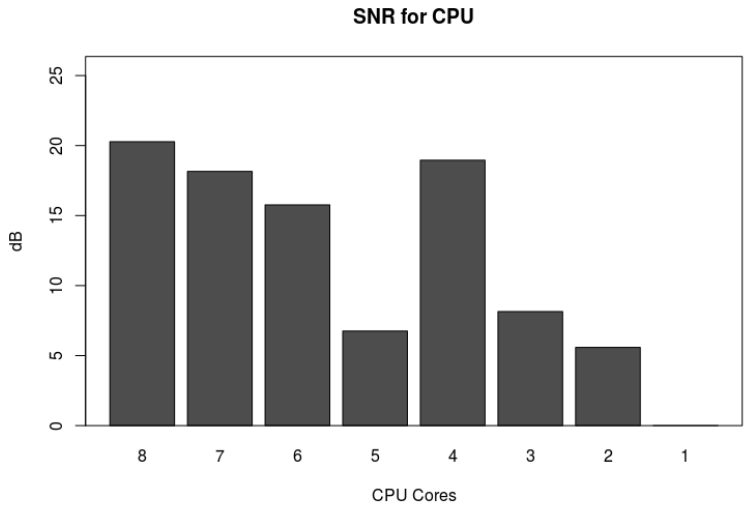


Figure 3.2: Comparison of SNR scores for different number of threads.

5. Divide this distance by standard deviation of noise values. Note: the noise was prerecorded for 5 minutes before the test. $SNR = \frac{CPU_{100} - CPU_0}{\sigma(CPU_{noise})}$
6. Repeat these Step 2 through Step 5 for each of the five segments of the signal.
7. Find the average of SNR scores for the signal.

From Figure 3.2.1, the highest mean SNR we observed was 21.08dB, with the use of 8 logical threads. The chart above represents that the SNR generally decreases as the number of threads reduces. We were expecting these results, as we believed that less cores will produce less electromagnetic fields, and thus deflect the magnetometer less. Although most of the values follow this pattern, the experiment with 4 and 5 threads yielded surprising results. We were not expecting such poor performance in terms of SNR for 5 threads scenario, and excellent performance for 4 threads. We repeated the tests several times (thus measuring SNR over 15 times across the segments), and the results were very similar. We speculate that this could be caused by the specific characteristics of Android Schedule. Throughout the code, we were not capable of sending tasks directly to physical threads. Instead, we open virtual threads, and Android distributes them across available physical CPU cores.

3.2.2 Problems

The biggest challenge we had for SNR tests was the synchronization. We implemented time synchronization mechanism which instantiates a *Calendar* class and checks how many seconds are left until the next minute. We then use this number with *Handler.postDelayed()* function to register the *SensorListener* exactly when the new minute

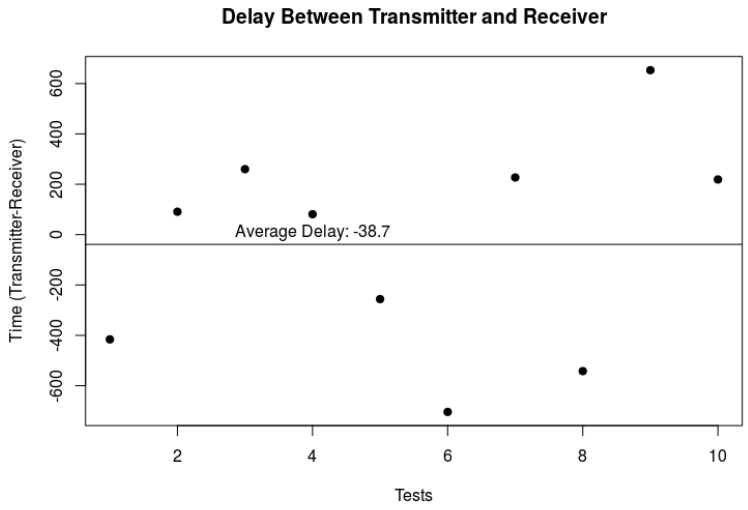


Figure 3.3: Delay between Transmitter starting time, and Receiver starting time in milliseconds.

begins. Once the *SensorListener* is registered, we begin to receive the reading from the sensor, and store them accordingly for further processing. Developing this solution in theory, we didn't consider the possible delays resulting from other tasks with higher priority. When we started troubleshooting why the SNR for the channel was lower than expected, we started logging system time in milliseconds, to observe when Transmitter and the Receiver start the SNR procedure.

Ideally, we expected them to be just few milliseconds apart. However, the Figure 3.3 summarizes , that the variance in delay is relatively large. Due to this large variance, we would not be able to fix this issue with simply hard-coding the adding delay. To prevent this from affecting SNR measurements, we decided to reduce the bitrate, and transmit each bit over three second period. This allows us one second margin, making it possible to observe the signal, even if the delay between the starting times reaches one full second. We, however, are not using the fill 3 seconds of "high"/"low" values. Instead, we are slicing the the first third, and the last third to avoid incorrect values that could have resulted from delay between the apps or from the transition between "high"/"low" values. Additionally, this approach helps us to synchronise between two separate devices, when we also have to account for the difference in system time.

3.3 Evaluate the inter-device transmission, by running transmitter and receiver on two different smartphones. Evaluate the SNR depending on a distance between smartphones.

3.3.1 Solution

For this test, we used *Samsung Galaxy Note Edge* (year 2014 model) with 4-cores CPU as the transmitter. And we were able to successfully receive the signal on the Pixel 3 when the phones were right next to each other. We were not able to transmit the signal across the devices when they were more than $5cm$ apart. Besides the surrounding electromagnetic noise, we assumed that the reason could be that 4 CPU cores produce less electromagnetic emission, thus make it less noticeable when used to transmit across further distance.

For the SNR tests we used*Huawei P20 Lite* as a transmitting device. Attempting to perform the tests, we discovered that our interdevice SNR tests result in negative values. This means that the variance of noise is higher than the distance between the average high and low signal points. We speculate this was the result due to various applications present and running on the transmitter device at all times, which may keep the CPU utilization up, which in return will produce magnetic field, and therefore will vary less from the true signal generated by spinning CPU to 100%.

3.3.2 Problems

Since our transmitter and receiver apps were relatively decoupled from each other, we did not face any issues launching this test on two separate devices. However, synchronizing the devices for SNR tests, was challenging. As mentioned previously, along with common system delay, we had to account for regular inconsistency of time between two physical devices. Even though the devices were connected to the same WiFi network, we could visually observe the time difference in just under a second. The inter-device transmission and decoding test left us satisfied that the actual transmission was possible. Although we did not observe any meaningful results with distance above $5cm$, the fact of possibility presents many way for attackers to exploit this fact. We speculate that a small $5cm$ transmission distance could be the result of low CPU core count, and a *few* background apps running on the transmitter device. We predict that a device with limited amount of apps running in the background, but with a higher CPU core count could perform inter-device transmission over further distances.

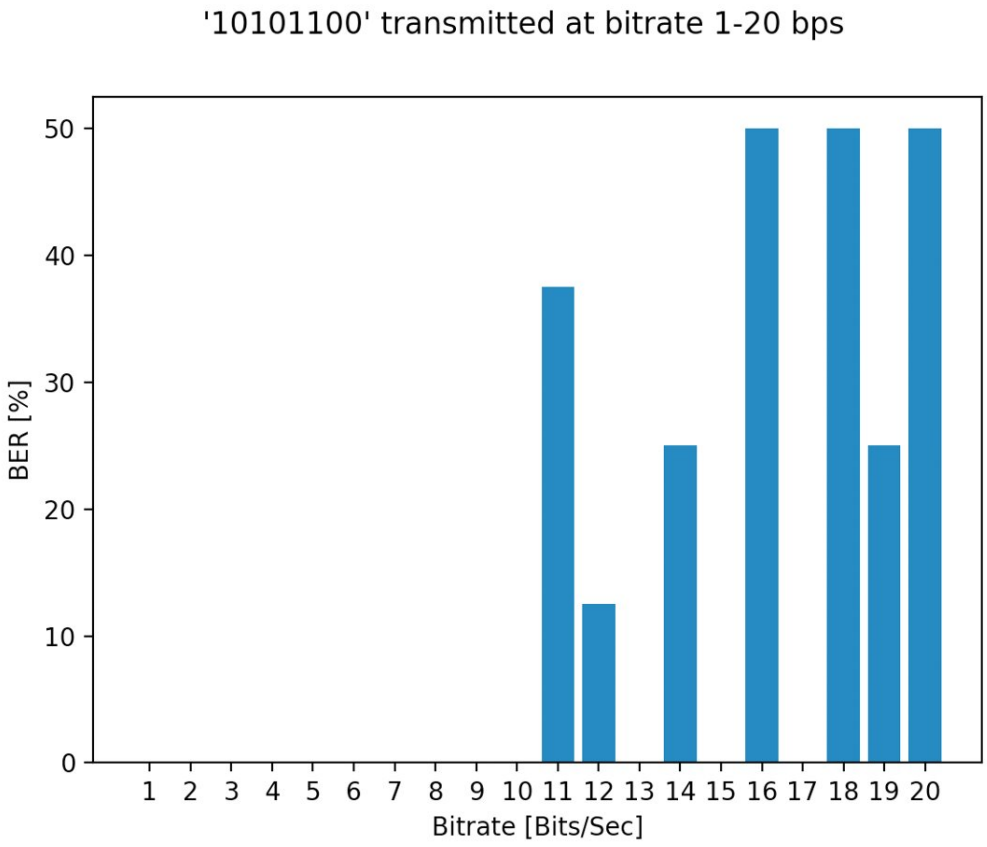


Figure 3.4: BER Plot

3.4 Identify the highest possible bitrate (== the shortest CPU load which is still observable in the sensor measurements). Evaluate the BER depending on the bitrate.

3.4.1 Solution

In order to minimize the background noise, and improve our chances or reaching higher transmission bitrates we decided to utilize the leanest versions of our apps to reduce the electromagnetic noise which could be caused by extra IO operations, extra threads running in the background and etc. For this particular test we used this commit with $hash$: $7c8c3299c8d5038e1b55b81ea470d84b3cefe4e4$ for the Transmitter and the Receiver apps. We were able to achieve 0% BER up to 17 bits per second for the transmission of 8 bit message "10101100". We then performed further test with several random messages 20 bits of length, which confirmed that up to 7 bps we can communicate on 0% BER. We expected the BER to go up with bitrate. Working with Pixel 3, we observed

that the fastest magnetometer update was producing approximately 100 readings per second. Hypothetically speaking, this also tells us that maximum possible bitrate for this magnetic covert channel on this device could be 100bps. However, this is a *perfect world* scenario, in which each magnetometer sensor reading is not affected by anything except the signal that we want to transmit. As we learned from the theoretical task that is not possible. Just considering one of the main ideas of the magnetometer - to be able to observe Earth electromagnetic fields, we already know that there will be constant noise in the covert channel constructed on the bases of magnetometer readings. There are many electromagnetic fields that are constantly introducing noise into the magnetometer. Additionally, having less values to express the preamble signature, makes the preamble signatures less distinct, which reduces reduces our chances to correctly find the preamble in the signal using cross-correlation. Our practical experiments showed that we it's possible to achieve 17bps transmission on our implementation and on Pixel 3. Possibly more efficient coding, reduction of noise, and the selection of the device based could increase this maximum bitrate significantly.

3.5 Evaluate the signal detection using the synchronization sequence as the time shift between actual and detected time point. Evaluate multiple preamble sequences with different bitrates. Present the histogram for each sequence.

3.5.1 Solution

Through this exercise we learned that not all sequences are equally good to be used for synchronization. We started our Transmitter-Receiver tests with "1010" preamble. Although it was challenging to pin point what exactly was wrong with out receiver, we soon understood, that a receiver often *misses* the preamble in the signal. We started looking for other, more unique preambles, and ended up with "001101". In this exercise we created a time shift calculator for prerecorded preambles. In our Receiver app, the user is capable of recording a preamble signature, and then test it in the time shift calculator. A signal will be generated with this preamble in the middle. The cross-correlation will be applied. And Time shift will be calculated as a different between the detected, and the actual index where the preamble starts. The actual index is determined by saving the system time when the preamble was played. And the detected index is determined by the results of cross-correlation. Interestingly, some preambles are detected better than others. Since are cross-correlating the preamble signature with its actual appearance in the signal, we are practically cross-correlating the preamble with itself. Such characteristic is know as *autocorrelation*. There happen to be a set of sequences that are characterized by high autocorrelation performance. These sequences are know as Barker codes. We were pleased to know, that our second preamble that we intuitively

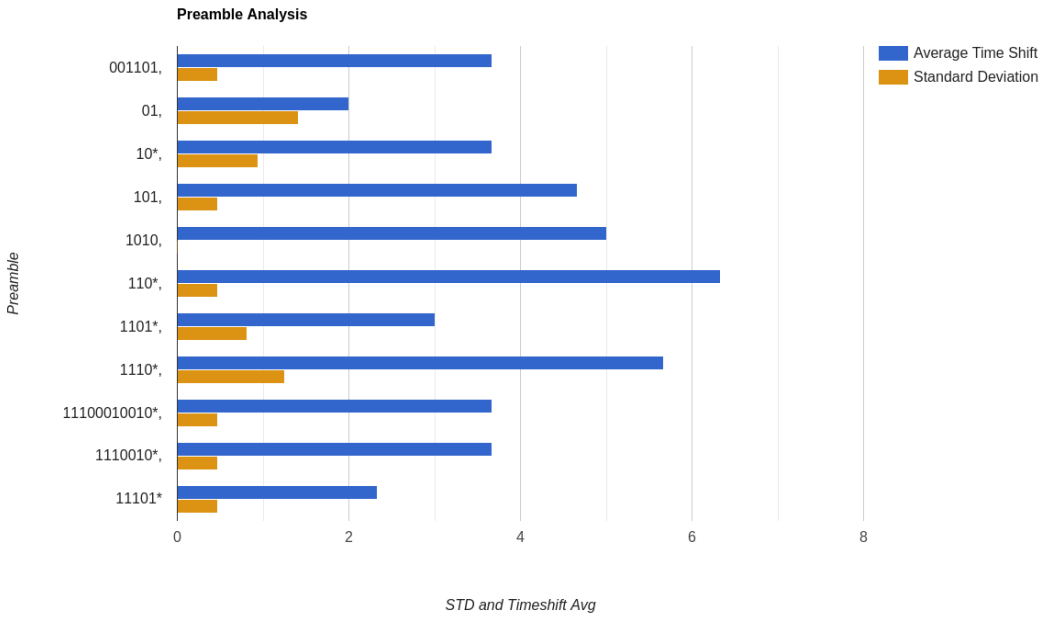


Figure 3.5: Preamble Analysis

picked ”001101”, after numerous trial and errors with decoding happened to be very close to a Barker synchronisation sequence ”1101”. We then tested the preambles that we have used, and the *Barker codes* in order to see whether there is really a difference in detecting a preamble in the signal. As we can observe from the graph above most of the Barker codes have relatively little, and very importantly more or less constant time shift. We were satisfied with the performance of ”1101”, and according to the results of this experiment, this was one of the most suitable preambles for our use case.

4 Alternative signal sources

In the proof of concept we used CPU to manipulate magnetometer values as a method for creating covert channel which was very successful, but it makes it obvious as to something is wrong as the performance of the device may suffer due intentional loading of the CPU.

Alternative methods of manipulating the magnetic fields was tried to find which method was most effective in getting the best results while being most inconspicuous. Some of the methods are:

4.1 Device screen

4.1.1 Solution

Main idea with device is screen is that while in use different applications do manipulate the screen in some manner which offers us a method which may go unnoticed by the user as to some thing malicious is occurring. We tried two methods of manipulating the screen.

Screen Brightness

The earlier versions of android allowed for changing of screen brightness without any special permissions, but current versions requires special permission to manipulate screen brightness.

The most effective manner with which we can generate changes in magnetic fields is to change from zero brightness to full brightness. This changes the screen brightness gradually due to animations active in the android by default. This method produces very noisy signal and is not decode able. Disabling the animations allowed for smoother transition between brightness levels. The signal produced is very small, thus, the original message is not recoverable.

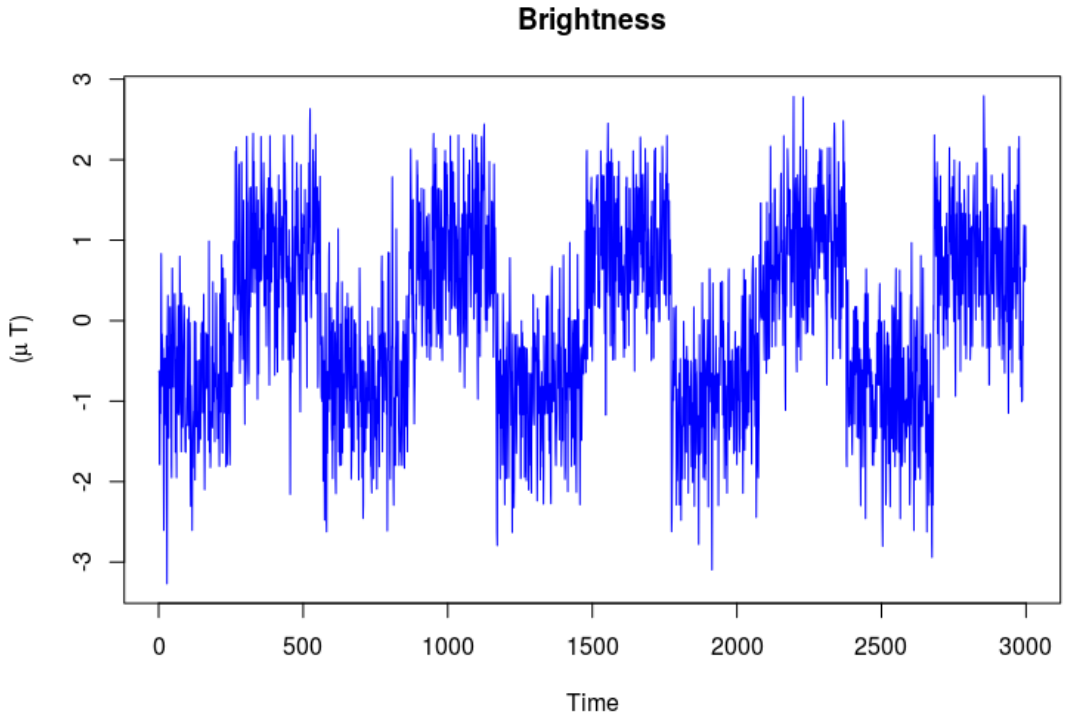


Figure 4.1: Received Signal for Screen Brightness

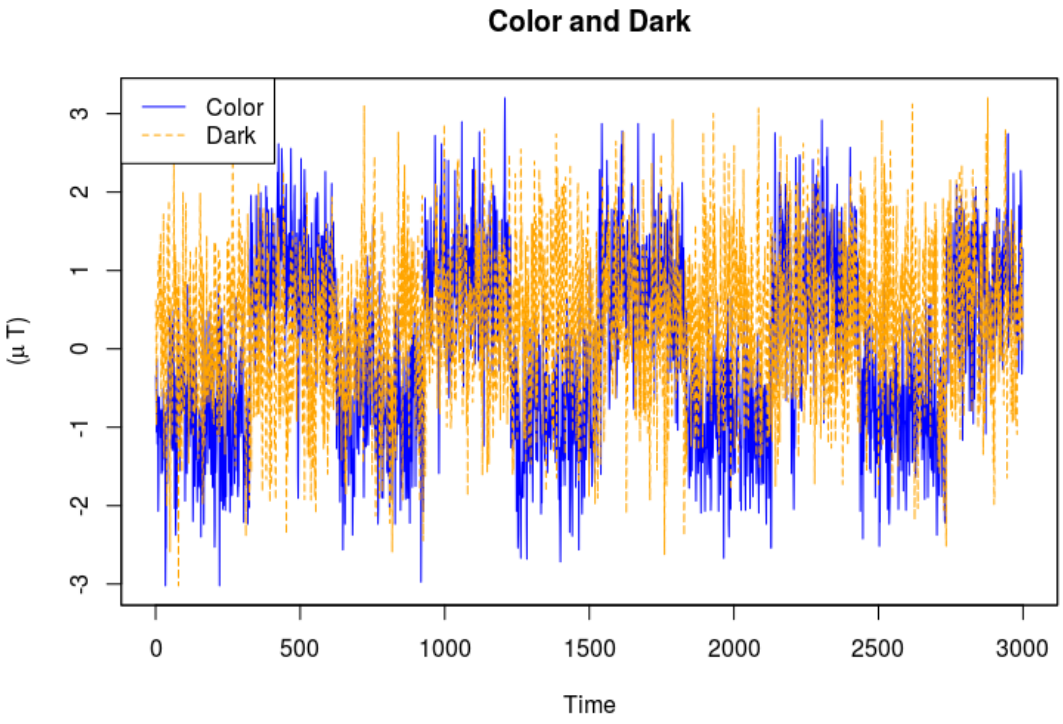


Figure 4.2: Received Signal for Color Change

Color Change

Changing the background color of the application does not require any special permissions, thus it provided a more discrete method for manipulation of magnetic waves generated by screen to create a covert channel.

The method tried for this is to work with extremes and thus produce as different reaction in magnetometer as possible. This was implemented by changing the colour of the background between BLACK and WHITE for bits '0' and '1' respectively.

4.1.2 Problems

Both methods have their own problems with implementing in a practical applications.

- Screen Brightness
    1. Latest android does not allow change of brightness with special permissions, limiting practical application.

2. The change in brightness needs to be drastic to detect message (e.g 0 to 100 and vice versa.
3. Android has a lot of animations related to various activities including brightness change, it makes signal noisy

- Color Change
  1. The screen brightness needs to be near 100% for it to affect magnetometer
  2. The change needs to be covering near full screen to affect the magnetometer, which is very noticeable.
  3. The change needs to be drastic (e.g. black to white and vice versa) for any affect
  4. It does not affect the result very much

4.2 Speakers

Speakers are moving electromagnets which generate a changing magnetic field along with sound which produced due to movement of the diaphragm. These magnetic changes affect the devices which are very nearby devices sensitive magnetic field changes.

4.2.1 Solution

Main idea with this transmission method is to generate a tone when transmitting logical '1' and stop the tone when send '0'. There is special class called 'ToneGenerator' in android which has all the basic tones used in normal mobile phone. for example, tone for dialing a number, tone for connecting a call etc. There is also 'Max_Volume' and 'Min_Volume' constants for 100 and 0 volume level.
Hence using above mentioned features we have developed a transmitter using speaker which starts playing the tone with maximum volume when it is supposed to transmit '1' and otherwise it stops the tone for transmitting '0' There are hundreds of different tones available in android and choosing the one which is good enough to generate magnetic field is rather hard. So we have tried couple of tones with loud and finally we have used 'TONE_DTMF_0' with 1336Hz, 941Hz frequencies.

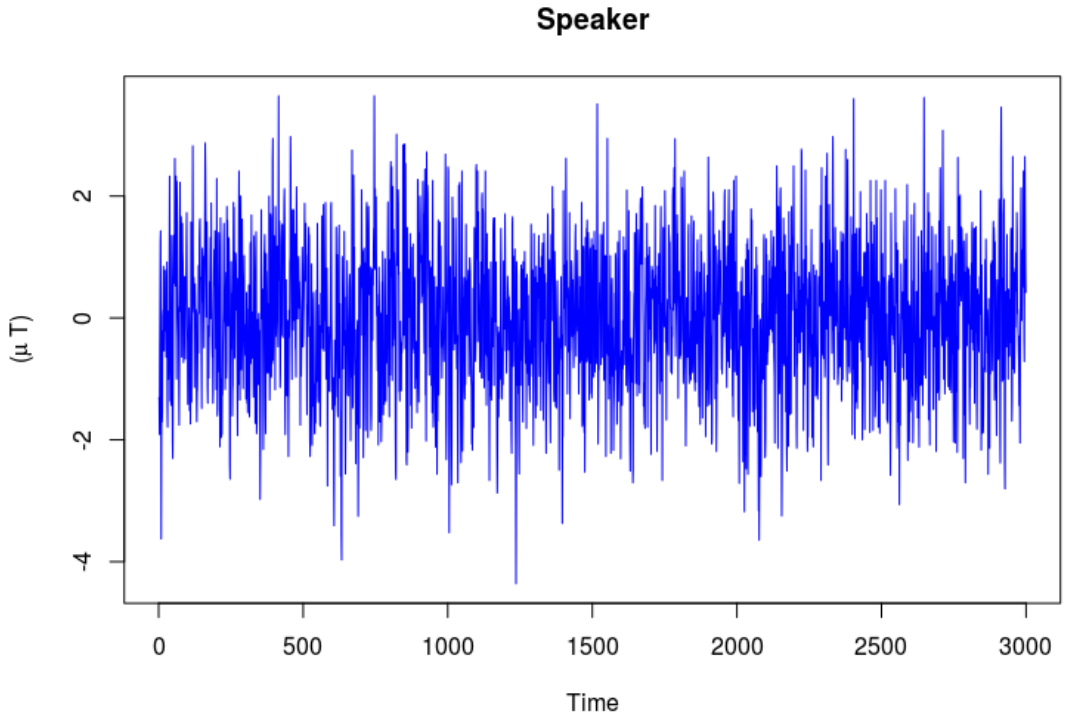


Figure 4.3: Received Signal for Speaker

4.2.2 Problems

The speaker needs to be near magnetometer in the phone to be able to be used for transmitting message. If the speaker is near by magnetometer, but the shielding is there on it then the affect is reduced further.

It needs to make a pretty loud noise to be able to generate enough power to transmit a signal.

Its activity can be made unnoticeable by using ultrasonic frequency for modulating the magnetometer. We did not test it with such frequencies.

4.3 Bluetooth

Bluetooth is common wireless protocol available on almost all devices available. The Bluetooth protocol works at the common open frequency of 2.4 GHz. It is used for all types of communication between devices from transferring files to playing music, this allows the Bluetooth communication to be inconspicuous as being used as a covert channel communication medium.

4.3.1 Solution

Bluetooth being a radio wave generating and receiving device/module can be used at sufficient power levels to be able to modulate the magnetometer. The radio waves are electromagnetic and have effect on devices which are sensitive to magnetic and/or electric interference.

There are multiple Bluetooth versions available. The latest version of it has Bluetooth Low Energy capabilities which allows it to and used in different manner than just the old versions which only allowed file transfer and music streaming when they became fast enough.

Bluetooth beacon

A beacon is any device which continuously transmits the same message again and again without changing anything.

This mode makes use of the BLE(Bluetooth Low Energy) mode of the module. As the signal is digital best way to generate a signal is to modulate the module to be transmitting and idle to generate the bits '1' and '0' respectively.

It is implemented using the Advertise API in the Android SDK.

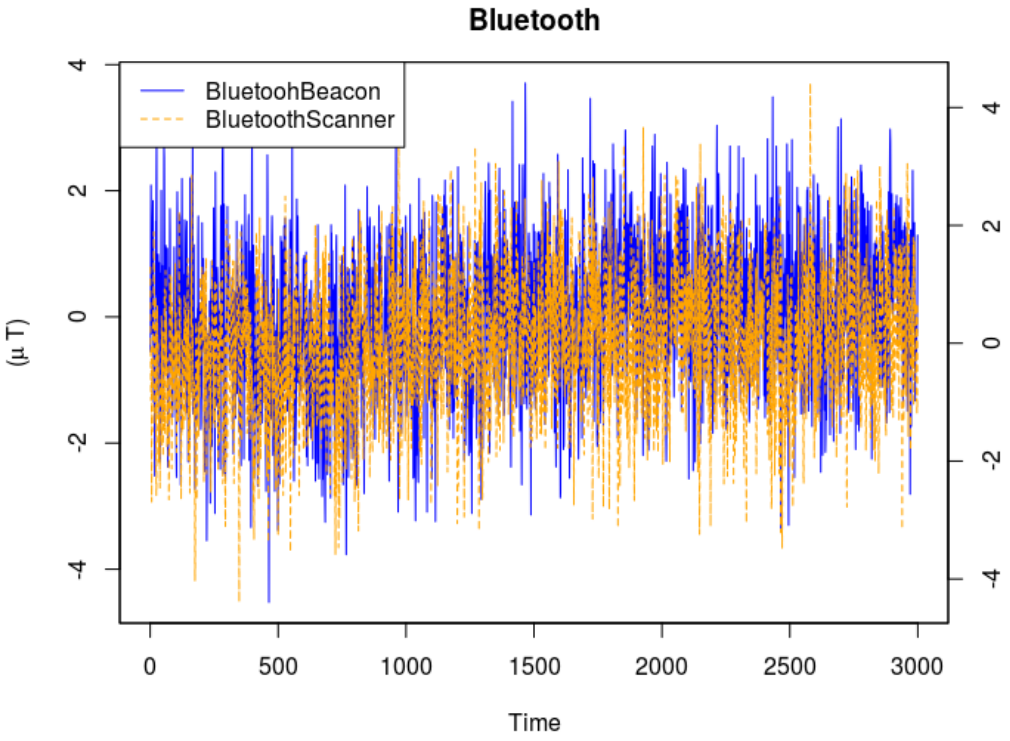


Figure 4.4: Received Signals for Bluetooth Scanner and Beacon

Bluetooth Scanner

The Bluetooth module uses a lot of power and its processing power to scan for new devices available in its range. This allows the CPU to be totally free and only affects the functionality of itself. If the device is already connected to a device and is transmitting information then also, the module puts most of its power in discovering new devices and the information transfer rate suffers.

This is method is implemented using the BluetoothManager APi provided in the Android SDK. In which the dicovery mode on is used for conveying bit '1' and it being off as bit '0'.

4.3.2 Problems

There are multiple physical limitations which make these methods ineffective:

1. BLE is really low energy thus does not generate powerful enough electromagnetic waves

2. The module needs to be physically close on the mother board which is not always the case.
3. Bluetooth module is shielded due to government radiation guidelines, this makes the fields to be low power.
4. There is a lot of noise present in the environment on the frequency on which it works.

4.4 WiFi module

The WiFi is a common wireless technology which works on frequencies 2.4 GHz and 5 GHz which are open frequencies. It is used for wireless internet connectivity, This makes the ambient noise on these frequencies to be quite large which reduces its capabilities as a magnetic covert channel.

4.4.1 Solution

This method was implemented by using apache commons net library, specifically its ftp Client module. We created a ftp Server on the system and uploaded a file to the server from phone. The transmitting was used as '1' and not transmitting as '0'.

4.4.2 Problems

There are multiple physical limitations which make these methods ineffective:

1. The module needs to be physically close on the mother board which is not always the case.
2. WiFi module is shielded due to government radiation guidelines, this makes the fields to be low power.
3. There is a lot of noise present in the environment on the frequency on which it works.

Also, there exists some practical limitations. There is lot of data transmission between router and client which introduces a lot of noise in this type of communication channel.

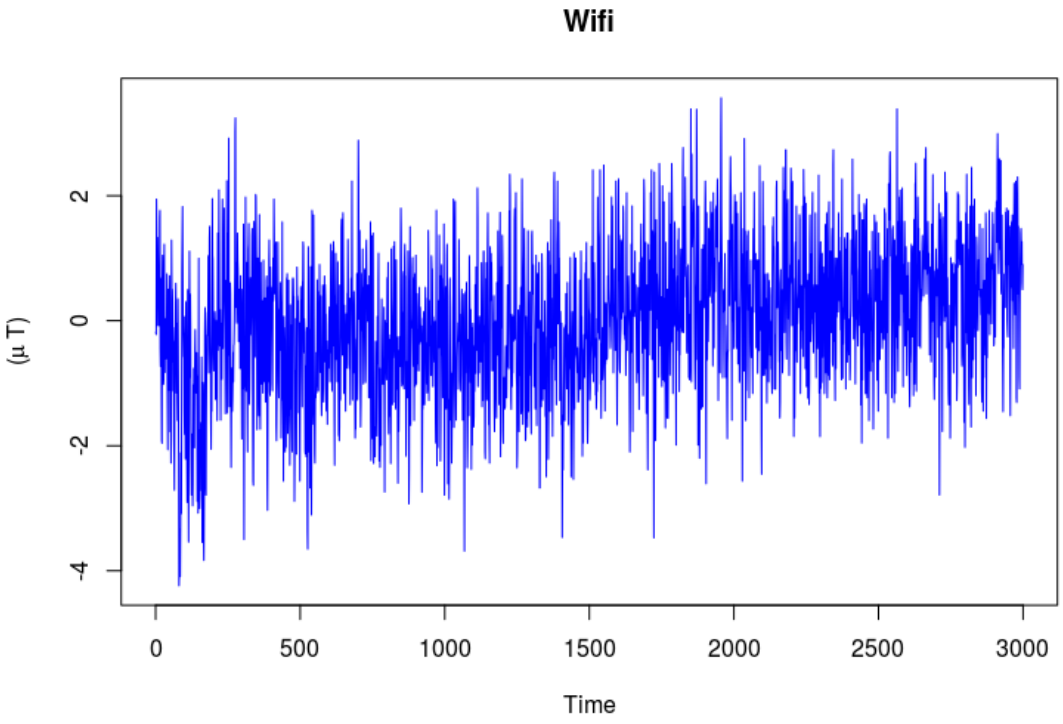


Figure 4.5: Received Signal for WiFi

4.5 GPS module

The GPS module in the phones takes a lot of power to function relative to other functionalities. The GPS depends on getting the signal from the satellites to get position of the device on earth. It needs at least three satellites to be available to be able to get a lock on the position using the triangulation method.

The modern GPS devices makes use of GNSS system to allow to get more accurate location. The GNSS works by providing weather and location correction information to remove signal errors caused due to various weather conditions.

The mobile phones may additionally use the network provider to get even more accurate position as it correlates the data and performs other position error corrections on the location obtained by the satellites.

4.5.1 Solution

The GPS covert channel tries to use processing and signal tracking capabilities of the module to generate enough magnetic interference in the normal working of the magnetometer to send a message with its modulation. As the signal we are trying to send is digital in nature the best method to generate the modulation is to cycle the GPS module on and off to get bits '1' and '0' respectively.

Our implementation makes use of the location API provided in Android SDK and turns cycles the GPS locating or not according the message we want to send.

4.5.2 Problems

Although the GPS module is being cycled as implemented it is not generating enough magnetic fields to get deflection/modulation on magnetometer to make the message recoverable.

There are multiple physical limitations which make these methods ineffective:

1. The module needs to be physically close on the mother board which is not always the case.
2. WiFi module is shielded due to government radiation guidelines, this makes the fields to be low power.
3. It takes more time to lock signal which reduces how fast and effectively it can be used

Evaluation SNR values calculated for the different methods implemented.

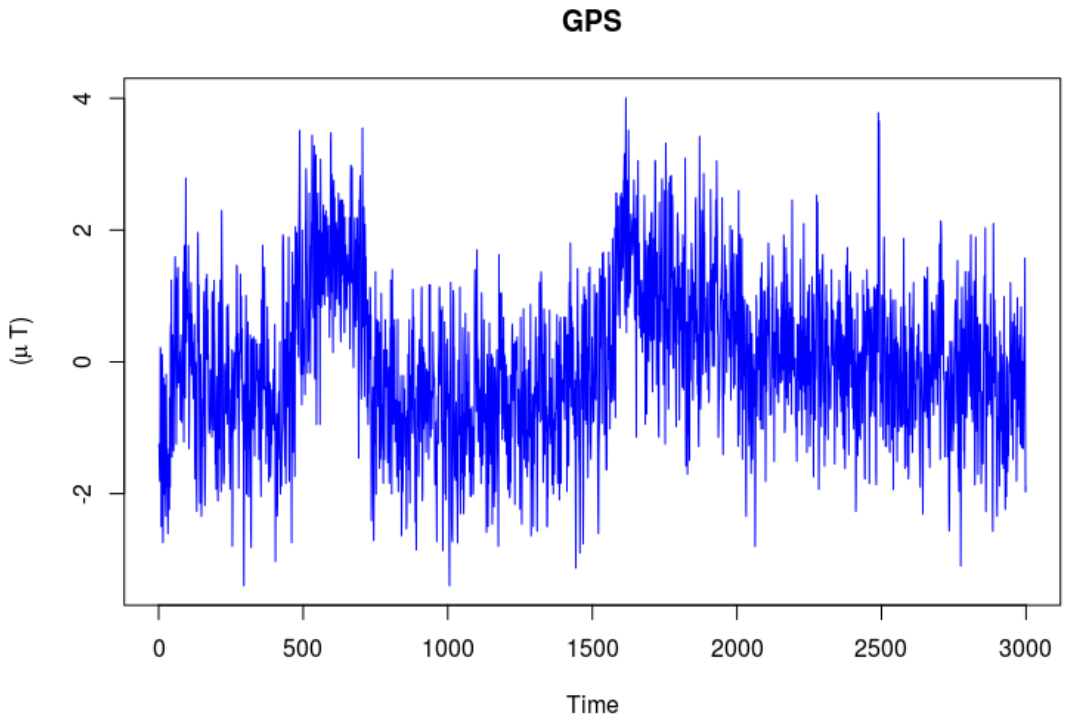


Figure 4.6: Received Signal for GPS

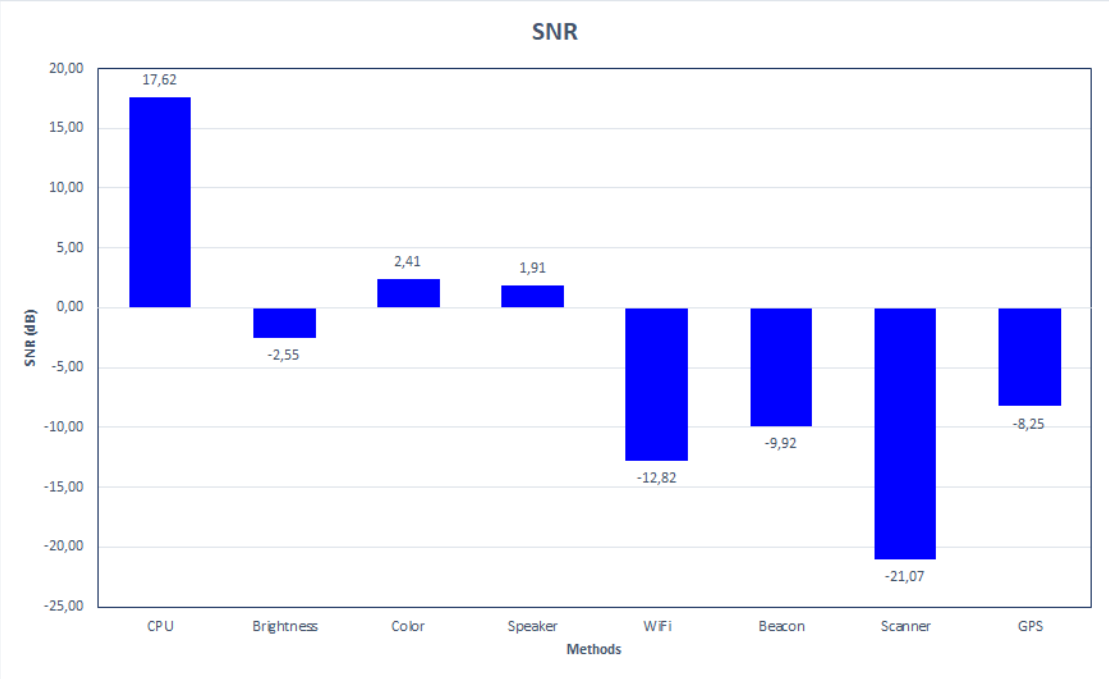


Figure 4.7: SNR values for different methods

- The best results were obtained by the CPU which gives the SNR value of 17 and above.
- The worst results were given by the methods generating electromagnetic radiation (Bluetooth, WiFi, GPS), this is due to the shielding on the respective modules to reduce radiation leakage.
- The screen brightness methods give better result, as they affect full length of device and thus are near magnetometer in all cases.

5 Countermeasures: analyzing procfs and sysfs data

5.1 Introduction

Main idea of this task is to develop a countermeasure against implemented covert channel. Basic requirement is to adjust the magnetometer values such that it can not be decoded by the receiver application. To calculate new adjusted sensor values we use timeseries of CPU Utilization and Power Consumption. Which requires device to be rooted to get hold of procfs and sysfs files.

5.2 Implement the recording of the instant CPU workload (over /proc/stat) and battery power traces (over /sys/class/power_supply/battery) as the time series.

5.2.1 Solution

To access CPU workload from `/proc/stat` and Power traces from `/sys/class/power_supply` we initially rooted the device. `/proc/stat` contains the counters aggregated since the system first booted. The very first "cpu" line aggregates the numbers in all of the other "cpuN" lines. These numbers identify the amount of time the CPU has spent performing different kinds of work. Time units are in USER_HZ or Jiffies (typically hundredths of a second). Below is the sequential list of columns and their significance from the file. Goal is to calculate device's CPU Utilization hence, only first line from the file is required to calculate aggregated CPU Utilization.

user: Time spent with normal processing in user mode.
nice: Time spent with niced processes in user mode.
system: Time spent running in kernel mode.
idle: Time spent in vacations twiddling thumbs.
iowait: Time spent waiting for I/O to completed. This is considered idle time too.
irq: Time spent serving hardware interrupts. See the description of the intr line for more details.
softirq: Time spent serving software interrupts.

steal: Time stolen by other operating systems running in a virtual environment.
guest:Time spent for running a virtual CPU or guest OS under the control of the kernel.

After this we faced a problem as SELinux is in enforcing mode and it does not allow an application to read the OS related files. To mitigate this issue, we changed the SELinux from enforcing to permissive mode. To achieve this we execute below mentioned commands.
`get enforce` (To know the current mode for SELinux)
`set enforce 0` (To set it to permissive mode)

CPU Utilization calculations
As these values are counters and to calculate instant CPU Utilization we have to take difference between two snapshots of the file. To achieve this we introduce sleep of 5 milliseconds and record present and prev values of all variables. Below mentioned formulas computes instant CPU Utilization with 5 milliseconds granularity. We could not reduce the sleep interval below 5 millisecond as it leads to totald to '0' which makes CPU Utilization 'NaN'

```
PrevIdle = previdle + previowait
Idle = idle + iowait
PrevNonIdle = prevuser + prevnice + prevsystem + previrq + prevsoftirq + prevsteal
NonIdle = user + nice + system + irq + softirq + steal

PrevTotal = PrevIdle + PrevNonIdle
Total = Idle + NonIdle
```

differentiate: actual value minus the previous one

```
totald = Total - PrevTotal
idled = Idle - PrevIdle

CPU\_Percentage = ((totald - idled)/totald)*100
```

Power Consumption calculations
To calculate instant power consumption of the device we are using two files from power_supply. First file `/sys/class/power_supply/battery/current_now` has instant current consumption counter and conceptually negative counter value indicates consumption, on the other hand, positive counter value indicates device is in charging state hence current is fed to the device.
Second file `/sys/class/power_supply/battery/voltage_now` has instant voltage counter and it is always positive.
It is straight forward as `Power=Current*Voltage`, we have to take product of counters from both files mentioned above. We are multiplying above product by '-1' to make correlation value positive and directly proportional to CPU this way it is easier to generalize the sensor adjustments formula in terms of implementation.

5.2.2 Implementation

To implement the timeseries of both Power Consumption and CPU Utilization we are using HandlerThreads and Service in android. We have implemented 'BatteryThread' and 'CPULoadReader' handler threads to calculate the CPU Utilization and Power Consumption.It stores all the values in double array of fixed size(record_time*100).
These Handler Threads are used by a service called 'TimeSeriesRecorderService' which generates a time series in synchronization with sensor values.
In Looper implementation 'MESSAGE_READ' message is handled by calculating the respective measurements using separate functions. Handler in a 'TimeSeriesRecorderService' calls the measurement by sending 'MESSAGE_READ' which is index '1'.
After calling the measurement function from respective handler, it is stored in predefined sized array and returned to the service after it is full.

5.2.3 Problems

We faced series of major problems during implementation for this task. Firstly, NPE (Null Pointer Exception) when reading *procfs* and *sysfs* files. As after rooting the device we could manually read the file from adb shell. But it was not working from application, after further analysis changing SELinux mode helped to solve this problem. Secondly, synchronizing sensor reading and both CPU and Battery readings are not possible with Java threads. Hence we decided to implement HandlerThread to achieve this. Finally, It was not clear that why battery reading are negative, but after reading android documentation we realised the significance of the sign in current_now file. Also, It after understanding that readings for Power Consumption does not correlate well when device is connected by USB cable we decided to perform all tests without USB plugged in.

5.3 Evaluate how the instant consumed power correlates with the peak and idle CPU activity.

5.3.1 Solution

Time series service from previous sub-task generates a time series with X,Y,Z,CPU,Battery readings in it. Here CPU and Battery corresponds to 'CPU Utilization' time series and 'Power Consumption' time series data. Same naming convention is used further sections. To correlate the information offline we are using cor() method in 'R' which by default calculates correlation coefficient using Pearson Algorithm.
Goal for this task is to correlate CPU and Battery time series with Magnetometer. And based on their relationship decide if they can be used to introduce sensor adjustments which ultimately avoids the attack.

	X	Y	Z	CPU	Battery
X:	1.00000	0.3702669	0.8793347	0.8844152	0.6685094
Y:	0.3702669	1.00000	0.4877364	0.4975796	0.3956130
Z:	0.8793347	0.4877364	1.00000	0.9193842	0.6893840
CPU:	0.8844152	0.4975796	0.9193842	1.00000	0.7236171
Battery:	0.6685094	0.3956130	0.6893840	0.7236171	1.00000

Table 5.1: Correlation Matrix

From table 5.1 we can see that magnetometer axes X and Z are strongly correlated with CPU by 0.88 and 0.91 respectively.Similarly Battery has 0.66 and 0.68 correlation coefficients for X and Z axis. we can observe that Y axis data is weakly correlated with Battery and CPU both.On the other hand, CPU and Battery has correlation of 0.72 which is obvious inference from their correlation with magnetometer. All correlation coefficients with Battery is positive as we multiply it by '-1' (details in section 5.2.1)
Based on strong correlation between magnetometer, CPU and Battery we can say that both CPU and Battery time series can be used to introduce adjustments in magnetometer which can avoid the attack.
Further, in Figure 5.1 it can be seen that strongly correlated sensor data varies with direct relation with CPU and Battery. Hence, it can be seen that peak for CPU and Battery is almost overlapping peaks for Sensor values. But, In case of plots for Battery on top there is slight lag of pattern highlighted in red. This happens as Power Consumption deflection trend is sluggish than Sensor and CPU. That means battery reacts slowly to high/low CPU activity.

5.4 Implement the sensor reading adjustments (for previously recorded traces). Evaluate the error (e.g., the MSE) which is introduced by the adjustments. Evaluate the PoC covert channel in presence of adjustments.

5.4.1 Solution

After understanding that both CPU and Battery time series can be used to adjust the magnetometer, we started working on sensor adjustments.Goal is to have two sensor adjustment formulas one using CPU time series and other using Battery. Also, introduced adjustments should be able to avoid the attack without affecting normal signal.

From Figure 5.1 bottom plots we aim to reduce the peaks correlating with 100% CPU. So, we first compute sensor value deflection for 1% CPU usage. Similarly, from top plots

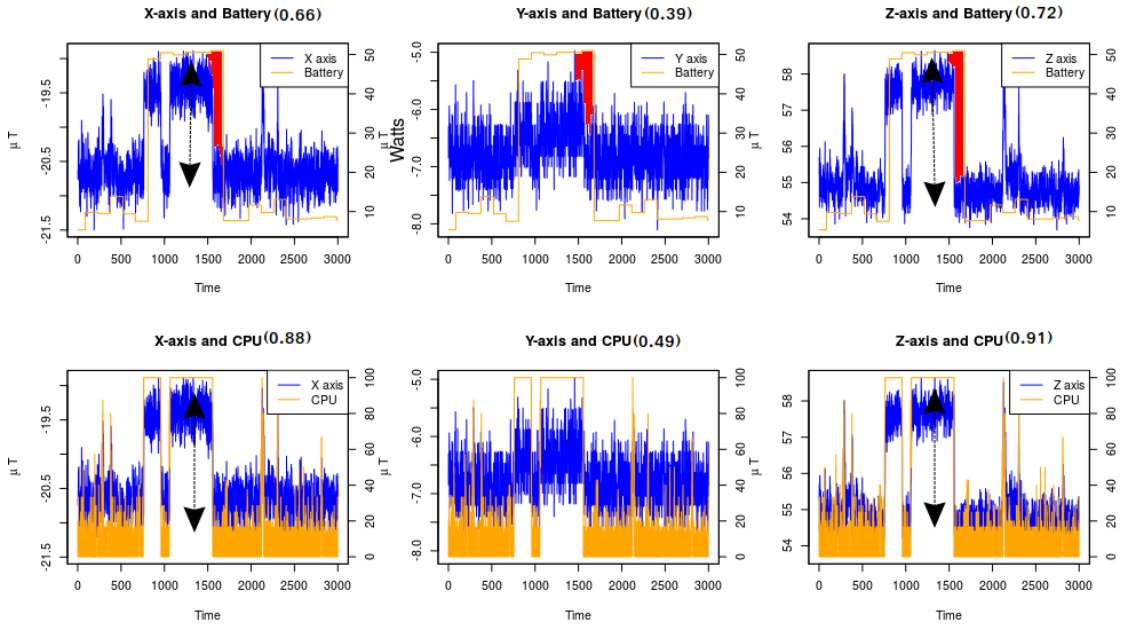


Figure 5.1: Sensor correlating with Battery and CPU

to reduce the peaks correlating with highest power consumption (>=45 watts) we first calculate sensor value deflection for 1 watt battery consumption. These values are referred as *adjustment coefficient (K)* for further sections. To compute 'K' we use Formula 5.1 and 5.2 for CPU and Battery respectively.

After successfully adjusting the sensor highlighted correlation coefficients from table 5.1 should be reduced thus making attack difficult and keeping the normal signal untouched.

Adjustment Coefficient (K) for CPU

$$K_{xyz}^{CPU} = \frac{mean(CPU_{xyz}^{100}) - mean(CPU_{xyz}^{0})}{100} \tag{5.1}$$

Where:

$$K_{xyz}^{CPU} = \text{Adjustment Coefficient for X,Y and Z axis thus, } K_x, K_y \textit{ and } K_z$$
$$CPU_{xyz}^{100} = \text{X, Y and Z values for 100\% CPU usage}$$
$$CPU_{xyz}^{0} = \text{X, Y and Z values for 0\% CPU usage}$$
$$Denominator(100) = \text{Range of CPU values}$$

Adjustment Coefficient (K) for Battery

$$K_{xyz}^{Battery} = \frac{mean(Battery_{xyz}^{\geq 45}) - mean(Battery_{xyz}^{\leq 15})}{Agg(B^{\geq 45}) - Agg(B^{\leq 15})} \tag{5.2}$$

Where:

$K_{xyz}^{Battery}$ = Adjustment Coefficient for X,Y and Z axis thus, K_x, K_y *and* K_z

$Battery_{xyz}^{\geq 45}$ = X, Y and Z values with power consumption ≥45

$Battery_{xyz}^{\leq 15}$ = X, Y and Z values with power consumption ≤15

$Agg(B^{\geq 45}) - Agg(B^{\leq 15})$ = Range of Battery values, B: Battery time series

After calculating adjustment coefficients (K_x, K_y *and* K_z) using formula 5.1 and 5.2 for CPU and Battery, they can be used to adjust X,Y and Z sensor values using below mentioned formula.

Sensor Adjustments

$$Adjusted_{xyz}^{CPU} = Raw_{xyz} + K_{xyz}^{CPU} \times C \tag{5.3}$$

$$Adjusted_{xyz}^{Battery} = Raw_{xyz} + K_{xyz}^{Battery} \times B \tag{5.4}$$

Where:

$Adjusted_{xyz}^{CPU}$ = Adjusted sensor values using CPU

$Adjusted_{xyz}^{Battery}$ = Adjusted sensor values using Battery

Raw_{xyz} = Real sensor values

K_{xyz} = Adjustment Coefficient

C = CPU time series

B = Battery time series

After performing multiple offline adjustments it shows that adjustment coefficients can be treated as a constants in adjustment formula 5.3. Hence, K_x, K_y and K_z constants assumed from tests are -0.0107075, -0.003415 and -0.027064 for CPU and -0.0275695, -0.0067435 and -0.0665255 for Battery respectively.As correlation coefficient for Y axis was smaller than X and Z axis. We can notice that K_y is smaller than both K_x and K_z.

Figure 5.2 shows the raw sensor values adjusted using both Battery and CPU time series with default coefficients mentioned above. We could observe reduction in the peaks for X and Z axis. As Y axis does not have meaningful sensor values for attack and considering small value of K_y Y axis almost stays same after adjustments. Adjusted results from the graph are not successfully decoded by the receiver hence avoiding the attack

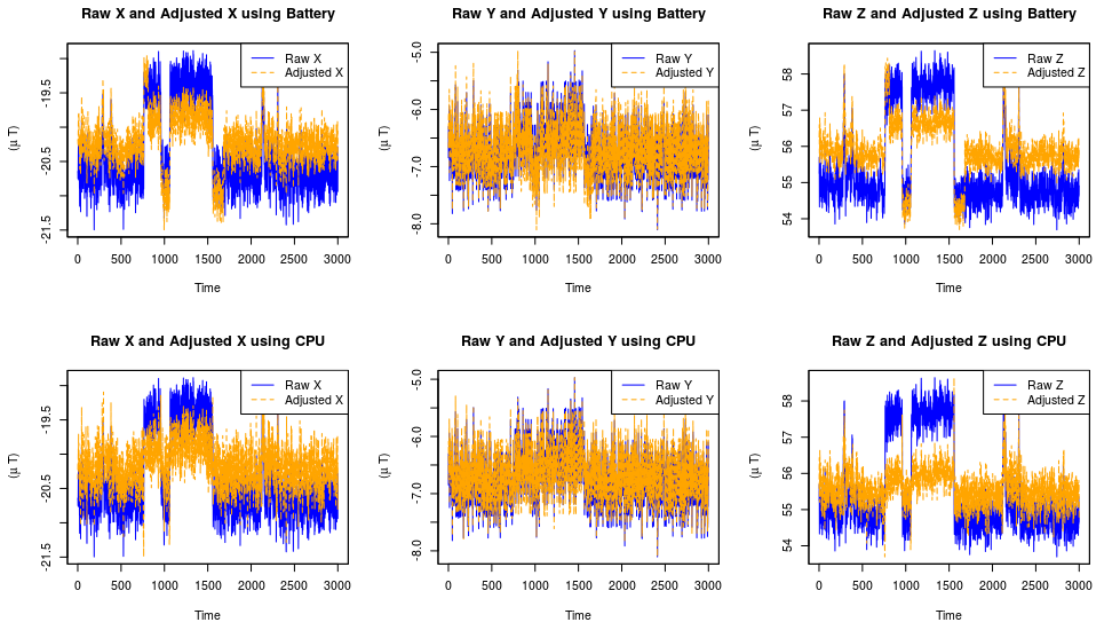


Figure 5.2: Adjusted Sensor and Raw Sensor

by adjusting sensor data using CPU and Battery is achieved.
Also, from the plots it can be seen that adjustments are only dealing with the peaks transmitted during the attack and not adjusting normal signal hence, non-attack relevant information from the signal is not affected.
Further section describes real-time sensor adjustments and evaluation of Error introduced after adjustments along with PoC evaluation results.

5.5 Implement the real-time adjustments of sensor readings: the function adjusts newly received sensor readings with regard to the latest power/CPU statistics.

5.5.1 Solution

Goal for this task is to perform online adjustment of the sensor reading. As we already have formula to adjust the sensor readings from earlier section, we need to use the formula to implement real-time adjustments.
We observed that calculating adjustment coefficients before attack improves the adjustments better than using constant coefficients. Therefore, we developed a coefficient calibration mechanism which computes values for the coefficients before attack is performed and if not calibrated we use predefined constant coefficients as default for adjustments.

Calibrating coefficients before attack is achieved by elevating CPU to 100% to mimic logical 1 and 0% to mimic for logical 0.Sensor, CPU and Battery values are further used to compute coefficients using formula 5.1 and 5.2. It is not always necessary to use calibrated coefficients as they can be assumed as constant values as mentioned earlier. But using calibrated coefficients helps improving the adjustments in sensor readings.

To implement real-time sensor adjustments we are using 'Handler Thread' utilities created earlier to get CPU and Battery time series. Further, a service called 'UpdateCoefficientService' calibrates adjustment coefficients before attack and 'SensorRecorderService' to apply real-time adjustments online.

Online Adjustments Algorithm

1. Calibrate Adjustment Coefficients (Optional)
   a) Transmit '10' with 1 bps (Assuming no ongoing attack)
   b) Record Sensor, procfs and sysfs data (X,Y,Z,CPU,Battery)
   c) Discard first and last 25 values for each transmitted bit to reduce the noise. (1bit 100 readings after discard only middle 50 are considered)
   d) Calculate mean values for each axis. For CPU $mean(CPU^{100}_{xyz})$ and $mean(CPU^{0}_{xyz})$. For Battery $mean(Battery^{\geq 40}_{xyz})$ and $mean(Battery^{\leq 15}_{xyz})$
   e) Compute absolute of the coefficients using formula 5.1 and 5.2
   f) Based on peak direction decide sign for K. For instance, if $(CPU^{100}_{x} > CPU^{0}_{x})$ then K is negative otherwise positive.
2. Calibrated coefficients are stored along with choice of default predefined coefficients for further real-time adjustments .
3. When sensor change event occurs store raw values and apply formula 5.3 and 5.4 for CPU and Battery adjustments online using coefficients calculated in previous step. If not calibrated default coefficients can be used for adjustments.
4. Adjusted Sensor values are passed to decoder for further decoding.

5.5.2 Evaluation

This section evaluates the error introduced in the signal after adjustments along with how adjustments can avoid the attack. Further, we also evaluate calibration mechanism of the coefficients.

1. Evaluate error introduced by adjustments
   Adjustments introduced are trying to bring the logical '1's close to '0's hence, to evaluate the introduced error we perform test where transmission of '10101010' is adjusted by both CPU and Battery time series. Error computed in this method denotes the quantity of error adjustments has introduced in raw sensor values.

   $$Error = RMSE(Adjusted_{xyz}, Raw_{xyz}) \tag{5.5}$$

   Table 5.2 contains error computed for X,Y and Z axis in relation with both CPU and Battery adjusted sensor values. from error values we can see that more adjustments is done for Z access followed by X and lowest for Y axis. This trend can be compared with the correlation matrix. Hence, strongly correlated axes (X and Z) has been more adjusted than weakly correlated axis (Y).

2. Evaluate PoC for attack
   From section 3.4 it can be seen that with 1bps bitrate and 8 bits payload PoC gives 0% BER. Therefore, we use same configurations to evaluate PoC with adjustments.
   Figure 5.2 contains BER for 5 adjustment tests performed using both Battery and CPU. All tests are performed with 1bps bitrate and 8bits payload, and we can clearly notice that BER value has increased from 0% (Before adjustment section 3.4) to non-zero (after adjustments) thus adjustments made are affecting the attack.

3. Evaluate Coefficient Calibration
   In this method we evaluate how close calibrated coefficients were to actual coefficients where actual coefficients are calculated after attack and real-time adjustments are completed.

   $$Error = RMSE(CalibratedCoeff, ActualCoeff) \tag{5.6}$$

   Table 5.3 contains error computed for K_x, K_y and K_z in relation with both CPU and Battery adjustment coefficients. from error values overall we can say that calibrated coefficients are almost close to actual coefficients for both CPU and Battery adjustment methods. But, comparing error between calibrated coefficients for CPU is more precise than that of Battery.

	X	Y	Z
$RMSE_{CPU}$:	0.4713476	0.1862691	1.146225
$RMSE_{Battery}$:	0.5286804	0.1991515	1.279777

Table 5.2: RMSE($Adjusted_{xyz}, Raw_{xyz}$)

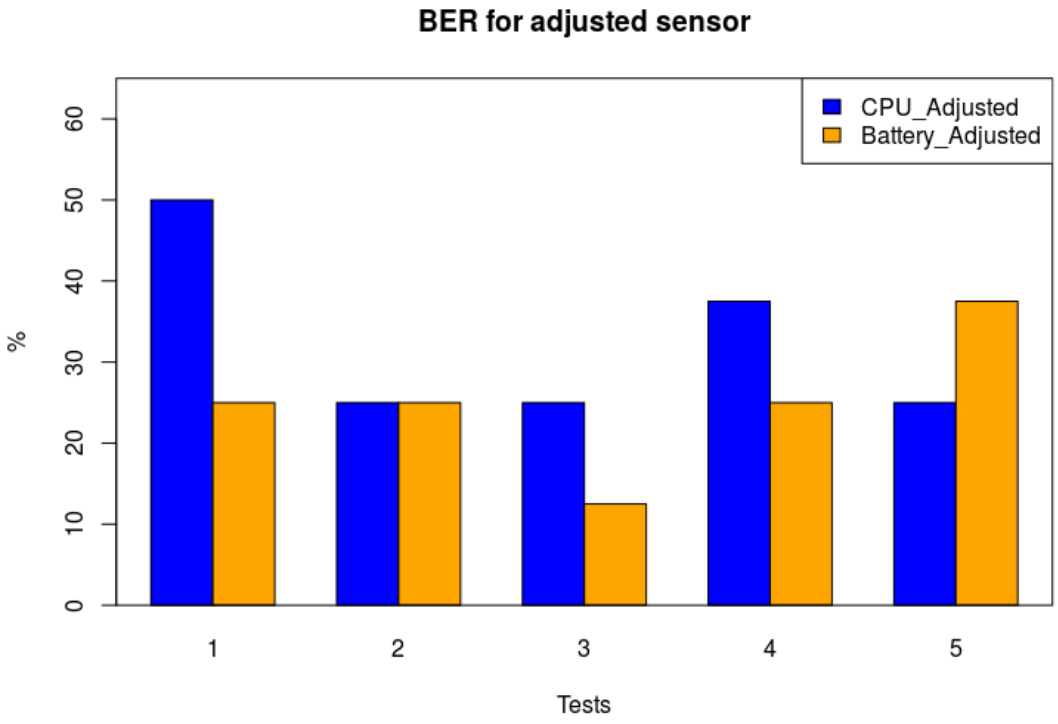


Figure 5.3: BER for Adjusted Sensor with 1bps/8bits

	K_X	K_Y	K_Z
$RMSE_{CPU}$:	0.0009648257	0.0009182117	0.0003511287
$RMSE_{Battery}$:	0.003127622	0.002467354	0.004227361

Table 5.3: RMSE(Calibrated Coeff, Actual Coeff)

6 Countermeasures: analyzing sensor fusion data

We would like to find out a countermeasure capable of detecting our covert-channel attack. Since the magnetometer is disturbed by the generation of magnetic field, a smart idea as countermeasure is to use sensors in order to detect the attack. By combining sensors together in some ways, we assume we can find some threshold which would indicate that there is a high magnetic field compared to a standard behaviour of the mobile.

6.1 Implement the recording of gyroscope and accelerometer together with magnetometer.

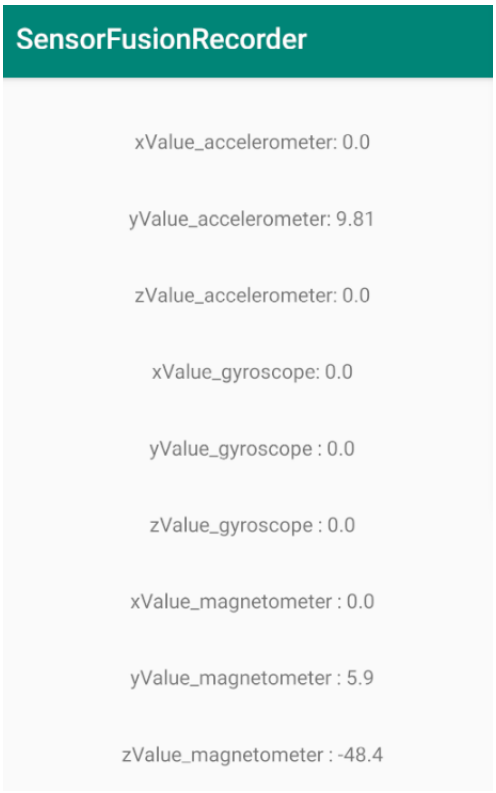
The first solution, we found, just uses a sensor manager, from the *SensorManager* class in Java, in order to instantiate three different sensors : an accelerometer, a gyroscope and a magnetometer, all coming from the *Sensor* class.
Once we had these sensors as variables, we implemented a function, called *onSensorChanged()*, which listens to any variation of the real sensors in order to print the new values on the screen, in real time.
An example of the user interface is given by *figure 6.1*.

6.2 Implement the function which computes the relation between magnetometer and other sensors (discuss with the supervisor). Evaluate how this relation holds for a static and moving device, identify the threshold which identifies abnormal deviation in measurements.

6.2.1 Theory

Before starting to speak about the relation between sensors, let us introduce the theory. As explained in the first part of this documentation, accelerometer provides the gravity vector (the vector pointing towards the centre of the earth), gyroscope provides the angular rotation speed around each 3D-axis and the magnetometer works as a compass.

Figure 6.1: First implementation of the application responsible for analyzing sensor fusion data



Information of two sensors only would be enough to compute orientation for the device, but it would also be inaccurate... In fact, the magnetometer output includes a lot of noise and the accelerometer is measuring all the forces applying to the device and could then be disturbed in an inappropriate way. The output from gyroscope has to be integrated to get some angles but through this process, some small errors are introduced in each step. Therefore, a drift appears making the gyroscope output disturbed.
Those errors need to use all the other sensors to be corrected. This is called sensor fusion and there exist many of them.

6.2.2 First idea : complete sensor fusion

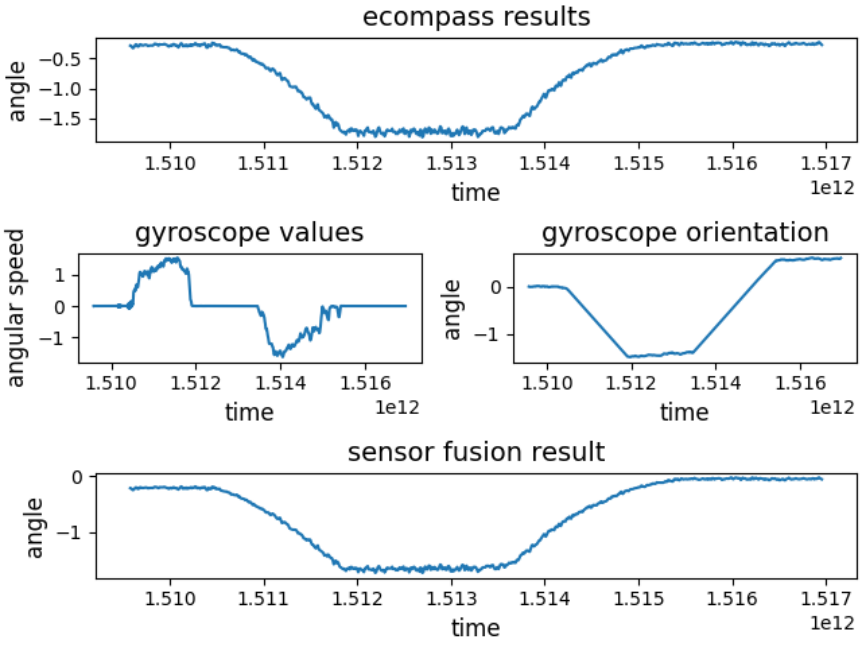
To avoid both, gyro drift and noisy orientation, the gyroscope output is applied only for orientation changes in short time intervals, while the fusion between magnetometer and acceletometer data, called ecompass, is used as support information over long periods of time. This is equivalent to low-pass filtering of the ecompass signal and high-pass filtering of the gyroscope signals.

Some results of the implementation are illustrated by *Figure 6.2*. It shows how the fusion sensor works by doing first a 90° rotation and going back to the initial position. To obtain these results, we used the following protocol :

- Run 30 seconds the sensor record.
- Turn the phone while the sensor record is running, first 90° to the right/left, then to the other direction to come back to the initial position.
- Pull files from the record with **adb** on a computer.
- Create file "motion_sensors_merged.csv" and run the python script "motion_merged.py" to fill in the file created.
- Do post-processing (using "sensor.py" in the folder called "pythonCountermeasure")

This can be improved and is somehow not accurate enough to detect our attack since we have only the results of sensor fusion and nothing to compare them. So there is no way to detect the side-channel attack with only the sensor fusion. We could use the ecompass and the gyroscope orientation to see how they differ from each other when the device is under a high magnetic field. That implies first to know how they behave usually without this high magnetic field and again that would not be accurate since the gyroscope produces a drift that we have to take in account.

Figure 6.2: Sensor Fusion results with a 90° rotation



6.2.3 Second idea : ecompass and IMU complementary filter

To avoid this drift, we used again the accelerometer. In fact, on one hand we have the ecompass (accelerometer + magnetometer) which can be disturbed by a high magnetic field. On the other hand, we would have the IMU complementary filter (accelerometer + gyroscope) which would not be disturbed theoretically by a high magnetic field. For the experimentation, we used the following protocol :

- Run 30 seconds the sensor record.
- First case : run nothing on the device.
- Second case : run a covert-channel attack with only '1' bits.
- Proceed as the previous protocol explained for post-processing.

The results are given by *Figure 6.3* and *Figure 6.4*. As we can see, there is noise from the magnetometer output in the ecompass fusion in both cases. However, one can distinguish that the average value for the difference between ecompass and IMU is around 0.1 rad for the first case and around 1.3 for the second case. That is why, we decided to use a threshold of 1 rad for the real-time detection.

Figure 6.3: First case : low magnetic field

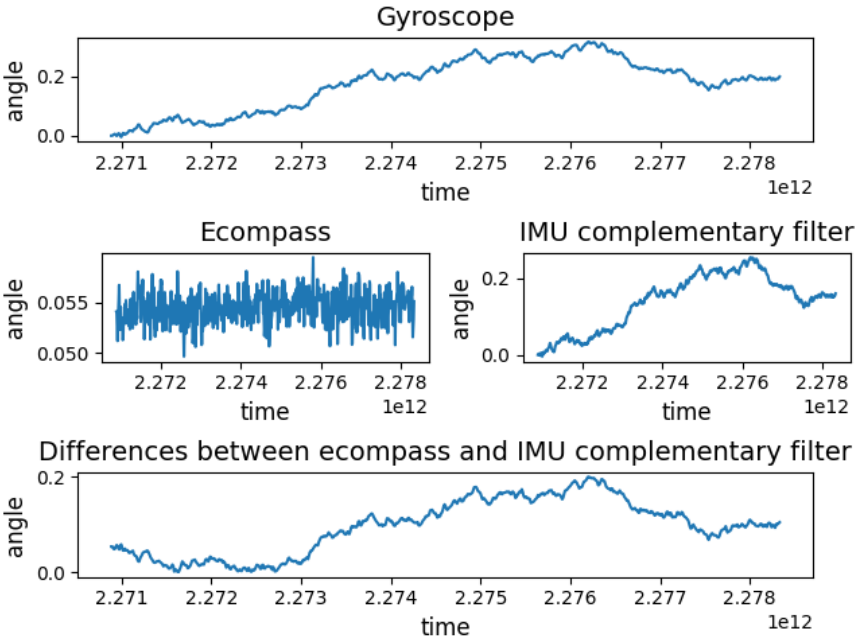


Figure 6.4: Second case : high magnetic field

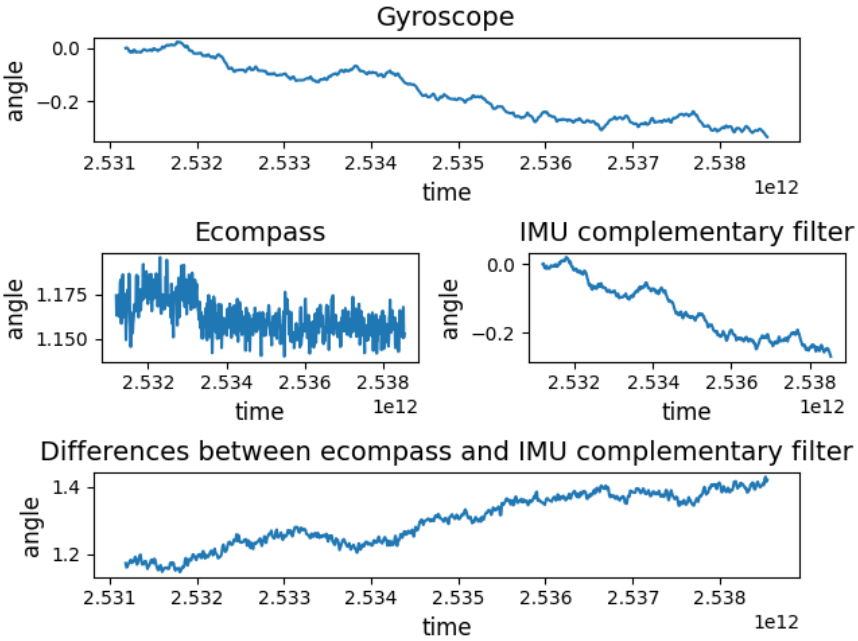
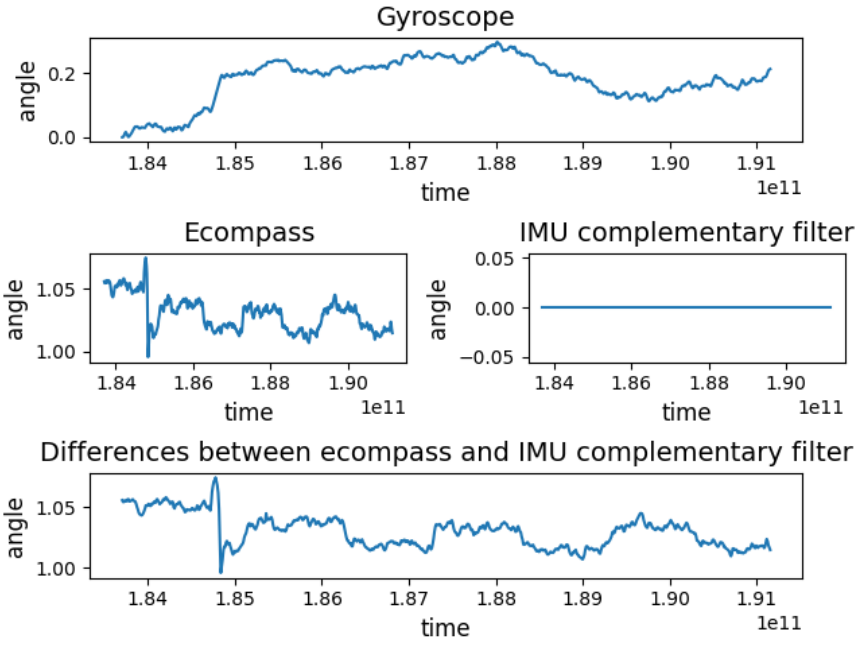


Figure 6.5: High-pass and low-pass filter under attack



6.2.4 Importance of the high-pass and the low-pass filter

As one can see on *Figure 6.3* and *Figure 6.4*, ecompass results include a lot of noise. A low-pass filter can reduce this noise and make the result smoother. Moreover, a high-pass filter can be applied to the IMU to make it appear as a straight line as much as possible.
To realize the low-pass filter, we used the following formula :
$new\_value = \alpha * new\_calculated\_value + (1 - \alpha) * previous\_value$
By running the script for some different values, we chose $\alpha = 0.2$. The result under attack can be seen on *Figure 6.5*.
The high-pass filter is not a proper high-pass filter. When the phone does not move, the result is a straight line (see *Figure 6.5*). When the phone executes a rotation, the result appears with a stair form because of a threshold that we decided experimentally in order to make the experimentation better without movement. We assumed it was correct to implement this filter following this way since we can detect correctly an attack running with this filter as illustrated by *Figure 6.5*.

From the result with python, we can find a threshold around 1.025 rad above which we would say that a high magnetic field is detected.

6.3 Implement the detection system which analyses relation between sensors in real-time and marks the magnetometer measurements as potentially disturbed if the relation does not hold.

Two different version for the real time detection have been implemented. However, none of them can detect properly the attack in real time. The first version is the application called *SensorFusionCountermeasure* and the second one is an extra feature of the receiver application.

Bibliography

[1] *Accelerometer*. Oct. 2019. URL: https://en.wikipedia.org/wiki/Accelerometer (cit. on p. 14).

[2] *Accelerometer vs. Gyroscope: What's the Difference?* URL: https://www.livescience.com/40103-accelerometer-vs-gyroscope.html (cit. on p. 15).

[3] *Application Sandbox*. en. URL: https://source.android.com/security/app-sandbox (visited on 11/05/2019) (cit. on p. 12).

[4] D. Asonov and R. Agrawal. "Keyboard acoustic emanations". In: *IEEE Symposium on Security and Privacy, 2004. Proceedings. 2004*. May 2004, pp. 3–11. DOI: 10.1109/SECPRI.2004.1301311 (cit. on p. 17).

[5] By. *Sniffing Keystrokes Via Laser, Power Lines*. en-US. Mar. 2009. URL: https://hackaday.com/2009/03/20/sniffing-keystrokes-via-laser-power-lines/ (visited on 11/04/2019) (cit. on p. 7).

[6] Y. Cai et al. "Magnetometer basics for mobile phone applications". In: 54 (Feb. 2012) (cit. on pp. 18, 19).

[7] Swarup Chandra et al. "Towards a Systematic Study of the Covert Channel Attacks in Smartphones". In: *International Conference on Security and Privacy in Communication Networks*. Ed. by Jing Tian, Jiwu Jing, and Mudhakar Srivatsa. Cham: Springer International Publishing, 2015, pp. 427–435. ISBN: 978-3-319-23829-6 (cit. on p. 12).

[8] Arijit Raychowdhury (Georgia Tech) Chastain D-E Shreyas Sen (Purdue University). *Conference on Cryptographic Hardware and Embedded Systems 2019*. 2019. URL: https://ches.iacr.org/2019/program.shtml (cit. on p. 4).

[9] Sujit Rokka Chhetri, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. "Poster : Exploiting Acoustic Side-Channel for Attack on Additive Manufacturing Systems". In: 2016 (cit. on p. 17).

[10] D. Das et al. "STELLAR: A Generic EM Side-Channel Attack Protection through Ground-Up Root-cause Analysis". In: *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. May 2019, pp. 11–20. DOI: 10.1109/HST.2019.8740839 (cit. on p. 5).

[11] M. Guri, M. Monitz, and Y. Elovici. "USBee: Air-gap covert-channel via electromagnetic emission from USB". In: *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. Dec. 2016, pp. 264–268. DOI: 10.1109/PST.2016.7906972 (cit. on p. 7).

[12] Mordechai Guri, Andrey Daidakulov, and Yuval Elovici. "MAGNETO: Covert Channel between Air-Gapped Systems and Nearby Smartphones via CPU-Generated Magnetic Fields". In: *CoRR* abs/1802.02317 (2018) (cit. on p. 8).

[13] Mordechai Guri et al. "GSMem: Data Exfiltration from Air-Gapped Computers over GSM Frequencies". In: *24th USENIX Security Symposium (USENIX Security 15)*. Washington, D.C.: USENIX Association, Aug. 2015, pp. 849–864. ISBN: 978-1-931971-232. URL: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/guri (cit. on p. 7).

[14] Mordechai Guri et al. "GSMem: Data Exfiltration from Air-gapped Computers over GSM Frequencies". In: *Proceedings of the 24th USENIX Conference on Security Symposium*. SEC'15. Washington, D.C.: USENIX Association, 2015, pp. 849–864. ISBN: 978-1-931971-232. URL: http://dl.acm.org/citation.cfm?id=2831143.2831197 (cit. on p. 5).

[15] Mordechai Guri et al. "DiskFiltration: Data Exfiltration from Speakerless Air-Gapped Computers via Covert Hard Drive Noise". In: *CoRR* abs/1608.03431 (2016) (cit. on p. 6).

[16] Mordechai Guri et al. "Fansmitter: Acoustic Data Exfiltration from (Speakerless) Air-Gapped Computers". In: *CoRR* abs/1606.05915 (2016) (cit. on p. 6).

[17] Mordechai Guri et al. "ODINI : Escaping Sensitive Data from Faraday-Caged, Air-Gapped Computers via Magnetic Fields". In: *CoRR* abs/1802.02700 (2018) (cit. on p. 7).

[18] M. Guri et al. "AirHopper: Bridging the air-gap between isolated networks and mobile phones using radio frequencies". In: *2014 9th International Conference on Malicious and Unwanted Software: The Americas (MALWARE)*. Oct. 2014, pp. 58–67. DOI: 10.1109/MALWARE.2014.6999418 (cit. on p. 7).

[19] *Gyroscope*. Oct. 2019. URL: https://en.wikipedia.org/wiki/Gyroscope (cit. on p. 15).

[20] Ahmed Al-Haiqi, Mahamod Ismail, and Rosdiadee Nordin. "Keystrokes Inference Attack on Android: A Comparative Evaluation of Sensors and Their Fusion". In: *Journal of ICT Research and Applications* 7 (Nov. 2013), pp. 117–136. DOI: 10.5614/itbj.ict.res.appl.2013.7.2.2 (cit. on p. 16).

[21] Gierad Laput et al. "EM-Sense: Touch Recognition of Uninstrumented, Electrical and Electromechanical Objects". In: Nov. 2015, pp. 157–166. DOI: 10.1145/2807442.2807481 (cit. on p. 4).

[22] *Magnetometer*. Sept. 2019. URL: https://en.wikipedia.org/wiki/Magnetometer (cit. on p. 15).

[23] *Magnetometer in Smartphones and Tablets*. URL: https://www.rotoview.com/magnetometer.htm (cit. on p. 18).

[24] Philip Marquardt et al. "(Sp)iPhone: Decoding Vibrations from Nearby Keyboards Using Mobile Phone Accelerometers". In: *Proceedings of the 18th ACM Conference on Computer and Communications Security*. CCS '11. Chicago, Illinois, USA: ACM, 2011, pp. 551–562. ISBN: 978-1-4503-0948-6. DOI: 10.1145/2046707.2046771. URL: http://doi.acm.org/10.1145/2046707.2046771 (cit. on p. 6).

[25] Nikolay Matyunin, Jakub Szefer, and Stefan Katzenbeisser. "Zero-permission acoustic cross-device tracking". In: Apr. 2018, pp. 25–32. DOI: 10.1109/HST.2018.8383887 (cit. on p. 17).

[26] Vasilios Mavroudis et al. "On the Privacy and Security of the Ultrasound Ecosystem". In: *Proceedings on Privacy Enhancing Technologies* 2017.2 (2017), pp. 95–112 (cit. on p. 9).

[27] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. "Gyrophone: Recognizing Speech from Gyroscope Signals". In: *USENIX Security Symposium*. 2014 (cit. on p. 17).

[28] S. Narain et al. "Inferring User Routes and Locations Using Zero-Permission Mobile Sensors". In: *2016 IEEE Symposium on Security and Privacy (SP)*. May 2016, pp. 397–413. DOI: 10.1109/SP.2016.31 (cit. on p. 13).

[29] Emmanuel Owusu et al. "ACCessory: Password Inference Using Accelerometers on Smartphones". In: *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*. HotMobile '12. San Diego, California: ACM, 2012, 9:1–9:6. ISBN: 978-1-4503-1207-3. DOI: 10.1145/2162081.2162095. URL: http://doi.acm.org/10.1145/2162081.2162095 (cit. on p. 16).

[30] Photis Patonis et al. "A Fusion Method for Combining Low-Cost IMU/Magnetometer Outputs for Use in Applications on Mobile Devices". In: *Sensors* 18.8 (Sept. 2018), p. 2616. DOI: 10.3390/s18082616 (cit. on p. 20).

[31] *Permissions overview*. en. URL: https://developer.android.com/guide/topics/permissions/overview (visited on 11/05/2019) (cit. on p. 13).

[32] *Process isolation*. en. Page Version ID: 895996740. May 2019. URL: https://en.wikipedia.org/w/index.php?title=Process_isolation&oldid=895996740 (visited on 11/05/2019) (cit. on p. 8).

[33] Amit Kumar Sikder et al. "A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications". In: *CoRR* abs/1802.02041 (2018). arXiv: 1802.02041. URL: http://arxiv.org/abs/1802.02041 (cit. on p. 17).

[34] Laurent Simon and Ross Anderson. "PIN Skimmer: Inferring PINs Through the Camera and Microphone". In: *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. SPSM '13. Berlin, Germany: ACM, 2013, pp. 67–78. ISBN: 978-1-4503-2491-5. DOI: 10.1145/2516760.2516770. URL: http://doi.acm.org/10.1145/2516760.2516770 (cit. on p. 17).

[35] Daniel Spiekermann, Jörg Keller, and Tobias Eggendorfer. "Towards Covert Channels in Cloud Environments: A Study of Implementations in Virtual Networks". In: July 2017, pp. 248–262. ISBN: 978-3-319-64184-3. DOI: 10.1007/978-3-319-64185-0_19 (cit. on p. 9).

[36] Raphael Spreitzer. "PIN Skimming: Exploiting the Ambient-Light Sensor in Mobile Devices". In: *CoRR* abs/1405.3760 (2014). arXiv: 1405.3760. URL: http://arxiv.org/abs/1405.3760 (cit. on p. 16).

[37] R. Spreitzer et al. "Systematic Classification of Side-Channel Attacks: A Case Study for Mobile Devices". In: *IEEE Communications Surveys Tutorials* 20.1 (2018), pp. 465–488. DOI: 10.1109/COMST.2017.2779824 (cit. on pp. 2, 3).

[38] Martin Vuagnoux and Sylvain Pasini. "Compromising Electromagnetic Emanations of Wired and Wireless Keyboards". In: *Proceedings of the 18th Conference on USENIX Security Symposium*. SSYM'09. Montreal, Canada: USENIX Association, 2009, pp. 1–16. URL: http://dl.acm.org/citation.cfm?id=1855768.1855769 (cit. on p. 7).

[39] Wikipedia contributors. *Electromagnetic attack — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-November-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Electromagnetic_attack&oldid=883886609 (cit. on p. 4).

[40] Wikipedia contributors. *Side-channel attack — Wikipedia, The Free Encyclopedia*. [Online; accessed 4-November-2019]. 2019. URL: https://en.wikipedia.org/w/index.php?title=Side-channel_attack&oldid=914698201 (cit. on p. 1).

[41] Kris Winer. *Simple and Effective Magnetometer Calibration*. URL: https://github.com/kriswiner/MPU6050/wiki/Simple-and-Effective-Magnetometer-Calibration (cit. on p. 19).

[42] Jiexin Zhang, Alastair R. Beresford, and Ian Sheret. "SensorID: Sensor Calibration Fingerprinting for Smartphones". In: *Proceedings of the 40th IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019 (cit. on p. 17).

[43] Li Zhuang, Feng Zhou, and J. D. Tygar. "Keyboard Acoustic Emanations Revisited". In: *ACM Trans. Inf. Syst. Secur.* 13.1 (Nov. 2009), 3:1–3:26. ISSN: 1094-9224. DOI: 10.1145/1609956.1609959. URL: http://doi.acm.org/10.1145/1609956.1609959 (cit. on p. 17).