Defence Science Journal, Vol. 71, No. 4, July 2021, pp. 476-490, DOI : 10.14429/dsj.71.15077


# An Efficient Multistage Fusion Approach for Smartphone Security Analysis

Sumit Kumar[#,*], S. Indu[@], and Gurjit Singh Walia[#]

[#]*DRDO-Scientific Analysis Group, Delhi - 110 054, India*
[@]*Electronics and Communication, Delhi Technological University, Delhi - 110 042, India*
[*]*E-mail: sumitkr@hotmail.com*

**ABSTRACT**

Android smartphone ecosystem is inundated with innumerable applications mainly developed by third party contenders leading to high vulnerability of these devices. In addition, proliferation of smartphone usage along with their potential applications in diverse field entice malware community to develop new malwares to attack these devices. In order to overcome these issues, an android malware detection framework is proposed wherein an efficient multistage fusion approach is introduced. For this, a robust unified feature vector is created by fusion of transformed feature matrices corresponding to multi-cue using non-linear graph based cross-diffusion. Unified feature is further subjected to multiple classifiers to obtain their classification scores. Classifier scores are further optimally fused employing Dezert-Smarandache Theory (DSmT). Strength of suggested model is assessed both qualitatively and quantitatively by ten-fold cross-validation on the benchmarked datasets. On an average of outcome, we achieved detection accuracy of 98.97% and F-measure of 0.9936.

**Keywords:** Android; Static features; Security analysis; Fusion; Smartphone

## 1. INTRODUCTION

Smartphone security analysis primarily deals with the fortification of a smartphone from threats and vulnerabilities posed by the malicious smartphone applications (apps). Hence, smartphone security analysis can mainly be categorised as apps security analysis. Smartphone has now become an essential part of human life due to its myriad of benefits viz. telephony, social networking, banking, e-commerce, messaging, video tele conferencing etc. Notwithstanding, smartphones are based on different platforms (Android, Windows Phone, iOS, and Blackberry OS, Kai OS etc.), Android OS Smartphones have captured more than 85.1% of the market share[1]. With its openness, popularity and over reliance, attacks on Android OS are also engulfing. As the Android platform permits the installation of apps from unconfirmed third-party resources, it creates the circumstances more difficult for smartphone users. Data kept in a smartphone (such as banking credentials, social networking passwords, official documents, contacts etc.) provoke assailants to devise methods to obtain this critical data illicitly by employing Android malwares such as Trojans, backdoors, worm, botnets, spyware, aggressive adware and ransomware.

Generally, malwares are embedded in the popular android apps by repackaging techniques[42] to pass these malicious apps(malapps) as benign apps and making them susceptible to malware attacks and security vulnerabilities. These malapps are produced to accomplish diverse attacks like pilfering personal info, transferring messages without permission, enticing users to malicious sites and posing grave risk to smart phone operators. To elude detection, malapps are continuously evolving with many variants that further takes a formidable challenge to identify them. Also, the hackers are designing the malwares in way to evade the Machine Learning(ML) classifiers[28]. As a result, effective and proficient detection methods are desperately needed to handle the growing complexity of Android malware. To tackle the numerous challenges in Android malapps detection, research fraternity has produced voluminous work in this arena. There are mainly three types of techniques reported in the literature for security analysis of android malwares viz. static analysis[19,44], dynamic analysis[43,44] and hybrid analysis[44]. Static analysis approach detects android malwares in mobile apps without executing them by simply examining the disassembled source code segments. In dynamic analysis, smartphone app is executed and the behaviour of the app is monitored to extract the dynamic features for app identification. Further, the amalgamation of static and dynamic analysis results in hybrid analysis which basically combines the static and dynamic features for app identification.

Static analysis for android malapps detection does not require a host system environment as the apps are not executed. It is also the most economical, proficient and accurate method for investigating the apps. The numerous static features[18] like permissions, app components, filtered intent, API calls etc. are reported in the literature. These features are extracted by disassembling the apps by APK tool[19]. Permission usage was extensibly exploited for development of solution for android malware detection. Investigations grounded on intents and permissions of applications are susceptible to false positive as benign applications too need sensitive permissions making

Received : 13 September 2019, Revised : 24 August 2020
Accepted : 25 February 2021, Online published : 01 July 2021

them misclassified as malapps. Techniques built on only API calls[44] frequency are inept to create connections amid the API calls to develop the sophisticated behavioural semantics of apps, leading to poor detection rate of novel malapps. Therefore, choosing multiple complementary features plays a significant part in effective detection of malwares.

In this paper, we propose an android based Smartphone Security Analysis paradigm using non-linear graph fusion and optimal fusion of classifier(s). Multiple complementary features are deduced through extensive investigation of benchmarked datasets. Complementary features are fused through cross-iterative graph diffusion. Thus a unified feature is generated and fed to optimal classifier for classifying the apps into benign or malicious with high detection accuracy. Outcomes of our results demonstrates that proposed method has better performance in classification and detection accuracy.

In a nutshell, this manuscript has the following key contributions:

(i) We suggest a static feature approach for smartphone security analysis that incorporates multiple feature unification through cross iterative diffusion. To our awareness, it is the first time that this approach is introduced to extract unified android static features.

(ii) Pragmatic and effective app security analysis framework is proposed wherein three ML algorithms are exploited to evolve a system to detect the malapps on the basis of unified feature representation. Further, outcomes of the ML algorithms were fused by DSmT[16] algorithm to improve the accuracy achieved by individual classifiers. In addition, we presented a complete investigational study based on CICMalDroid2020[41], AMD[40] and Drebin[15] malapps database and comparative experimentations with state-of-the-art methods to validate the efficiency and proficiency of our approach.

The rest of the research study is orchestrated as follows: Section 2, deliberates the smartphone security analysis related work in identifying malapps by static analysis approach. In Section 3, we expound our proposed framework for android malware detection. Section 4 discusses about evaluation of framework i.e. about experimental design along with the details of the datasets used for training and testing and experimental validation of suggested framework using both qualitative and quantitative measures. In the Section 5, we confer some limitations of the proposed method and concluding remarks along with future directions for the proposed work are highlighted under Section 6.

## 2. RELATED WORK

Contemporarily, there have been plenty of investigations in smartphone security analysis, which can be roughly classified as dynamic, static and hybrid analysis[19,43,44]. Although, review on android smartphone based security analysis were reported in various papers[19,43,44]. Comprehensive review of recent work in the field of static analysis does not exist. Hence, in this section, we have gleaned the important insights from the different research papers based on the static analysis approach which will help the reader to gauge the gap in the research. All these work can be categorised into two categories as follows:

### 2.1 Multiple Features and Single Classifier

Significant permission identification based analysis considered by li[2], *et al.* for detection of malwares. In this, 22 significant permissions were identified and SVM algorithm was used for the classification. For known and unknown malwares, detection accuracy of approximately 93.62% and 91.4% respectively was achieved. Ensemble Rotational Forest based model[3] was proposed that exploits permissions, permission rate, sensitive API's etc. as key features to detect malware with accuracy of 88.26%. Significant pairings of the permissions[4] extracted from apps that can be threatening were identified leading to a malicious application detection with approx. accuracy of 95.44%. approximately. For this, datasets were analysed intricately and edge weights are allocated to pairs of permission depending on their frequency of occurrence in the datasets. MalPat[6] was realised by extracting highly sensitive permission linked APIs for detecting malware with a high F1 score of 98.24% using Random Forest(RF) algorithm.

### 2.2 Multiple Features and Multiple Classifiers

An ensemble[5] of three detection models based API frequency, API calls and API sequence was created to achieve detection accuracy of 98.98%. An automated malware app detection tool[7] with unique ensemble learning method using permissions and API calls with Naïve Bayes, Decision Tree(DT), Random Tree(RT) etc. to detect malwares was reported with the detection rate of 99% (approx.) with very low false positives. Authors[10] generated a feature vector that represents malware features having same attributes with benign applications. In this, model learning approaches and automatic upgrade system for malapps detection using a multimodal deep learning method was proposed with accuracy of 98%. Classification model FalDroid[11] based on shared features automatically classifies android malware with detection accuracy of 94.2% using frequent subgraphs. Wang[12], *et al*. used multiple features and ensemble of classifiers viz. KNN, SVM, NB, CART and RF for android based malapps detection through majority vote fusion method. Both string features and structural features were exploited in Droid Ensemble[13] and three ML algorithms viz. Support Vector Machine(SVM), K-Nearest Neighbour(KNN) and RF were used to achieve detection accuracy of 98.4%. A four layered static detection model using MD5, malevolent permissions, dangerous permissions and intents was proposed by Song[14], *et al.*. Authors[20] used API calls abstraction method to decrease the number of API calls be used as a feature and three ML algorithms (KNN, RF and SVM) were used for achieving detection with 98% accuracy. A novel method[21] for identifying android based malapps to automatically detects malware by extracting the multiple static features such as API calls, permissions, network addresses, and mapped these features into a single feature space vector. Further, Linear SVM and DT algorithms were employed to implement the multi-label classification. Detection accuracy of 98% and 63% was achieved for small size families' malware and zero day malwares respectively. In MalResLSTM[22], authors presented Long Short Term Memory(LSTM) based method to classify malapps. Feature extracted were mapped to vector space and processed in the LSTM based deep learning model to achieve the

accuracy of 99.32%. Oluwafemi Olukoya[23], *et al.* investigates a malware detection model based on sensitive permissions and UI based app descriptions. Investigational outcomes establish precision of 90%. DroidDomTree[24] excavates the tree structure of API calls in Android apps for identifying malapps. A tree structure of API calls accentuates a path flow and recognizes the layout of APIs and hence stresses the prominence of some APIs in an app. It accomplished detection rates varying from 98.1% to 99.3% using eight different classifiers (J48, AD Tree, RF, RT, AdaBoost, Naïve Bayes, Radial Basis Function Network, KStar). Although, most of the work exploited either multiple classifiers or multiple features for development of solution for smartphone security analysis, a comprehensive solution exploiting both optimal combination of classifier and efficient fusion of multiple features was not investigated.

In another dimension, model based android malware detection approach were also investigated. For instance, Roni Mateless[25], *et al.* presented a model for malapp detection with 97.8% detection accuracy. Decompiled source code contains API calls, keywords, function names, strings in human format etc. Malevolent codes vary from the text because of the syntax rules of compilers and to prevent detection. NLP method was adapted here to classify the apps. Tian[26], *et al.* investigates a method to detect the repackaged apps by code heterogeneity analysis. Code structure was divided into various subset and each subset was classified based on the features. Each subset depicts dependence based region. In this partition based detection, False Positive(FP) rate of 2.97% and False Negative(FN) rate of 0.35%were obtained. In MAMADROID[27], a Markov chain based behavioural model for detecting the android malware. Sequences of API calls were modelled as Markov Chain. Model has achieved the F-measure of 0.87. Han[28], *et al.* proposed malicious app detection scheme by using irretrievable feature transformations so that the evading of ML classifiers by hackers becomes impossible. However, the most of these methods could not achieve high accuracy and lack adaptivity.

In sum, survey of the closely linked literature revealed that most of the above approaches used traditional classifiers which can detect malapps using one or more classifiers. Multiple classifiers using multiple features gives improved overall performance in comparison to the single classifier using multiple features. Also, most of work either focussed on optimal combination of features or optimal combination of classifiers. Hence, future direction of smartphone security analysis is to take benefit of both feature-level and score-level fusion. Therefore, we proposed a unified feature based on cross iterative diffusion of eight complementary feature and optimum fusion of three ML algorithms to enhance the detection accuracy. Minutiae of suggested method described in the following section.

## 3. PROPOSED FRAMEWORK FOR SMARTPHONE SECURITY ANALYSIS

In this manuscript, multistage fusion model wherein both feature and scores are optimally combined is proposed to achieve highly robust Android malware detection. Overview of the proposed framework is depicted in Figure 1. For this, we have designed three modules viz. multi-cue feature extraction, feature unification, and optimal classifier fusion to achieve efficient malware detection. For this, semi-automated tool (taking the aid of APK tool) was made to extract the features from the decompiled Android Package Kit. APK tool decompresses, **.apk* files into **.dex* and AndroidManifest. xml. Features extracted consists of API calls, Permission, Intents, App Components, Native Code, Op Code, Hardware Feature, Network Address. Each feature is exposed to decision tree learning for generating corresponding similarity matrices.

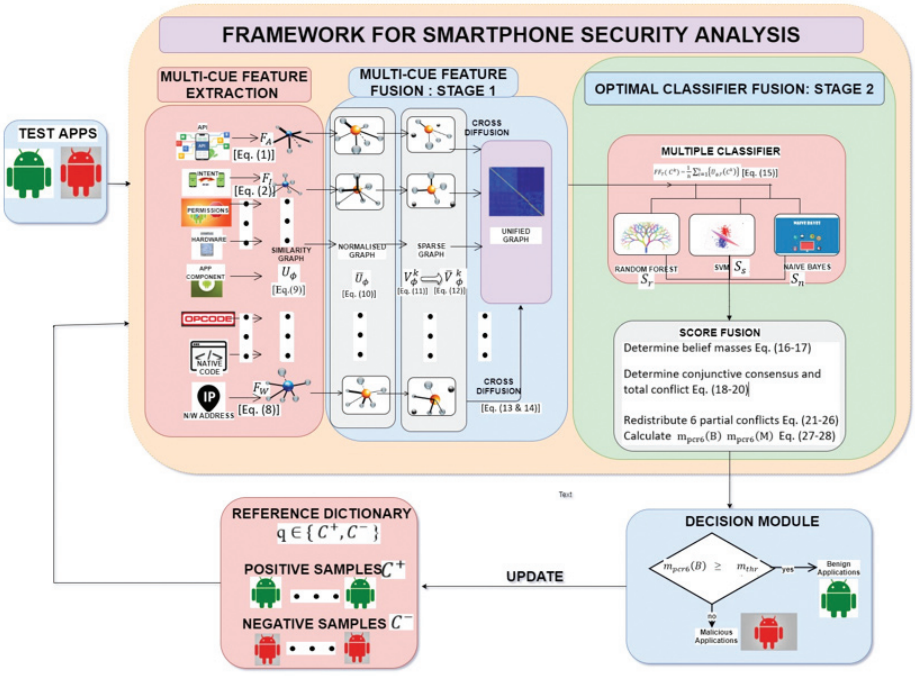


**Figure 1. Proposed smartphone security analysis framework. Stage1 extract complementary information from eight multi-cue to obtain unified feature. Unified feature is furtherclassified using optimal fusion of three classifiers at stage 2.**

These matrices are subjected to normalisation procedure accompanied by filtering out the most weighted similarities to generate the sparse matrices. To unify these features, unified-graph is created via graph fusion technique based on cross-iterative diffusion. This method of enhances the robust connections and filters out the weaker ones. Thus, unified set of features are generated for further classification. Unified feature is further subjected to multiple classifier to make final decision about app classification.

Due to multitude of features embedded in the android apps, single ML algorithm presents its inability to classify these apps effectively[43]. Hence, more than one classification algorithms are exploited to detect complex malapp. In the proposed framework, we have chosen three ML algorithms viz. Random Forest(RF)[32], Support Vector Machine(SVM)[31] and Naïve Bayes(NB)[33] for classification of android apps. Also, we proposed optimal combination of these classifier to accurately classify applications into two classes, namely, benign and malicious. Classification algorithms are chosen to compensate the demerits of the individual classifiers. For instance, RF is usually used when there are more number of features than observations. Its performance is excellent in spite of having noise in the predictor variables and it is also not vulnerable to overfitting. Also, RF classifier is preferred for large dataset as it is not susceptible to outliers. On the limited dataset, SVM performance is optimal for two class problem where the data is outlier. For small dataset, NB performs optimally. It is simple and speedy to give classification results. In proposed classification solution, three classifiers complements each other in framework and synergised the performance of resultant classifier. Our approach exploits three classifiers in parallel and the output scores of all the classifiers are fused to synergised the overall performance for detection. Respective classifier scores viz. $S_r$ for RF, $S_s$ for SVM, $S_n$ for NB are further transformed into belief masses using Shafer Model[17]. Masses for the three class focal elements are optimally combined using PCR-6 Rules[17], where classifier's conflicting mass is redistributed in proportion to the mass which is contributing to the conflict. Finally, in the decision model, belief mass $m_{pcr6}(B)$ is compared with the threshold value $m_{thr}$ and test app is classified into benign depending on whether $m_{pcr6}(B)$ is greater than or equal to the $m_{thr}$ or malign otherwise. Details of the proposed Android malapp detection framework is presented in the next sub-sections.

### 3.1 Multi-cue Feature Fusion

Multi-cue feature fusion process consists of multiple feature extraction and their fusion using cross diffusion of extracted features. Multiple features are extracted for achieving the high performance. Cross diffusion of features extract complementary information to obtain highly distinct unified feature leading to creation of clear and distinct boundary between benign and malicious class.

In the proposed framework, we have modelled multi-cue feature fusion problem as eight graphs and fused them by iterative cross diffusion process.

#### *3.1.1 Multi-cue Feature Extraction*

Multiple features are extracted for given test app $t$ along with apps from reference dictionary, $q \in \{C^+, C^-\}$ , $C^+$ corresponds to benign apps and $C^-$ corresponds to malicious app. Reference dictionary apps are updated with time so as to incorporate the new apps in the proposed framework to enhance its detection capability. In the proposed approach, we have extracted eight features namely, API calls, Permission, Intents, App Components, Native Code, Op Code, Hardware Feature, Network address using from semi-automated tool.

Description of the eight extracted feature types are as follows:

*APIs:* Android OS has many APIs (Application Programming Interface) that are used for interacting with Android smartphones. Malwares extensively used APIs to target the Android ecosystem. API's are present in the **.dex* class of an app and can also be found in the Smali Files of the APK. By extensive analysis of the dataset, we have chosen $k1$ number of API's listed in Table 1, whose frequency of occurrence is taken as a feature value. Feature value is determined using Eqn (1)

**Table 1. Details of API, permission and intent feature**

| Feature | Position of Feature (*i*) | Symbol |
|---|---|---|
| API | AutoSmsReceiver, BootReceiver, PhoneCallReceiver, abortBroadcast, GetCall state, getActiveNetworkInfo(),getDataActivity(),getDeviceId(),getNetworkType(), getSimOperator(), getSimSerialNumber(), getSimState(), getSubscriberId(), classes.dex, entry.loadClass(),getConnectionInfo(), getSupplicantState(), setWiFiEnabled(), execHttpRequest(), Runtime.exec(), Cipher.GetInstance(), sentTextMessage(), getMessageBody(), getSubscriberID(), getLastKnownLocation(), com.android.contacts() | $\varphi_{1i}$ |
| Permission | Access_Network_State, set_Prefered_Application, Access_Wi-Fi_State, Access_Fine_Location, Call_Phone, Change_Network_State, Get_Accounts ,Internet, Install_Packages, read_Contacts, Read_Logs, Read_Phone_State, Read_Sms, Receive_Boot_completed, Restart_packages, Receive_Sms,Send_Sms, Vibrate, Write_Secure_Settings, Read_History_Bookmarks, Update_Device_stats, Manage_Documents, Install_Location_Provider | $\varphi_{2i}$ |
| Intents | Boot_Completed, Send_To, Dial, Screen_off, Text, Send, User_Present, Screen_On, Call, Package_Data_Cleared, Text, Send, Quickboot_Poweron, Time_Changed, Sms_Received, Airplane_Mode, Battery_Changed Get_Content, Data_Sms_Received | $\varphi_{3i}$ |

$$F_A^t = \sum_{i=1}^{k1} f(\varphi_{1i}) \quad (1)$$

where, $f(\varphi_{1i})$ is a function that determines the frequency of occurrence of API, $\varphi_{1i}$ denotes API positioned at $i^{th}$ place in Table 1 and $F_A^t$ is the API related feature vector of the test app $t$ . Similarly, API feature $F_A^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$ .

*Permissions:* Android apps requests for permissions from smartphone user during the app installation. Permissions are essentially required to protect the privacy of the users making the permissions the most vulnerable conduit for launching attacks in the android smartphones. Arora[4], *et al.* exploited numerous permissions for malware identification. Permissions are stored in Manifest.xml file of the app source code. In our model, we have taken frequency of occurrence of most risky permission's request also as a feature vector. For this, $k2$ number of permissions are chosen considering their frequency of call by various malicious apps. Feature vector related to permission is determined using Eqn (2).

$$F_P^t = \sum_{i=1}^{k2} f(\varphi_{2i}) \quad (2)$$

where, $f(\varphi_{2i})$ is a function that determines the frequency of occurrence of most frequent permission , $\varphi_{2i}$ denotes such permission positioned at $i^{th}$ place in Table 1 and $F_P^t$ is the permission related feature vector of the test app $t$ .Similarly, permission feature $F_P^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$ .

*System Intents:* Intent[36] is basically a message used to kick start activity in apps. Starting a service and activity and delivering a broad cast are three basic usage of intents. Malware writers are exploiting intents for launching numerous attacks. We have chosen $k3$ number of intents (listed in Table 1) that are widely used to segregate the malapps from benign apps. Therefore, Intents are taken as feature parameter. Total count of these intents is determined using Eqn (3)

$$F_I^t = \sum_{i=1}^{k3} f(\varphi_{3i}) \quad (3)$$

where, $f(\varphi_{3i})$ is a function that determines the frequency of occurrence of Intents $\varphi_{3i}$ , positioned at $i^{th}$ place in Table 1 and $F_I^t$ is the intent related feature vector of $k3$ number of intents of app $t$ .Similarly, Intent feature $F_I^q$ for initially stored apps are extracted for $q \in \{C^+, C^-\}$ .

*APP Component:* App components characterise applications and they are the conduits though which the user or system accesses an app. These components are related by the app's manifest file AndroidManifest.xml that describes the components of an app and dictates the interaction mechanism. There are 4 main app components lying in Android app i.e. Service, Activity, Broadcast Receiver, and Content Provider in the app's manifest file and frequency of these app components are taken as feature parameter and is determined using Eqn (4)

$$F_C^t = f(AppComponent) \quad (4)$$

$F_C^t$ is the feature value for test app $t$ . Similarly, app component feature $F_C^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$ .

*Native code:* Native code is processor specific code and does not run on the emulator. It is used to hide malicious content in the app as this code is difficult to understand. Therefore, another feature parameter is the total sum of these native codes in an application. Native code feature parameter for app $t$ is calculated using Eqn (5)

$$F_N^t = f(NativeCode) \quad (5)$$

Similarly, native code feature $F_N^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

*OP Code:* An opcode or Operation Code is a machine language instruction that stipulates the operation to be performed with CPU. Frequency of sequence of opcodes extracted from the apps can be taken as features for malapp identification. Opcodes are exploited by the malware writers because of their similarity to app code and frequency of these op codes are taken as the next feature parameter and is determined using Eqn (6)

$$F_O^t = f(OPCode) \quad (6)$$

$F_O^t$ is the feature value for test app *t*. Similarly, op code feature $F_O^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

*Hardware Feature:* Hardware features[37] are used by Android apps to access hardware of the android smartphone and are listed in the AndroidManifest.xml file. Hardware features are characterised by "android. hardware" in the manifest file. In the proposed framework, by extensive analysis of dataset, we have chosen55 hardware feature for generating feature parameter. The frequency of 55 hardware features in the manifest.xml file of an app is taken as feature parameter and is determined using Eqn (7)

$$F_H^t = f(android.hardware) \quad (7)$$

$F_H^t$ is the feature value for test app $t$ . Similarly, hardware feature $F_H^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$

*Network Addresses*: Malapps designers often wants to interact with malapps so as to direct the user's critical data on the smartphone to designated network addresses of the C&C server embedded in malapps. So, network address can be taken as the feature parameter. The total number of these network addresses in an application is the network address based feature parameter for test app *t* is calculated by Eqn (8)

$$F_W^t = f(network\_address) \quad (8)$$

$F_W^t$ is the feature value for test app $t$ . Similarly, network address feature $F_W^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$ .

In sum, we have constructed eight feature descriptor as mentioned in Eqn (1) to Eqn (8) for every test application and reference dictionary apps. Similarly, network feature $F_W^q$ for reference dictionary apps are extracted for $q \in \{C^+, C^-\}$ . Feature vectors extracted using Eqn (1) to Eqn (8) are further fused to obtain unified feature. A set of $C^+$ number of benign and $C^-$ number of malicious apps are stored as a reference dictionary. This set is updated with the advent of the new apps being subjected for analysis. In feature unification, features for test and reference dictionary apps are extracted and subjected

for creation of non-linear graph. In the graph, test app feature act as one node and reference dictionary apps as other nodes. Following this, eight graphs are generated for each test app $t$.

For each feature descriptor $F_\phi^t \in \{F_A, F_I, F_P, F_C, F_N, F_O, F_H, F_W\}$ of test app $t$, we construct graphs $U_\phi^t \in \{U_A, U_I, U_P, U_C, U_N, U_O, U_H, U_W\}$ using an edge weight described as the similarity between feature descriptors of two apps $t$ and $q$ where $q \in \{C^+, C^-\}$. In this, similarity matrices $U_\phi \in \mathbb{R}^{N*N}$ is constructed by decision tree, where $N = q+1$. For feature pair values $(F_\phi^t, F_\phi^q)$ corresponding to $t$ and $q$ apps for $\phi^{th}$ feature, similarity parameter $U_\phi(t,q)$ is calculated by passing them through decision trees using Eqn (9)

$$U_\phi(t,q) = \frac{f((L(F_\phi^t)) \cap (L(F_\phi^t)))}{T_t}, \phi \in [A, I, P, C, N, O, H, W] \tag{9}$$

$L$ above is the set of class labels of trees grown and $T_t$ is the total number of decision trees made. Graph generated using Eqn (9) are further fused using proposed cross diffusion to achieve unified feature. Details of features unification follows in turn.

### *3.1.2 Multi-cue Feature Unification*

Multi-cue features extracted from decompiled source code may not be linearly associated and need non-linear based fusion technique to combine this complementary info. To integrate multi-cues efficiently, non-linear graph based cross-diffusion process was introduced by Wang[29], *et al.* Further, improved version[30] of this work[29] was explored for classification. In this work, complimentary info from multi-cue data is extracted and non-linear unified graph was generated by cross diffusion process. Classification results[30] demonstrates that the multi-cue information unification by non-linear graph method is more precise than linear graph methods. Cross diffusion approach proposed by Walia[38], *et al.* is employed for feature fusion in the proposed framework. This method is better than previous methods[29,30] and improves the detection accuracy because of the iterative normalisation of similarity matrix and updated sparse representation. Unique graph unification approach accomplishes non-linear feature fusion with iterative normalisation. This keeps a robust representation of the apps (malicious or benign) and rejects weak features that make the classifier vulnerable to unreliable results.

Similarity matrix generated using Eqn (9) for each feature graph is again normalised using Eqn (10) to obtain respective normalised matrices $\overline{U}_\phi^t, \phi \in \{A, I, P, C, N, O, H, W\}$. Normalisation technique sets the similarity of each app with itself as constant $\varepsilon$, and the similarities with rest of the apps in the test set to (1- $\varepsilon$). The first row of the $\overline{U}_\phi^t$ comprises the edge weights respective to test app $t$

$$\overline{U}_\phi(t,q) = \begin{cases} (1-\varepsilon)\dfrac{U_\phi(t,q)}{(\sum_{q=1}^{N} U_\phi(t,q)}, t \neq q \\ \varepsilon, t = q \end{cases} \tag{10}$$

$\overline{U}_\phi(t,q)$ above is further used to derive a sparse vector depiction of the training app $t$ to keep the most similar features and discard the other using Eqn (11)

$$V_\phi^k = \begin{cases} U_\phi(t,q), if\, \overline{U}_\phi(t) \in K - NN(t) \\ 0, otherwise \end{cases} \tag{11}$$

We further normalised $V_\phi^k$ as $\overline{V}_\phi^k$ vector using Eqn (12)

$$\overline{V}_\phi^k = \frac{V_\phi^k}{\sum_{i=1}^{N+1} V_{\phi,i}^k} \tag{12}$$

Normalisation further allocates the weights amongst the strong links giving robust sparse depiction. Fused feature descriptor $FF_T(t) \in \mathbb{R}^{N+1}$ is obtained by fusing the different feature sparse vectors $\overline{V}_\phi$ by cross diffusion approach, where $h$ is the number of features taken.

$$U_{\phi,i+1}(t) = \overline{V}_{\phi,i}^k * \left( \frac{1}{h-1} * \sum_{j=1, j\neq\phi}^{h} \{\overline{U}_\phi(t)\} \right) * \left( \overline{V}_{\phi,i}^k \right)^{transpose} \tag{13}$$

where, $i$ is the $i^{th}$ iteration of cross diffusion process and $h$ =8

To enhance the effectiveness of the diffusion , modification of the recursive operation by normalisation of each row respective to the test app in the similarity matrix $U_{\phi,i+1}(t) \in \mathbb{R}^{N+1}$ obtained after every iteration as

$$\overline{U}_{\phi,i+1}(t) = \frac{U_{\phi,i+1}(t)}{\sum_{j=1}^{N} \left| \{U_{\phi,i+1}(t)\} \right|} \tag{14}$$

Following this, normalised sparse vector $\overline{V}_{\phi n,i+1}$ is obtained from sparse vector $V_{\phi n,i+1}$ in the next iteration using Eqn(14), Eqn(11) and Eqn(12). Lastly, the mean of adjacency list of test app for each feature descriptor $\overline{U}_{\phi,T}(t)$ is taken to find the fused feature descriptor $FF_T(t)$ as given in Eqn (15)

$$FF_T(t) = \frac{\sum_{j=1}^{8} \left( \overline{U}_{\phi[j],T}(t) \right)}{8} \tag{15}$$

where, $T$ is the final iteration of the normalisation process of cross diffusion and $\phi \in \{A, I, P, C, N, O, H, W\}$ .This $FF_T(t)$ is taken as a unified feature and given as input to train the classifier(s). Detail of unified feature classification for test app follows in next subsection.

## 3.2 Optimal Classifier Fusion

Unified feature for the test app is applied to classification module for final decision. For this we subjected the unified feature to three trained classifiers in parallel to determine their classification scores. Proposed classification model comprises of two phases viz. individual training classifier score estimation and optimal combination of the individual classifier scores. For classification of test app, three classifier scores viz. Random Forest $S_r$ and Support Vector Machine $S_s$ , Naïve Bayes $S_n$ are determined when unified feature is fed to these individual trained classifier. These classifiers scores can be combined by various score fusion approaches[16,35,45] available in the literature. In the proposed method, respective classifier scores attained are

further subjected to classifier score fusion paradigm where the obtained scores from the classifiers are converted to respective belief masses and the conflicts amongst individual belief is redistributed and resolved by means of DSmT based PCR-6 rules[16].

Scores of the different classifiers are fused using Shafer's model. For this, the frame of discernment $\left(F_r = \{B, M\}\right)$ is specified by of two focal elements viz. Benign $(B)$ and Malicious $(M)$ corresponding to whether the app is benign or malicious. Each classifier in the model delivers a score about classification. The individual belief mass is obtained by transforming the classifier score $\left(S_r, S_s, S_n\right)$ with the aid of Denoeux Belief System[17] using equations (16) and (17):

$$m_j(B) = C_j * S_j(B) \tag{16}$$

$$m_j(M) = 1 - C_j * S_j(B) \tag{17}$$

Where $j \in \{r, s, n\}$ and $C_j$ is the confidence factor of individual classifier. Further, belief masses are optimally fused by means of DsmT-based PCR-6 rules. For this conjunctive consensus is determined using Eqn (18) and Eqn (19):

$$m_{rsn}(B) = \prod_{j=1}^{3} m_j(B) \tag{18}$$

$$m_{rsn}(M) = \prod_{j=1}^{3} m_j(M) \tag{19}$$

Where $j \in \{r, s, n\}$ and $B, M$ corresponds to benign and malicious app respectively.

Total conflict amongst classifiers is obtained which consists of partial conflicting masses of benign and malicious scores using Eqn (20):

$$\begin{aligned} m_{rsn}(B \cap M = \varnothing) = {} & m_r(B) * m_s(M) * m_n(M) + m_r(M) * m_s(B) * m_n(M) \\ & + m_r(M) * m_s(M) * m_n(B) + m_r(M) * m_s(B) * m_n(B) \\ & + m_r(B) * m_s(B) * m_n(B) + m_r(B) * m_s(B) * m_n(M) \end{aligned} \tag{20}$$

Total conflict comprises of six number of partial conflicts which are further reallocated amongst benign and malicious scores using (21-26) equations, where $b1$ to $b6$ are redistributed conflict masses for the benign focal element and $m1$ to $m6$ are redistributed conflict masses for the malicious focal element respectively.

$$\frac{b1}{m_r(B)} = \frac{m1}{m_s(M) + m_n(M)} = \frac{m_r(B) * m_s(M) * m_n(M)}{m_r(B) + m_s(M) + m_n(M)} \tag{21}$$

$$\frac{b2}{m_s(B)} = \frac{m2}{m_r(M) + m_n(M)} = \frac{m_r(M) * m_s(B) * m_n(M)}{m_r(M) + m_s(B) + m_n(M)} \tag{22}$$

$$\frac{b3}{m_n(B)} = \frac{m3}{m_s(M) + m_r(M)} = \frac{m_r(M) * m_s(M) * m_n(B)}{m_r(M) + m_s(M) + m_n(B)} \tag{23}$$

$$\frac{b4}{m_s(B) + m_n(B)} = \frac{m4}{m_r(B)} = \frac{m_r(M) * m_s(B) * m_n(B)}{m_r(M) + m_s(M) + m_n(B)} \tag{24}$$

$$\frac{b5}{m_r(B) + m_n(B)} = \frac{m5}{m_s(M)} = \frac{m_r(B) * m_s(M) * m_n(B)}{m_r(B) + m_s(M) + m_n(B)} \tag{25}$$

$$\frac{b6}{m_r(B) + m_s(B)} = \frac{m6}{m_n(M)} = \frac{m_r(B) * m_s(B) * m_n(M)}{m_r(B) + m_s(M) + m_n(M)} \tag{26}$$

The final belief regarding whether the test app is benign or malign is derived by adding the redistribution masses and corresponding conjective consensus using equations Eqn (27) and Eqn (28)

$$m_{pcr6}(B) = m_{rsn}(B) + b1 + b2 + b3 + b4 + b5 + b6 \tag{27}$$

$$m_{pcr6}(M) = m_{rsn}(M) + m1 + m2 + m3 + m4 + m5 + m6 \tag{28}$$

The final belief whether test app $t$ is benign or malicious is determined from by $m_{pcr6}(B)$ or $m_{pcr6}(M)$ . Thereafter, decision is taken by comparing the final beliefs with a threshold value. If value of $m_{pcr6}(B)$ is greater than or equal to the threshold ( $m_{thr}$ )value, then test app $t$ is declared as benign otherwise it is declared as malicious. *Algorithm1* sum up the pseudocode for proposed framework for Smartphone Security Analysis. In the next section, performance evaluation of proposed method against other state-of-the-art malware analysis methods follows.

**Algorithm 1: Proposed Smart Phone Security Analysis**

**Function: Security Analysis** $S(t, C^+, C^-)$

**For** $\left(C^k \in C\right)$ do

S $\leftarrow [C^k, C^+, C^-]$

derive $F_\phi^t \in \{F_A, F_I, F_P, F_C, F_N, F_O, F_H, F_W\}$ from Eqn(1-8)

**for** $F_\phi$ do

Derive $U_\phi^t \in \{U_A, U_I, U_P, U_C, U_N, U_O, U_H, U_W\}$ from Eqn (9)

normalise $U_\phi$ to $\overline{U}_\phi$ using Eqn (10)

**if** $\overline{U}_\phi(t) \in k - NN(t)$ then

$V_\phi^k \leftarrow U_\phi(t, q)$

**else**

$V_\phi^k = 0$

**end**

normalise $V_\phi^k$ to $\overline{V}_\phi^k$ using Eqn(12)

**repeat**

**find** $U_{\phi,i+1}(t)$ using $\overline{V}_{\phi,i}^k$ and $\overline{U}_\phi(t)$ from Eqn (13)

normalize $U_{\phi,i+1}(t)$ to $\overline{U}_{\phi,i+1}(t)$ using Eqn(14)

$\overline{U}_{\phi,i}(t) := \overline{U}_{\phi,i+1}(t)$

**until convergence**

**end**

find $FF_T(t)$ using $\overline{U}_{\phi,T}(t)$ Eqn(15)

find $\left(S_r, S_s, S_n\right)$ using $FF_T(t)$

find $m_j(B)$ and $m_j(M)$, $j \in \{r, s, n\}$ using Eqn (16) and Eqn(17)

find $m_{pcr6}(B)$ using Eqn(18) to (26)

find $m_{rsn}(B)$, $m_{rsn}(M)$ and $m_{rsn}(B \cap M = \varnothing)$ from Eqn(18-20)

find $m_j$ and $b_j$ from Eqn(21-26) for $j = 1,2,3,4,5,6$

find $m_{pcr6}(B)$ using Eqn(27)

**if** $m_{pcr6}(B) \geq m_{thr}$

return (benign)

**else**

return (malicious)

**end**

## 4. EXPERIMENTAL RESULTS AND DISCUSSION

Experimental validation includes both qualitatively and quantitatively evaluation of proposed framework on the chimeric datasets as mentioned in Table 2. Qualitative evaluation is done through statistical investigation of extracted features of datasets and score-distribution of the classifiers. Also, quantitative analysis is done by numerous performance matrices viz. Accuracy, Decidability Index(DI), Equal Error Rate (EER), F1 Score and sensitivity. We also compared our proposed framework with four state-of-art methods employing static features HEMD[8], MLIF[9], DS[35] and FGF[39]. The details of the experimental validation follow in turn.

### 4.1 Datasets

For evaluation of proposed framework, Database (DB1 to DB5) comprising of both benign (B) and malign(M) apps datasets is formulated. Benign apps are acquired mainly from Google Play Store and CICMalDroid2020. After downloading the benign apps, we subject them through online Virus-Total scanner that has about 70 antivirus scanners in its arsenal. Application is tagged as benign if the antivirus scanner recognised it as benign, else it is considered as malign or malicious. Malicious apps are collected from benchmarked datasets viz. Drebin[15] and AMD[40] and CICMalDroid2020[41] that covers the diverse families of malware. In total, 4000 apps are collected and rearranged in the form datasets (DB1 to DB5) which is detailed in Table 2. First, four group(DB1-DB4) of 1000 apps each from the benign and malicious apps is created and consolidated group of all the 4000 apps is named as DB5.

Further, evaluation of the framework was performed on MATLAB 2017b on an i7,2.2 GHz processor having 16 GB RAM to implement proposed framework. In DB1-DB5, we split the dataset of apps into ten equal subsets and select a subset of apps randomly for testing and left over subsets is used as training apps. To overcome over fitting of results, 10 fold cross-validation technique is used and mean values are reported as results. Next section covers the experimental validation where the proposed framework is analysed both in terms of qualitative and quantitative analysis. Particulars of Qualitative analysis follows in the subsequent sub-section.

### 4.2 Qualitative Analysis

Qualitative performance is evaluated for the proposed framework over the datasets. Qualitative performance is mainly done by comparing the score distribution analysis of different state-of-the-art techniques and frequency analysis of extracted features on the datasets. Qualitative analysis results are deliberated as follows:

*Frequency Distribution Analysis:* Frequency of occurrence of eight complementary features are determined for different datasets. For consolidated dataset DB5, extracted features viz. API calls, Permission, Intents, App Components, Native Code, Op Code, Hardware Feature and Network Address are plotted as bar charts as shown in Fig. 2, for benign and malicious apps, wherein benign apps total feature values are presented in blue colour and malapps total feature values are depicted as red colour bar. From the bar graph, it is apparent that the eight chosen features are discriminative and hence provides a great performance regarding the classification.

*Score Distribution:* To evaluate the proposed optimal classifier performance, score scatter distribution plots are examined for both benign and malicious apps. The outcomes corresponding to DB1 database are shown in Fig. 3. Scores for benign and malicious apps are determined and plotted against app number resulting in the scatter plot as shown in

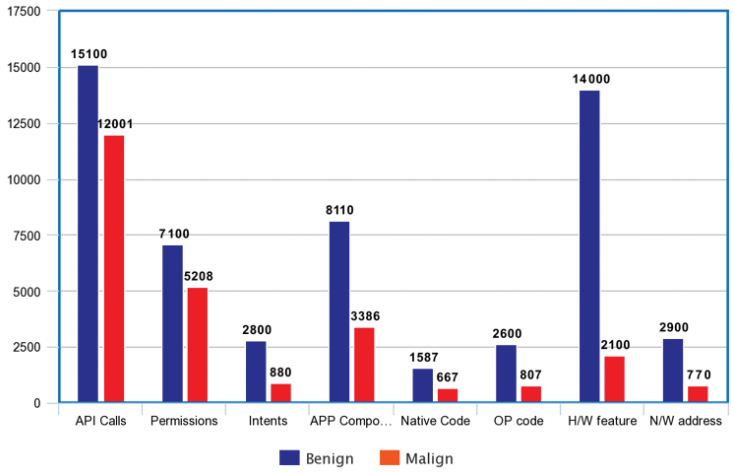


**Figure 2. Frequency distribution analysis for extracted eight features for database DB5.**

**Table 2. Databases for experimental validation**

| App Category / Database | Malign Apps(M) | Benign Apps(B) | Remarks |
|---|---|---|---|
| **DB1** | 500 | 500 | Drebin(M) GooglePlay(B) |
| **DB2** | 500 | 500 | AMD(M) GooglePlay(B) |
| **DB3** | 500 | 500 | CICMalDroid2020 (for both M&B) |
| **DB4** | 500 | 500 | AMD(M) CICMalDroid2020(B) |
| **DB5** | 2000 | 2000 | Consolidated |

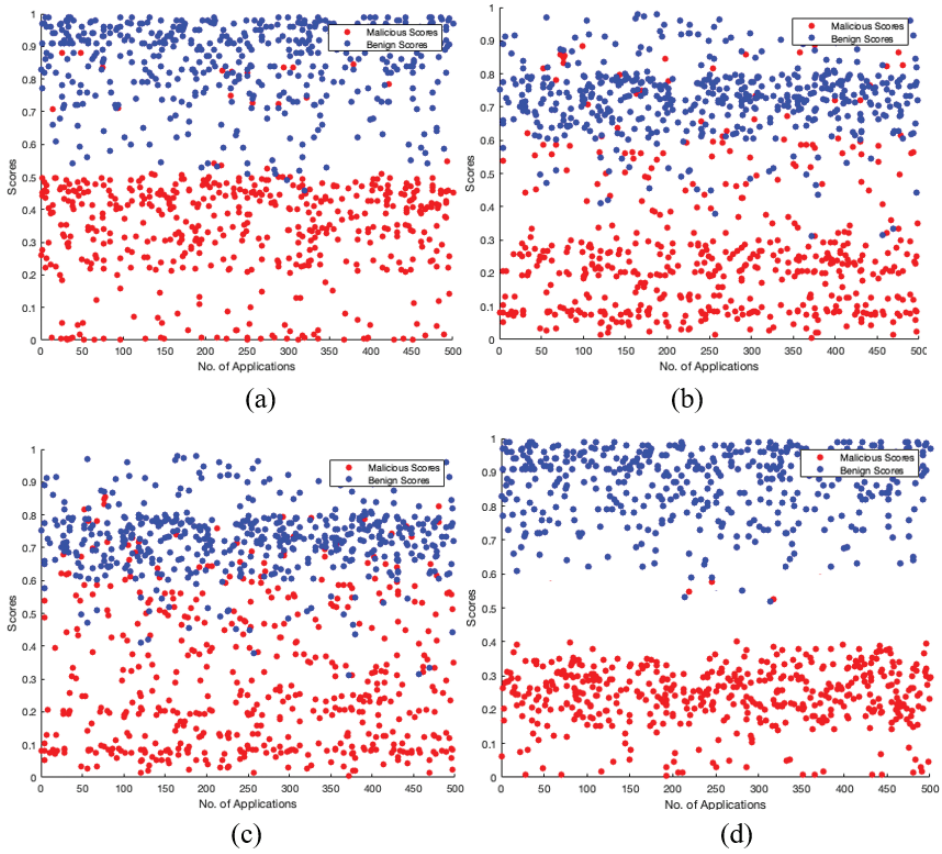


**Figure 3. Scatter plots for DB1 dataset: (a) MLIF[9] (b) HEMD[8] (c) DS[35] and (d) Proposed method.**

Fig. 3. Figures 3(a, b, c, d) displays scatter distribution plot drawn for state-of-the-art methods[8,9,35] and the proposed method respectively. From the Fig. 3, it is clear that most of score are dispersed in the area from 0.4 to 0.6, which is marked as conflicting area. Concentration of apps scores in this range of conflict is maximum for other state-of-the-art methods. However , using unified feature *UF* produced by cross iterative diffusion process and proposed optimal classifier, apps scores are broaden as depicted in Fig. 3(d). Hence, proposed classifier is efficient as it has broadened the classifier(s) score values corresponding to malicious and benign apps.

Score for database DB1 are plotted vs frequency of scores value. Overlapping of score values of benign and malicious apps to a large extent render the decision model ineffective. Overlapping of distribution scores occurs for methods[8,9,35] and proposed method as shown in Figs 4(a), 4(b), 4(c) and 4(d) respectively. Minimum overlapping of scores occurs for the proposed method as depicted in the Fig. 4(d).

Furthermore, score distribution for the state-of-the-art method and the proposed method are depicted Figure 4. As shown, score distribution in the proposed multi-stage fusion model in Fig. 4(d), has minimum overlap. It undoubtedly shows that the distributed scores of the proposed framework can perform better classification. Qualitative analysis further strengthened the Quantitative analysis of proposed framework follows in the next section.

### 4.3 Quantitative Analysis

For the suggested method, quantitative investigation is achieved via ten-fold cross validation on 5 databases (DB1, DB2, DB3, DB4, DB5) of dataset as listed in Table 2. For this, evaluation metrics i.e. Sensitivity, Accuracy, F1 Score, equal error rate and decidability index are calculated and outcomes are compared with the state-of-the-arts methods[8,9,35,39].

Decidability determine distance between benign and malicious score distribution and determined by Eqn (29).

$$D = \frac{\left|\mu_B - \mu_M\right|}{\sqrt{\frac{\sigma_B{}^2 + \sigma^2_M}{2}}} \tag{29}$$

where, $\mu_B$ and $\mu_M$ are mean and $\sigma_B$ and $\sigma_M$ are variances corresponding of benign and malicious score distributions respectively. Sensitivity is percentage of positives which are correctly recognised by binary classifier. F1 Score is weighted mean of sensitivity and precision. Accuracy measures number of correct prediction to the number of predictions or input samples. Sensitivity, F1 Score and Accuracy are determined using equations 30, 31, and 32 respectively.

$$Sensitivity = \frac{TP}{TP + FN} \tag{30}$$

$$F1Score = \frac{2TP}{2TP + FP + FN} \tag{31}$$

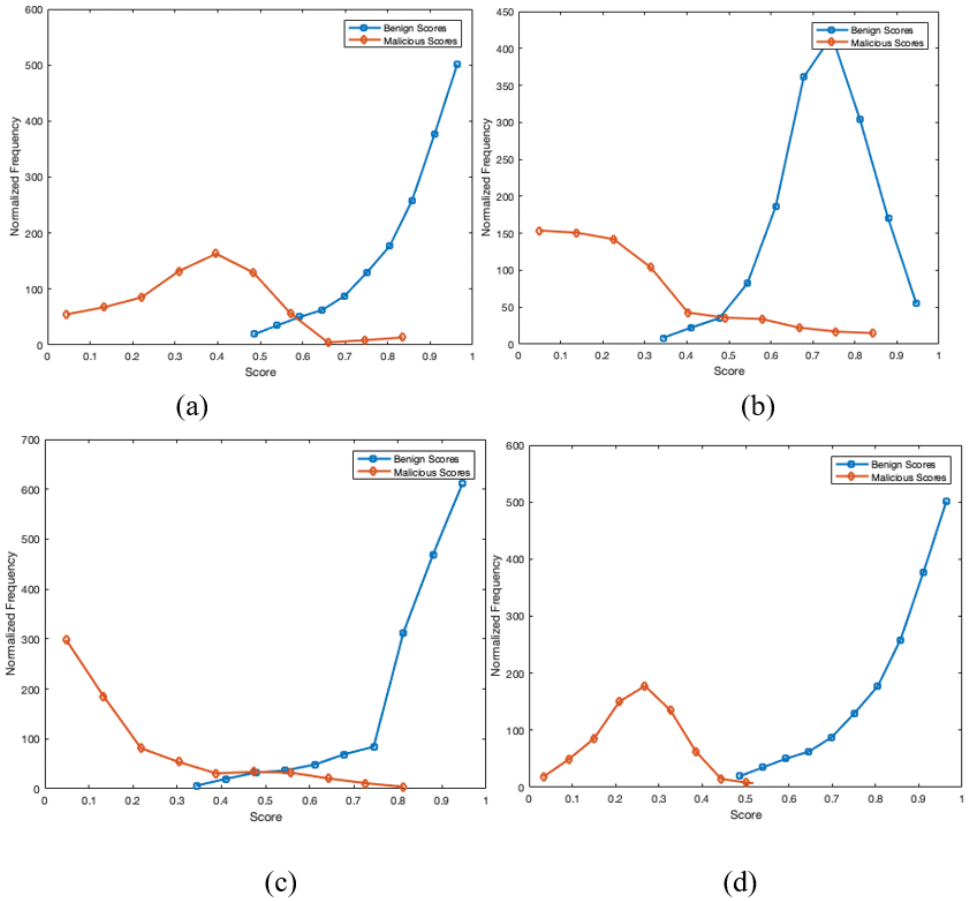


**Figure 4. Score-distribution plots for DB1: (a) MLIF[9] (b) HEMD[8] (c) DS[35] and (d) Proposed method.**

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (32)$$

where, TP, TN, FP, FN are True Positives, True Negatives, False Positives, and False Negatives respectively. Greater value of the decidability index indicates better classification. Decidability index corresponding to database DB1, for various methods are calculated and tabulated in Table 3. Average decidability indexes for are calculated as 2.9544[8], 3.3551[9], 2.7934[35] and 4.10152[39]. Proposed framework attained avg. decidability value of 5.4328 and same is validated by least overlapping of plots in Figure. 4(h). This comparatively higher value of decidability of the proposed framework in Table 3 is attained largely due to nonlinear feature fusion through cross iterative diffusion and optimal combination of classifiers score.

Receiver Operating Characteristic (ROC) curves have been determined for proposed method, and four state-of-the-art methods[8,9,35,39]. The results are depicted in Fig. 5. ROC determined the performance of a classifier as its decision threshold is varied. It is evident from the Fig. 5, for low False Acceptance Rate, proposed method achieves very high False Rejection Rate or in other terms very high true acceptance rate. There is also radical drop in false acceptance rate for state-of-the-art methods. Among ROC curves of the state-of-the-arts methods, method[9] outperformed methods[8,39,35]. It is apparent from the ROC curves that proposed framework is extremely precise and proficient.

Proposed method has also been compared with other state-of-the-art methods by calculating the Equal Error Rate (EER) using the ROC curves. Proposed framework achieved very low average EER of 1.0408, whereas other methods attained comparatively higher EER of 7.8562[8], 3.7800[9], 8.3026[35] and 5.9544[39]. Performance enhancement is attributed mainly to the non-linear graph fusion of eight feature vectors and optimal fusion of classifier scores by DSmT-based proportional conflict redistribution (PCR-6) rules where concurrent scores are enhanced and discordant scores are suppressed.

In addition, we determine the sensitivity, accuracy and F1 Score for other state-of-art method and proposed method and results are tabulated in Table 5.

On evaluation over datasets as listed in Table 2, avg. detection accuracy of 91.8%[8], 96.3%[9], 95.38%[35] and 93.89%[39] has been accomplished. Likewise, average detection accuracy

**Table 3. Decidability Index for different Methods**

| Dataset | HEMD[8] | MLIF[9] | DS[35] | FGF[39] | Proposed Method |
|---|---|---|---|---|---|
| **DB1** | 2.9699 | 3.5197 | 2.5298 | 4.1384 | **5.63** |
| **DB2** | 2.9167 | 3.3807 | 2.8906 | 4.1264 | **4.461** |
| **DB3** | 2.9373 | 3.3455 | 2.9636 | 4.0071 | **5.820** |
| **DB4** | 2.9671 | 3.0686 | 2.7068 | 4.0814 | **5.265** |
| **DB5** | 2.9812 | 3.4612 | 2.8765 | 4.1543 | **5.985** |

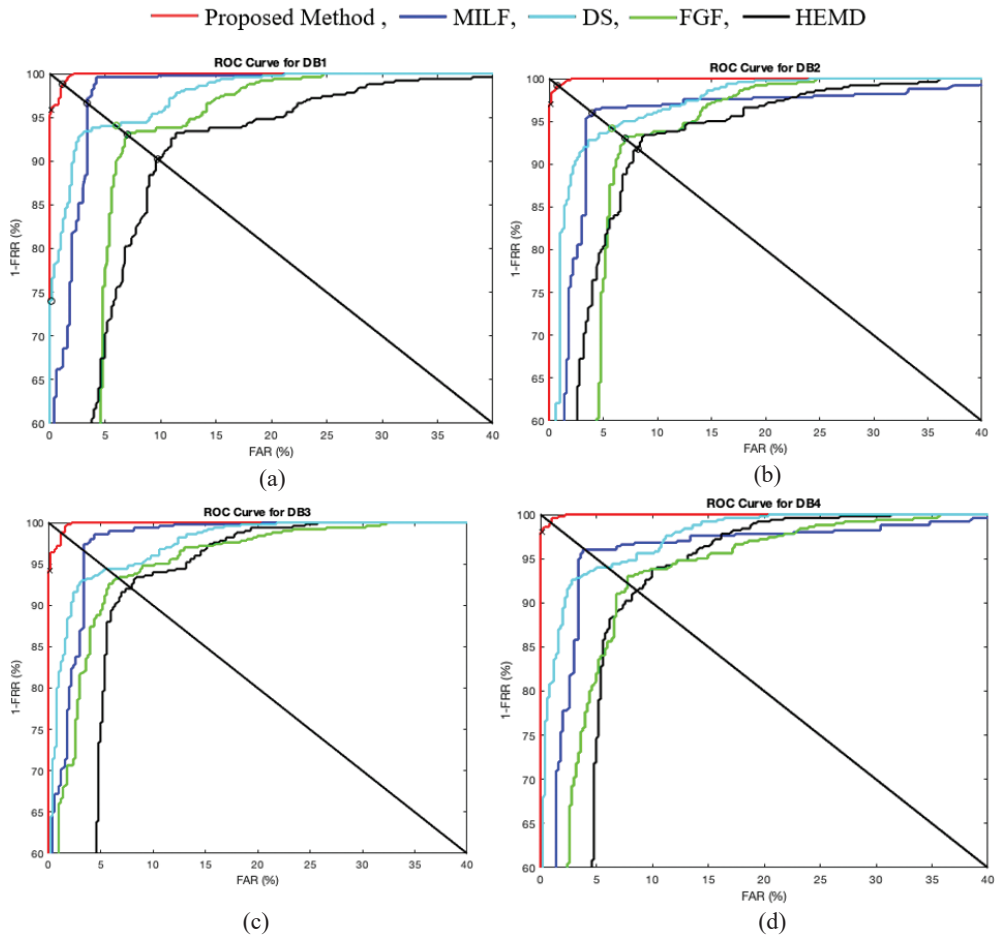


**Figure 5. Comparison of ROC curves for state-of-the-art method and proposed method (a) DB1, (b) DB2, (c) DB3, (d) DB4.**

**Table 4. Evaluation of average equal error rate for different methods**

| Dataset | HEMD[8] | MLIF[9] | DS[35] | FGF[39] | Proposed Method |
|---|---|---|---|---|---|
| **DB1** | 6.9860 | 3.4000 | 9.7804 | 5.9940 | **1.2012** |
| **DB2** | 8.5828 | 3.5000 | 7.0858 | 5.7942 | **0.8008** |
| **DB3** | 8.1836 | 3.4000 | 6.7864 | 5.9940 | **1.1010** |
| **DB4** | 7.1856 | 4.4000 | 8.6260 | 5.9940 | **1.0010** |
| **DB5** | 8.3434 | 4.2000 | 9.2344 | 5.9962 | **1.1000** |

of 98.97% was attained for proposed method. In order to gauge the real time application of proposed method, we have determined time and space complexity of proposed method. On an average, proposed method needs 5.5 sec to evaluate test app. Also, proposed method extracts dimensionality reduced feature as unified feature. Realisation of proposed method is achieved in few KBytes of memory.

Overall performance of the proposed framework versus details of the state-of-the-art methods chosen for comparison, discussed in the next section.

### 4.4 Overall Performance

Proposed smartphone security analysis framework outperforms the other comparable state-of-the-art methods viz. HEMD[8], MLIF[9], DS[35] and FGF[39] both in terms of qualitative and quantitative analysis when evaluated over datasets comprising of benign and malicious apps as tabulated in Table 2. Improvement for average accuracy of the proposed method by 7.14%[8], 2.63%[9], 3.59%[35] and 5.08%[39] has been achieved. An average accuracy of 98.97% was attained for the proposed method. Our framework handles the limitations posed by the state-of-the-art methods by conflict resolution amongst classifiers and redistribution of conflicts to produce improved set of fused scores with better scattering as can be seen in the scattering plots in Fig. 3. Score distribution plots in Fig. 4 clearly depicts that the overlapping of malicious and benign scores is reduced to a great extent. Significant improvement in the average decidability index values of proposed model to 5.4328 as compared to 2.954[8], 3.355[9], 2.793[35] and 4.101[39] further reinforces our claim for the better performance of the proposed method. This improved value of decidability index of the proposed framework is attained mainly due to nonlinear

**Table 5. Comparison of performance metrics namely sensitivity, accuracy and F1 Score for different comparable methods**

| Dataset | | HEMD[8] | MLIF[9] | DS[35] | FGF[39] | Proposed method |
|---|---|---|---|---|---|---|
| **DB1** | Sensitivity | 0.9281 | 0.9660 | 0.9002 | 0.9381 | 0.9880 |
| | Accuracy | 0.9271 | 0.9650 | 0.9481 | 0.9380 | 0.9880 |
| | F1 Score | 0.9627 | 0.9827 | 0.9474 | 0.9680 | 0.9939 |
| **DB2** | Sensitivity | 0.9201 | 0.9600 | 0.9102 | 0.9401 | 0.9920 |
| | Accuracy | 0..9191 | 0.9590 | 0.9531 | 0.9400 | 0.9919 |
| | F1 Score | 0.9584 | 0.9796 | 0.9529 | 0.9691 | 0.9959 |
| **DB3** | Sensitivity | 0.9122 | 0.9660 | 0.9301 | 0.9381 | 0.9880 |
| | Accuracy | 0.9112 | 0.9650 | 0.9630 | 0.9380 | 0.9889 |
| | F1 Score | 0.9541 | 0.9827 | 0.9638 | 0.9681 | 0.9939 |
| **DB4** | Sensitivity | 0.9241 | 0.9640 | 0.9102 | 0.9401 | 0.9900 |
| | Accuracy | 0.9231 | 0.9630 | 0.9531 | 0.9400 | 0.9899 |
| | F1 Score | 0.9606 | 0.9817 | 0.9529 | 0.9691 | 0.9949 |
| **DB5** | Sensitivity | 0.9064 | 0.9794 | 0.9583 | 0.9544 | 0.9945 |
| | Accuracy | 0.9110 | 0.9650 | 0.9520 | 0.9385 | 0.9898 |
| | F1 Score | 0.8220 | 0.9645 | 0.9521 | 0.9360 | 0.9898 |

feature fusion through cross iterative diffusion and optimal combination of classifiers score values. Feature fusion process used in the proposed framework exploits complementary info from the eight individual features.

Zhu[8], *et al.* uses a set of 4 features viz. permission rate, permission, sensitive API, system events to detects malwares in the apps with a single RF classifier. Here, low accuracy is attributed to lack of feature and score fusion in the model. In Mlifdect[9], authors employ parallel machine learning and information fusion approach. Normal vector based feature transformation was employed along with DS theory and probability for malapps detection. Here any conflict arising between the classifier score is not resolved. Authors[35] proposed a multi classifier (SVM, J48, Bayes Net) and fusion method to identify malapps. Jiang[39], *et al.* proposed a static feature (native code, intent filter, reflection, root, permissions) based malapp framework using four ML classifiers (KNN, NB, SVM, J48). This method also achieved low detection accuracy due to lack of feature fusion and optimal classifier fusion.

In a nutshell, the proposed multistage fusion framework for smartphone security analysis outclass other state of the art techniques. It is suitable for classifying test app as malicious or benign with high detection accuracy by feature fusion through cross iterative graph diffusion method and optimal fusion of classifier scores. Quantitative performance enhancement is attributed to extraction of multiple features and their fusion through cross diffusion. Also, our smartphone security analysis framework outperforms numerous limitations of state-of-the-art methods mainly due to extraction of complementary information and optimal fusion of classifiers to create clear and distinct boundary between the benign and malicious classes.

## 5. LIMITATIONS

In this manuscript, smartphone security analysis framework is proposed wherein optimal classification of multi-cue is achieved. Performance is evaluated both qualitatively and quantitatively over the benchmarked dataset and we achieved significant improvement in performance. However, proposed method could not detect self-altering malware and zero-day attack. Performance is hindered by bytecode encryption. Apart from this, examining the malicious application is restricted by anti-reverse engineering methods such as obfuscation as it forestalls the access of system API calls.

## 6. CONCLUSION AND FUTURE DIRECTION

Challenges faced in the identification of malapps in the smartphones, motivated us to develop an efficient multi-fusion based android malapp detection method based on static analysis. In our methodology, eight static features are exploited for development of solution. Our multi-fusion technique is a two stage fusion approach. In first stage, deduced static features are fused into a single unified robust vector through extraction of the complementary information from eight features using non-linear graph unification. In second stage, optimal classifiers are used for classification of an app. Proposed framework has been designed to make classification robust to noise and hence help in drastically improving the score distribution. Classification is performed by training RF, SVM, and NB classifiers followed by classifier scores fusion using PCR-6 rule that resolved conflicts amongst classifiers besides redistributing the conflicts efficiently. Qualitative investigations of outcomes disclosed that proposed optimal classifier broadened the score-distribution of malign and benign apps. Moreover, our method has attained average accuracy of 98.97%, average equal error rate of 1.04%, average F1 score value of 0.9936 and average sensitivity value of 0.9905 when evaluated over datasets as listed in Table 2. Quantitative analysis of suggested method vs. state-of-the-art techniques reveal that proposed method outperforms all of them.

In future, we will extend the proposed method for different types of malware. Main reason for misclassification is non-inclusion of dynamic features and structure related static features in our model. Therefore, we will extend our framework to hybrid analysis to improve the detection efficiency by including the dynamic analysis to static analysis framework. Also, training on more datasets covering the different families of malware from diverse sources will solve under-fitting and overfitting problems in detection.

## REFERENCES


1. IDC, https://www.idc.com/promo/smartphone-market-share/os (visited on 20 Sep,2020)
2. Li, J.; Sun, L.; Yan, Q.; Li, Z.; Srisaan, W & Ye, H. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE T Ind Inform.*, 2018, **14**(7), 3216-3225. doi: 10.1109/TII.2017.2789219
3. Zhu, Hui-Juan & You, Zhu-Hong & Zhu, Zexuan & Shi, Wei-Lei & Cheng, Li. DroidDet: effective and robust detection of Android malware using static analysis along with rotation forest model. *Neurocomputing*, 2018, **272**, 638-646. doi:10.1016/j.neucom.2017.07.030
4. Arora, Anshul; Peddoju, Sateesh Kumar & Conti, Mauro. PermPair: Android Malware Detection Using Permission Pairs. *IEEE T Inf Foren*, 2019, **15**, 1968-1982. doi: 10.1109/TIFS.2019.2950134.v
5. Ma, Z.; Ge, H.; Liu,Y.; Zhao, M & Ma,J. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms, in *IEEE Access*, 2019, **7**, 21235-21245. doi: 10.1109/ACCESS.2019.2896003
6. Tao, G.; Zheng, Z.; Guo, Z and Lyu, M. R .MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs in *IEEE T Reliab*, 2018,**67**(1),355-369. doi: 10.1109/TR.2017.2778147
7. Yerima, S. Y. ; Sezer,S and Muttik, I. High accuracy android malware detection using ensemble learning, in *IET Inform secur*, 2015,**9**(6),313-320. doi: 10.1049/iet-ifs.2014.0099
8. Zhu, Hui-Juan & Jiang, Tong-Hai & Ma, Bo & You, Zhu-Hong &Shi, Wei-Lei & Cheng, Li. HEMD: a highly efficient random forest-based malware detection framework for Android. *Neural Comput Appl*, 2018, **30**, 3353-3361. doi:10.1007/s00521-017-2914-y
9. Wang, Guanxin & Zhang, Dafang & Su, Xin & Li, Wenjia. Mlifdect: Android Malware Detection Based on Parallel Machine Learning and Information Fusion. *Secur Commun Netw*. 2017, **2017**(1), 1-14. doi: 10.1155/2017/6451260
10. Kim,T.; Kang,B.; Rho,M.; Sezer, S and Im, E.G. A Multimodal Deep Learning Method for Android Malware Detection Using Various Features. *IEEE T Inf Foren Sec*, 2019, **14**(3), 773-788. doi: 10.1109/TIFS.2018.2866319
11. Fan, Ming.; Liu, Jun.; Luo, Xiapu.; Chen, Kai.; Tian, Zhenzhou.; Zheng, Qinghua & Liu, Ting. Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis, *IEEE T Inf Foren Sec*,2018,**13**(8),1890-1905, doi: 10.1109/TIFS.2018.2806891
12. Wang, Wei.; Li Yuanyuan.; Wang,Xing.; Liu, Jiqiang & Zhang, Xiangliang. Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers, *Future Gener. Comput. Syst*. 2018, **78**(3),987-994. doi: 10.1016/j.future.2017.01.019
13. Wang, Wei.;Gao, Zhenzhen; Zhao, Meichen.; Li, Jiqiang; Liu, Jiqiang & Zhang, Xiangliang. DroidEnsemble: Detecting Android Malicious Applications with Ensemble String and Structural Static Features. *IEEE Access*, 2018, **6**, 31798-31807. doi:10.1109/ACCESS.2018.2835654
14. Song, Jun.; Hana ,Chunling.; Wang, Kaixin.; Zhaoa, Jian.; Ranjanb ,Rajiv & Wang, Lizhe. An integrated static detection and analysis framework for android. *Pervasive Mob. Comput*. 2016, **32**, 15-25. doi: 10.1016/j.pmcj.2016.03.003
15. Arp , Daniel.; Spreitzenbarth, Michael.; Hubner, Malte.; Gascon, Hugo & Rieck, Konrad. DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket. In: NDSS, 2014. doi: 10.14722/ndss.2014.23247
16. Smarandache, Florentin & Dezert, J : Advances and Applications of DSmT for Information Fusion, American Research Press, 2015. 506p.
17. Denoeux, T. & Masson, M. H. (2010) Dempster-Shafer Reasoning in Large Partially Ordered Sets: Applications in Machine Learning. In: Huynh, VN.; Nakamori, Y & Lawry J. Inuiguchi M. (eds) Integrated Uncertainty Management and Applications. *Advances in Intelligent and Soft Computing*, **68**. Springer, Berlin, Heidelberg. doi:10.1007/978-3-642-11960-6_5
18. Wang, Wei.; Zhao, Meichen.; Gao, Zhenzhen.; Xu, Guangquan.; Xian, Hequn.; Li, Yuanyuan & Zhang, Xiangliang. Constructing Features for Detecting Android Malicious Applications: Issues, Taxonomy and Directions, in *IEEE Access*, 2019, 7, 67602-67631. doi: 10.1109/ACCESS.2019.2918139
19. Faruki, Parvez.; Bharmal, Ammar.; Laxmi, Vijay.; Ganmoor, Vijay.; Gaur, Manoj Singh.; Conti, Mauro & Rajarajan , Muttukrishnan. Android Security: A Survey of Issues, Malware Penetration, and Defenses, *IEEE Commun Surv Tut*, 2015, **7**(2),998-1022. doi: 10.1109/COMST.2014.2386139
20. Zhang, H.; Luo, S.; Zhang, Y & Pan, L. An Efficient Android Malware Detection System Based on Method-Level Behavioural Semantic Analysis. *IEEE Access*, 2019, **7**, 69246- 69256. doi: 10.1109/ACCESS.2019.2919796
21. Qiu, Junyang.; Zhang, Jun.; Luo, Wei.; Pan, Lei.; Nepal, Surya.; Wang,Yu & Xiang, Yang. A3CM: Automatic Capability Annotation for Android Malware, in *IEEE Access*, 2019, **7**, 147156-147168

doi: 10.1109/ACCESS.2019.2946392

22. Alotaibi, A. Identifying Malicious Software Using Deep Residual Long-Short Term Memory, *IEEE Access*, 2019, **7**, 163128-163137. doi: 10.1109/ACCESS.2019.2951751
23. Olukoya, Oluwafemi & Mackenzie, Lewis & Omoronyia, Inah. Security-Oriented View of App Behaviour using Textual Descriptions and User-Granted Permission Requests. *Comput Secur*, 2019, **89**, 101685. doi: 10.1016/j.cose.2019.101685
24. Alam, Shahid & Alharbi, Soltan & Yildirim, Serdar. Nested Flow of Dominant APIs for Detecting Android Malware. *Comput Netw*, 2020, **167**, 107026. doi: 10.1016/j.comnet.2019.107026
25. Mateless, Roni & Rejabek, Daniel & Margalit, Oded & Moskovitch, Robert. Decompiled APK based malicious code classification. *Future Gener Comput Syst*. 2020, **110**, 135-147 doi:10.1016/j.future.2020.03.052
26. Tian, Ke; Danfeng, Yao; Barbara, G. Ryder; Gang, Tan & Guojun, Peng. Detection of Repackaged Android Malware with Code-Heterogeneity Features. *IEEE T Depend Secure*, 2020,**17**, 64-77. doi:10.1109/TDSC.2017.2745575
27. Onwuzurike, Lucky & Mariconti, Enrico & Andriotis, Panagiotis & De Cristofaro, Emiliano & Ross, Gordon & Stringhini, Gianluca, MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioural Models, *ACM Trans. Priv. Secur*, 2017, **22**(2), 1-34. doi: 10.1145/3313391.
28. Q. Han, V. S. Subrahmanian and Y. Xiong. Android Malware Detection via (Somewhat) Robust Irreversible Feature Transformations, *IEEE T Inf Foren* Sec,2020,**15**,3511-3525. doi: 10.1109/TIFS.2020.2975932.
29. B. Wang, J. Jiang, W. Wang, Z. Zhou & Z. Tu. Unsupervised metric fusion by cross diffusion,2012. *In* IEEE Conference on Computer Vision and Pattern Recognition, Providence, RI, 2012, 2997-3004. doi: 10.1109/CVPR.2012.6248029.
30. Tong, Tong & Gray, Katherine & Gao, Qinquan & Chen, Liang & Rueckert, Daniel. Multi-Modal Classification of Alzheimer's Disease Using Nonlinear Graph Fusion, *Pattern Recogn*. 2017,**63**,171-181. doi: 10.1016/j.patcog.2016.10.009.
31. Vapnik, Vladimir N. *In* The Nature of Static Learning Theory, Springer, 2000. pp.123-216. doi:10.1007/978-1-4757-3264-1
32. Breiman, L. Random forests. *Mach. Learn*, 2001, **45** (1). 5–32. doi:10.1023/A:1010933404324
33. Webb, G. I., Keogh, E., Miikkulainen, R., Miikkulainen, R. & Sebag, M. *In* Encyclopedia of Machine Learning, Springer, 2011.pp. 713–714. doi:10.1007/978-0-387-30164-8_576
34. Xiao, Fuyuan. Multi-sensor data fusion based on the belief divergence measure of evidences and the belief entropy. *Information Fusion*, 2019, **46**, 23-32. doi: 10.1016/j.inffus.2018.04.003
35. Du, Yao & Wang, Xiaoqing & Wang, Junfeng. A static Android malicious code detection method based on multi-source fusion. *Secur Commun Netw*, 2015, **8**(17), 3238-3246. doi:10.1002/sec.1248
36. Xu, K.; Li, Y & Deng, R. H. ICCDetector: ICC-based malware detection on Android, *IEEE T Inf Foren Sec*. 2016, **11**(6), 1252–1264. doi:10.1109/TIFS.2016.2523912
37. Hardware Features, https://developer.android.com/guide/topics/manifest/uses-feature-element#hw-features (visited on Sep 20,2020)
38. Walia, G. S.; Ahuja, H. ; Kumar, A. ; Bansal, N & Sharma, K. Unified Graph-Based Multicue Feature Fusion for Robust Visual Tracking, in *IEEE T Cybernetics*, 2020, **50**(6),2357-2368. doi: 10.1109/TCYB.2019.2920289
39. Jiang, Xu.; Mao, Baolei**.;** Guan, Jun & Xingli, Huang. Android Malware Detection Using Fine-Grained Features, *Scientific Programming*, 2020, **2020**,1-13. doi: 10.1155/2020/5190138
40. Wei, F.; Li, Y.; Roy, S.; Ou, X & Zhou, W. Deep Ground Truth Analysis of Current Android Malware. *In* Polychronakis, M & Meier, M. (eds) Detection of Intrusions and Malware, and Vulnerability Assessment. 2017. Lecture Notes in Computer Science, 2017, **10327**. Springer, Cham. doi:10.1007/978-3-319-60876-1_12
41. Mahdavifar, S. ; Kadir, A. F. Abdul .; Fatemi, R.; Alhadidi, D and Ghorbani, A. A. Dynamic Android Malware Category Classification using Semi-Supervised Deep Learning, *In* IEEE Intl Conf on Dependable, Autonomic and Secure Computing(DASC),2020,515-522. doi:10.1109/DASC-PICom-CBDCom-CyberSciTech49142.2020.00094
42. Rastogi, Sajal & Bhushan, Kriti & Gupta, B B. Android Applications Repackaging Detection Techniques for Smartphone Devices. *Procedia Comput Sci*, 2016,**78**, 26-32. doi:10.1016/j.procs.2016.02.006
43. Kumar, S.; Indu, S.; Walia, G.S. Smartphone Traffic Analysis: A Contemporary Survey of the State-of-the-Art. *In* Proceedings of the Sixth International Conference on Mathematics and Computing. Advances in Intelligent Systems and Computing, 2020, **1262**, 325-343, Springer, Singapore. doi:10.1007/978-981-15-8061-1_26
44. Kumar S.; Indu S.; Walia, G.S. Recent Advances in Android Malware Detection-A Survey. *In* 1st International Conference on Communication, Computing and Signal Processing, 2020.
45. Xiao, Fuyuan. A new divergence measure for belief functions in D–S evidence theory for multisensor data fusion. *Information Sciences*, 2020, **514**, 462-483. doi: 10.1016/j.ins.2019.11.022

## CONTRIBUTORS

**Mr Sumit Kumar** received his BE(ECE) and ME(ECE) from Delhi College of Engineering, University of Delhi, Delhi. He is presently working as a senior scientist at DRDO laboratory in Delhi. His current area of research interest includes smartphone security, machine learning and artificial intelligence. He is also a member of CRSI and IEEE.
In the current study, he conceived the framework, carried out the literature survey and completed the investigational work by evaluating the proposed framework both qualitatively and quantitatively.

**Prof. S. Indu** did her PhD in the area of Visual Sensor Networks from University of Delhi, Delhi, India. She Joined Electronics and Communication Engineering Department of Delhi College of Engineering in 1999. Currently she is working as Dean (Student Welfare) and Professor of ECE Department of Delhi Technological University. She has published around 200 papers in reputed Journals and National and International conferences. Her area of research interest is Computer Vision, Sensor Networks and Image Processing. She received Commendable research award, 2018 and 2019, of Delhi Technological University. She has completed three DST sponsored project and currently she is PI of two DST sponsored projects.
In the current study, she reviewed the incremental work and provided many valuable inputs.

**Dr Gurjit Singh Walia** received his PhD in the field of computer vision from Delhi Technological University (Formerly Delhi College of Engineering), New Delhi, and the ME in electronics from Punjab Engineering College, Chandigarh. He is working as a Senior Scientist with Defence Research and Development Organisation, New Delhi. His current research interests include machine learning, pattern recognition, and information security. He has published over 40 research papers in international journals and conferences. He is reviewer of various IEEE transactions, Elsevier and Springer journals.
In the current study, he has corrected the final manuscript and provided expert guidance for conducting study.