Database Design Approaches for Secure Storage on Mobile Devices

A. Vivek Chandra, K. Harish Kumar, M. K. Chaithanya, Poonguzhali P., and Mahesh U. Patil

***Abstract*—The increase in the usage of mobile devices has increased the number of mobile applications and sensitive data stored by these applications. This has put forward many new challenges for securing the data using cryptographic algorithms and performing search for relevant data. As Android is one of the most widely used mobile operating system and Java is its development language, the paper presents two design approaches for secure storage and retrieval of Java objects on Android Platform. Along with the security of the data, the overhead involved in securing the data is also considered while designing these approaches. At the end of the paper, a comparison study of the performance of the two proposed approaches is presented.**

***Index Terms*—Encryption, decryption, serialization, de-serialization, cryptography, android, database, querying.**

I. Introduction

In the modern world, the reachability of a mobile device is increased because of reduced hardware manufacturing costs, improved computing capabilities and wide availability of the open source mobile operating systems. This reachability and the computing power has developed many new origins for application development on mobile platforms. The applications installed on a mobile device holds a variety of sensitive (private) data of the user like passwords, credit/debit card details, personal details, etc., The sensitive data stored by these applications need to be protected with necessary security mechanisms. One such security mechanisms is encrypting the data using the cryptographic algorithms. The encryption provides security to the data, but there lies a challenge in providing security without affecting the user experience. So, the paper presents two design approaches for effectively querying over an encrypted data. The related approaches for querying on the encrypted data are discussed in Section II. Our approaches are discussed in Section III. Section IV gives a detailed analysis along with results of the proposed approaches.

II. Related Work

The encryption of data provides security to the data but searching on the encrypted data proves time consuming. The traditional way to search on an encrypted data is to decrypt entire data and search the decrypted data for matching records. This approach is time consuming, as it is needed to decrypt all the records to make a search.

Erez Shmueli *et al*. in [1] discusses about major challenges and design considerations pertaining to database encryption. It presents an attack model and main relevant challenges of data security, encryption overhead, key management and integration footprint. The presents related academic work on alternative encryption configurations pertaining to encryption locus, indexing encrypted data and key management.

Dawn Xiaodong Song *et al.* in [2] proposes four schemes that allow searching the encrypted data without decryption. However those schemes are suitable for client server environment.

In [3] H. Hacigumus *et al.* proposes a query method over encrypted data, in which the query processing is performed at service provider's site while decryption and remainder of the query processing is performed at the client site.

Hyun-A Park *et al.* in [4] proposes a group search scheme over encrypted data. The shared sensitive information requires more security and privacy protection, in that paper two schemes are proposed which can search the encrypted document without re-encrypting all documents in a server even if group keys have to be updated. The schemes can support general database normalization for encrypted database.

Eu-Jin Goh in [5] discusses about hash table based secure indexes, which uses hash tables for storing key words and pointers to encrypted documents. This approach is mainly aimed at querying encrypted documents for a particular keyword without actually decrypting the complete document.

Z. Wang *et al.* in [6] proposes a framework which implements query over encrypted data based on B+ tree. Before encrypting the data, a B+ tree index is generated for the data. When querying the encrypted data, it locates the encrypted records related to the input query key based on the B+ tree index and then it decrypts the encrypted records to accomplish the results. In this approach the B+ tree must be encrypted to avoid leaking of confidential information. The results of experiments for approach discussed in [6] show that the query performance over the encrypted data decreases about 20 percent compared with the plain text query performance.

In [7] Hong Zhu *et al.* proposes a query scheme, which conducts fast LIKE and range query over the encrypted character strings in databases. The strings are represented as characteristics matrix & the size of this matrix is large and requires much computation.

References [2]-[4] discuss about searching on encrypted data for a client server architecture which is not suitable for mobile device environment. Reference [5] discusses a method to search on an encrypted document using secure indexes.

Manuscript received December 18, 2014; revised February 3, 2015. This work is supported by Department of Electronics and Information Technology (DeitY), Ministry of Communications & IT, Government of India.

The authors are with C-DAC, Hyderabad, Telangana 500005, India. (e-mail: vcamancha@cdac.in, kharish@cdac.in, mkchaithanya@cdac.in, poonguzhalip@cdac.in, maheshp@cdac.in).

DOI: 10.7763/IJFCC.2015.V4.371

References [6], [7] require more computation power and the mobile devices have limited computation capabilities when compared to a desktop environment.

In [8] Mohammed Alhanjouri *et al*. discusses an approach to query over the encrypted data using the HashMap. In this approach the mapping between the plain text and the encrypted text is maintained using the KEY, VALUE pairs of the HashMap. In this, the KEY is a plain text and the VALUE is an encrypted text of the plain text. The key being an un-encrypted content, will reveal some information about the data. Also, in this approach, we need to have index on a particular column for the query to be successfully performed. By considering the shortcomings in the previous approach, a new approach of Hash Table, which solves the previous shortcomings and is suitable for the mobile environment is proposed.

Also, a Field based approach is proposed and a comparison study with respect to performance of both the approaches is presented in this paper.

III. Implementation

The proposed database design introduce two approaches for encrypted storage and querying on Java objects (from here on referred to as records)

1) Hash Table Based Approach.

2) Field Based Approach.

A. Hash Table Based Approach

Save Operation: Figure 1 depicts the architecture of save operation for Hash Table based approach. As shown in the Figure 1, first the Java object which need to be saved is concurrently processed by two engines. The first one is the hash engine and the second one is the crypto engine. In the hash engine, there are two sub components, i.e., field extractor and hash manager. The field extractor extracts all the individual fields of the object. Then the hash manager calculates the hash of these extracted fields and stores those hashes in a hash table. This hash table is provided as input to the database manager. Calculation of hash section below explains in detail the steps involved in calculating the hash and storing those hashes in the hash table with an example. Concurrently, the object is given to the crypto engine also. The crypto engine consists of two sub components. One is the object serializer and the other is data encryptor. The Java object provided to the crypto engine is first serialized by the object serializer and then the serialized object is encrypted by the data encryptor. The encryption key provided to the data encryptor is generated using a PBES (Password Based Encryption Standard) algorithm where the password is supposed to be provided by the user. The encrypted object is stored in the database and the corresponding row id is mapped to the hash table. The mapping of hash table to the row id is explained in detail in the Mapping of hash table to the row id section below.

Calculation of hash: After each individual field of the Java object is extracted by the Field Extractor, they are hashed by the Hash Manager. For performing this hash operation, each field value is first concatenated with the corresponding field name. The hash operation is now performed on this concatenated string. This concatenation of field name with field value is aimed at removing the unwanted search results as explained in the example below. These hashes are then stored in the hash table. The Algorithm 1 shows the implementation of save operation in Hash Table approach.

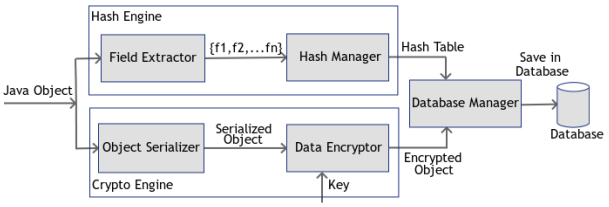

Fig. 1. Save operation in Hash Table approach.

Algorithm 1 Save operation in Hash Table approach

```
1: procedure SAVE(Java_object)                          ▷ un-encrypted record
2:     objectBytes ← SERIALIZE(Java_object)
3:     encrypted_record ← ENCRYPTRECORD(objectBytes)
4:     SAVERECORD(encrypted_record)
5:     rowId ← GETLASTINSERTEDROWID
6:     field_name_array[ ] ← EXTRACTFIELDNAMES(Java_object)
7:     field_value_array[ ] ← EXTRACTFIELDVALUES(Java_object)
8:     n ← field_name_array.SIZE()
9:     for i ← 0, i < n do
10:        hashValue ← MD5(field_name_array_i + fieldvalue_value_array_i)
11:        STOREHASH(hashValue,rowId)
12:    end for
13: end procedure
```

For example, consider a contact bean object whose fields are contact_id, contact_first_name, contact_last_name, contact_address, contact_number, contact_email and contact_organization. We consider the two records shown in Table I as sample data for all our examples discussed in the following sections.

TABLE I: SAMPLE CONTACT RECORDS

ID	First Name	Last Name	Address	Org.	Email	Phone
1	vivek	chandra	Hyderabad	CDAC	vca@cdac.in	23150115
2	chandra	sri	Hyderabad	CDAC	sri@cdac.in	23150117

The contact record 1 is saved by initially calculating the hash values of all the field entries as described in the above section. The hash table after calculating hash keys is shown in Table II. Record table after serializing and encrypting the object is shown in Table III.

TABLE II: HASH TABLE

Keys	Row id's
HASH(contact_id+l)	
HASH(contact_last_name+vivek)	
HASH(contact_first_name+chandra)	
HASH(contact_number+23150115)	
HASH(contact_address+hyderabad)	
HASH(contact_organization+CDAC)	
HASH(contact_email+vca@cdac.in)	

TABLE III: RECORD TABLE

Row Id	Encrypted Object
1	Encrypted Contact Record 1

Mapping of hash table to the row id- Now each individual

Java object has multiple entries in the hash table, each entry representing a field of the object. Also, each Java object is serialized, encrypted and then this encrypted Java object is stored as a row in the database. This row id is mapped to each individual hash entry which corresponds to that particular object. Now, the row id's in the record table are mapped to the entries in the hash table corresponding to record 1 (contact 1). The updated hash table is shown in Table IV.

TABLE IV: HASH TABLE

Keys	Row id's
HASH(contact_id+l)	1
HASH(contact_last_name+vivek)	1
HASH(contact_first_name+chandra)	1
HASH(contact_number+23150115)	1
HASH(contact_address+hyderabad)	1
HASH(contact_organization+CDAC)	1
HASH(contact_email+vca @ cdac in)	1

Table V and Table VI shows the updated database tables after adding the contact 2 shown in Table I.

TABLE V: HASH TABLE

Keys	Row id's
HASH(contact_id+l)	1
HASH(contact_first_name+vivek)	1
HASH(contact_last_name+chandra)	1
HASH(contact_number+23150115)	1
HASH(contact_address+hyderabad)	1,2
HASH(contact_organization+CDAC)	1,2
HASH(contact_email+vca @ cdac.in)	1
HASH(contact_id+2)	2
HASH(contact_first_name+chandra)	2
HASH(contact_last_name+sri)	2
HASH(contact_number+23150117)	2
HASH(contact_email+sri@cdac.in)	2

TABLE VI: HASH TABLE

Row Id	Encrypted Object
1	Encrypted Contact Record 1
2	Encrypted Contact Record 2

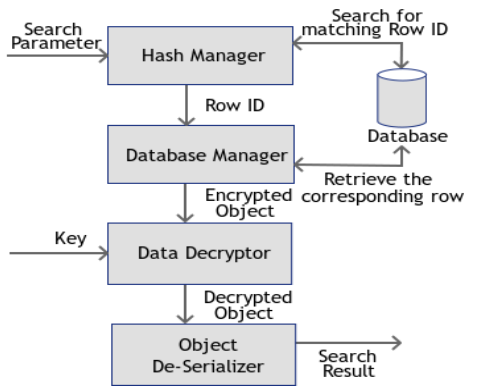

Fig. 2. Search operation in Hash Table approach.

Search Operation: The architecture of search operation for the Hash table based approach is depicted in Fig. 2. The search parameter is first provided to the hash manager which in turn calculates the hash of that parameter as explained in the Calculation of hash section above. This calculated hash is searched against the hash table for a match. If a match is found, the corresponding row id which was mapped earlier (refer to Mapping of hash table to the row id section) is retrieved. With this row id, the encrypted object is retrieved by the database manager. This encrypted object is decrypted and then de-serialized to get the required Java objects.

Algorithm 2 Search operation in Hash Table approach

```
1: procedure SEARCH(search_string)                    ▷ input search key
   "fieldname+fieldvalue"
2:     hashValue ← MD5(search_string)
3:     record_ids[ ] ← SEARCHHASHTABLE(hashValue)
4:     R ← record_ids.SIZE()
5:     for i ← 0, i < R do
6:         objectBytes ← RETRIEVEFROMOBJECTTABLE(record_ids_i)
7:         decrpytedObjectBytes ← ECRYPTBYTES(objectBytes)
8:         result_i ← DESERIALIZE(decrpytedObjectBytes)
9:     end for
10:    return result
11: end procedure
```

The Algorithm 2 shows the implementation of search operation in Hash Table approach.

For example, consider the contacts shown in Table I and we want to retrieve all the contact records whose last name is "sri". First we query the hash table for retrieving row id's corresponding to matching records.

SELECT rowid from HashTable WHERE key = HASH*(contact_last_name+sri)

This query returns a list of row id's who are having the last name as "sri". Now we perform another query requesting records with row ids retrieved in the above step.

SELECT encrypted_record from RecordTable WHERE rowid = 2

This query returns all the encrypted records that are having last name as "sri". These records are decrypted and de-serialized.

Consider another example, where we want to retrieve records whose last name is 'chandra' from the contacts shown in Table I. In this case if we saved only hash of field value then the query would return 2 records one whose last name is 'chandra' and other whose first name is chandra which is not the desired result. So, to overcome this result we are calculating the hash of fieldname+fieldvalue. So if we want to fetch for record whose last name is 'chandra' then the hash of contact_last_name+chandra is calculated, which gives a different result when compared to hash of contact_first_name+chandra and accordingly the corresponding records are fetched.

B. Field Based Approach

Save Operation: Fig. 3 depicts the architecture for save operation in Field based approach. In this approach, the individual fields of the Java object which need to be saved are extracted by the Field Extractor. Each of these fields are encrypted individually by the Field Encryptor. The encryption key provided to the Field Encryptor is generated in the same way as in Hash table based approach, i.e., using a PBES (Password Based Encryption Standard) algorithm where the password is supposed to be provided by the user. Once the encryption of each individual field is completed, they are saved in the database by the Database Manager. In the database, each Java object is represented by a row and each field in the Java object is represented by a column in the row. That means, the

number of columns in a row is equivalent to the number of fields in the Java object.

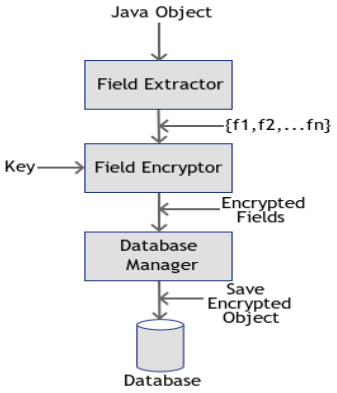

Fig. 3. Save operation in field approach.

The Algorithm 3 shows the implementation of save operation in Field approach.

Algorithm 3 Save operation in Field approach

```
1: procedure SAVE(Java_object)                    ▷ un-encrypted record
2:     field_value_array[ ] ← EXTRACTFIELDVALUE(Java_object)
3:     field_name_array[ ] ← EXTRACTFIELDNAME(Java_object)
4:     CREATETABLE(field_name_array)   ▷ Create the table with field names as columns
5:     n ← field_value_array.SIZE()
6:     for i ← 0, i < n do
7:         encrypted_fields_i ← ENCRYPTFIELD(field_value_array_i)
8:     end for
9:     SAVERECORD(encrypted_fields)   ▷ Save the field values corresponding to field names in database table
10: end procedure
```

For example, consider the contacts shown in Table I. In this approach we first extract values of all the fields of the input bean object and encrypt those values. These values are saved into the database table as shown in Table VII. The Table VII represents two bean objects saved in Field based approach.

TABLE VII: RECORD TABLE IN FIELD APPROACH

ID	First Name	Last Name	Addr.	Org.	Email	Phone
enc* (1)	enc (vivek)	enc (chandra)	enc (Hyderabad)	enc (CDAC)	enc (vca@cdac.in)	enc (23150115)
enc (2)	enc (chandra)	enc (sri)	enc (Hyderabad)	enc (CDAC)	enc (sri@cdac.in)	enc (23150117)

*enc- encrypted value

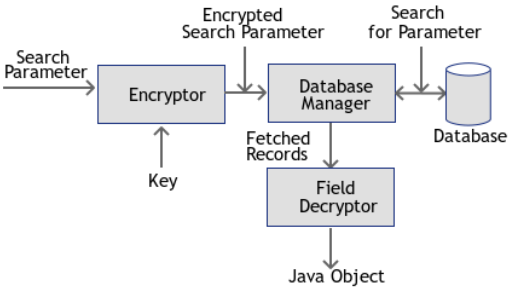

Fig. 4. Search operation in field approach.

Search Operation: The architecture of search operation in Field based approach is depicted in Fig. 4. Here, the search parameter is one or more individual fields of a Java object. This search parameter is encrypted by the Encryptor and this encrypted search parameter is provided as input to the Database Manager in order to perform the search operation. All the records which contain the encrypted search parameter as one of its column are returned by the Database Manager. These records are decrypted by the Field Decryptor to get the required Java objects

The Algorithm 4 shows the implementation of search operation in Field approach.

Algorithm 4 Search operation in Field approach

```
1: procedure SEARCH(fieldname, fieldvalue)        ▷ input search string fieldname, fieldvalue
2:     encrypted_field_value ← ENCRYPTFIELDVALUES(fieldvalue)
3:     encrypted_record ← SEARCHRECORDTABLE(field_name, encrypted_field_value)
4:     decrypted_record ← DECRYPTRECORD(encrypted_record)
5: end procedure
```

For example, consider the contacts shown in Table I and we want to retrieve all the contact records whose last name is "chandra". First we encrypt the input search key and perform select query using the search key.

SELECT * from RecordTable WHERE
contacts_last_name = ENC_VALUE("chandra"

This query returns a list of records whose last name is "chandra".

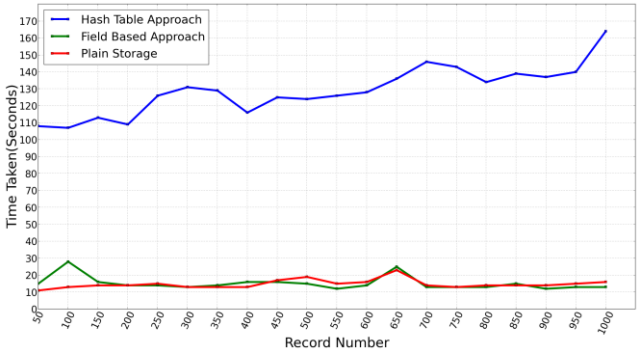

Fig. 5. Time analysis for save operation on HTC One X.

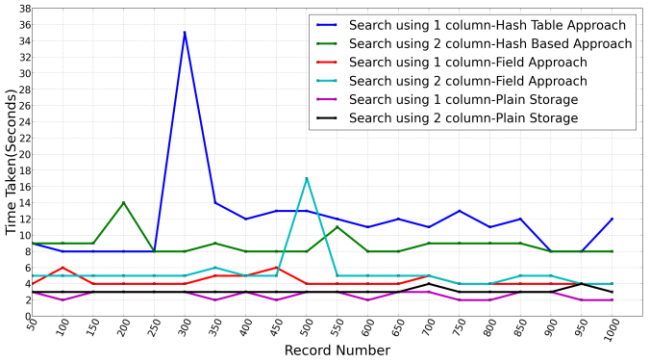

Fig. 6. Time analysis for search operation on HTC One X.

IV. EXPERIMENTAL RESULTS

The approaches discussed in Section III are implemented and tested on HTC One X, Micromax Unite 2 and Moto E. The device configurations [9]-[11] are shown in Table XI The performance of the all the approaches is calculated on the sample contacts schema represented in Table I. All the

observations are recorded on the devices mentioned in table XI in Annexure A.

Fig. 5 shows the observations for save operation in Hash table approach, Field based approach and Plain storage on HTC One X Device.

Fig. 6 shows the observations for search operations in Hash Table approach, Field based approach and Plain storage on HTC One X device.

Fig. 7 shows the observations for save operation in Hash Table approach on HTC One X, Micromax Unite 2 and Moto E devices.

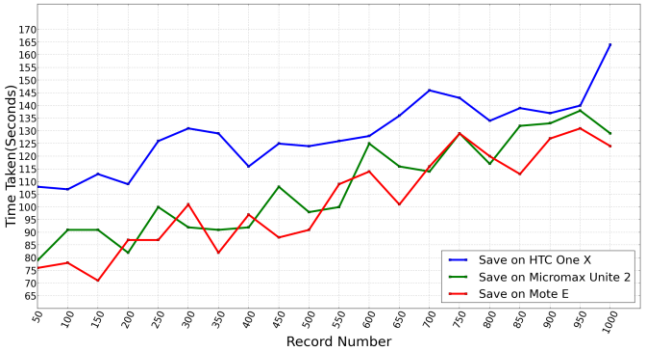

Fig. 7. Time analysis for save operation in Hash approach on HTC One X, Micromax Unite 2 and Moto E.

Fig. 8 shows the observations for search operation in Hash Table approach on HTC One X, Micromax Unite 2 and Moto E devices.

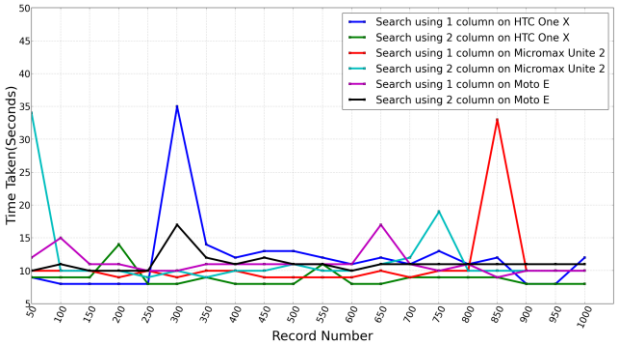

Fig. 8: Time analysis for search operation in Hash approach on HTC One X, Micromax Unite 2 and Moto E.

Fig. 9 shows the observations for save operation in Field based approach and Plain storage on HTC One X, Micromax Unite 2 and Moto E devices.

Fig. 10 shows the observations for search operation using 1column in Field based approach and Plain storage on HTC One X, Micromax Unite 2 and Moto E devices.

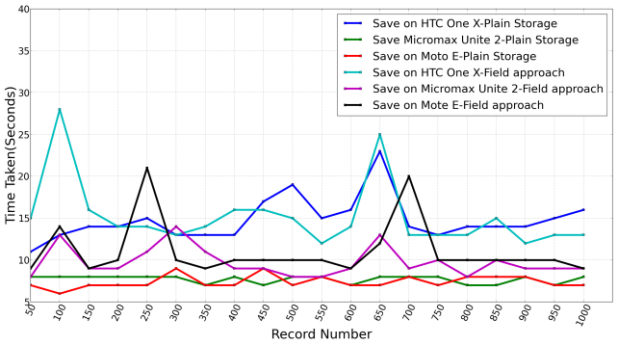

Fig. 9. Time analysis for save operation in Plain storage andField based approach on HTC One X, Micromax Unite 2 and Moto E.

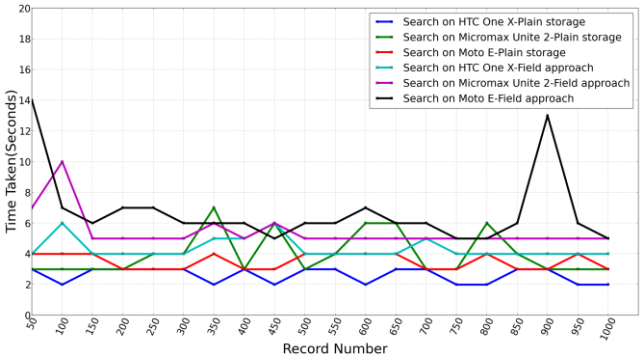

Fig. 10. Time analysis for search operation using 1 column in Plain storage and Field based approach on HTC One X, Micromax Unite 2 and Moto E.

Fig. 11 shows the observations for search operation using 2columns in Field based approach and Plain storage on HTC One X, Micromax Unite 2 and Moto E devices.

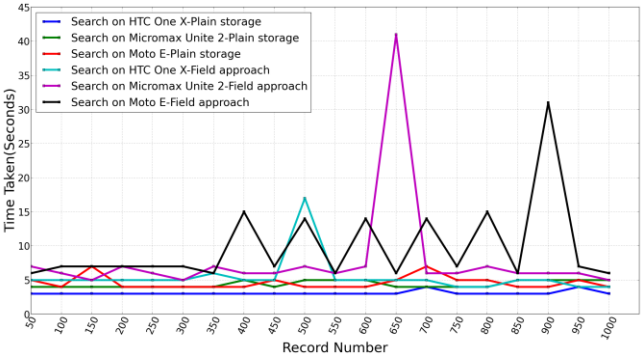

Fig. 11. Time analysis for search operation using 2 columns in Plain storage and Field based approach on HTC One X, Micromax Unite 2 and Moto E.

TABLE VIII: SAVE OPERATION IN HASH TABLE AND FIELD BASED APPROACH (OBSERVATIONS FOR 1000 RECORDS)

	Hash Table Approach			Field Approach		
	HTC One X	Micr-omax Unite 2	Moto E	HTC One X	Micr-omax Unite 2	Moto E
Median	131	104	10	14	9	14
Standard	18.08	60.90	22.85	4.78	3.44	4.86
Minimum Value	93	70	9	11	8	10
Maximum Value	286	1213	480	79	59	78
1st Quartile	119	91	10	13	8	13
3rd Quartile	142	120	11	16	11	16

TABLE IX: SEARCH OPERATION IN HASH TABLE (OBSERVATIONS FOR 1000 RECORDS)

	HTC One X		Micromax Unite 2		Moto E	
	1 Col-umn	2 Col-umn	1 Col-umn	2 Col-umn	1 Col-umn	2 Col -
Median	12	9	10	10	11	11
Standard	7.56	2.17	2.41	3.13	2.15	2.59
Minimum Value	7	7	9	9	9	9
Maximum Value	159	35	60	38	36	38
1st Quartile	9	8	9	10	10	10
3rd Quartile	13	9	10	10	11	11

TABLE X: SEARCH OPERATION IN FIELD BASED APPROACH (OBSERVATIONS FOR 1000 RECORDS)

	HTC One X		Micromax Unite 2		Moto E	
	1 Col-umn	2 Col-umn	1 Col-umn	2 Col-umn	1 Col-umn	2 Col-umn
Median	5	5	5	6	6	7
Standard Deviation	1.37	1.47	2.40	3.41	2.92	3.13
Minimum Value	3	4	4	5	5	5
Maximum Value	19	21	40	76	33	34
1st Quartile	4	5	5	6	6	6
3rd Quartile	5	5	6	6	7	7

Tables VIII, IX and X show the median, standard deviation, min value, max value, 1st quartile and 3rd quartile for the Hash Table and Field based approaches. All the observations are done taken on a sample data of 1000 records and is performed on all the three devices mentioned in Table XI in Annexure A.

ANNEXURE A

TABLE XI: DEVICE CONFIGURATION

	HTC One X	Micromax Unite 2	Moto E
OS	Android 4.2.2	Android 4.2.2	Android 4.4.4
Chipset	Nvidia Tegra 3	Mediatek MT6582	Qualcomm Snapdragon 200
CPU	Quad-core 1.5 GHz	Quad-core 1.3 GHz Cortex-A7	Dual-core 1.2 GHz Cortex-A7
GPU	ULP GeForce	Mali-400MP2	Adreno 302
RAM	1 GB	**1** GB	1 GB

V. CONCLUSION

The increase in the mobile device usage has increased the number of mobile applications which in turn has increased the storing of sensitive data on these devices. This has put many challenges for securely storing and searching on the sensitive information held by mobile applications, in order to address these challenges no approach specific to mobile environment are available. We have discussed two approaches for saving and querying on the data. The two approaches discussed in our paper are Hash Table approach and the Field based approach. The Field based approach gives good results when compared with the Hash Table based approach. Both the save and search operations in Field based approach fared better than the Hash Table approach. The Field based approach also fared near to the Plain storage method. After calculating the considerable overhead of encryption in the Field based and Hash Table approach, the overhead that is added in Field based method is 1.04% while it is 7.44 times more in Hash Table based approach. The Plain storage method is faster than all the other available method, while from the encrypted approaches the Field based approach fares near to Plain storage approach and is considered better than Hash Table approach in terms of time taken. From our experimental results we observed that the Field based approach is better than the Hash Table based approach in terms of time taken.

ACKNOWLEDGMENT

The authors acknowledge Department of Electronics and Information Technology (DeitY), Ministry of Communications & IT, Government of India for supporting this work.

REFERENCES

[1] E. Shmueli, R. Vaisenberg, Y. Elovici, and C. Glezer, "Database encryption - An overview of contemporary challenges and design considerations," *S1GMOD Record*, vol. 38, no. 3, 2009.
[2] D. X. D. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted Data," in *Proc. IEEE Symposium on Security and Privacy*, pp. 44-55, 2000.
[3] H. Hacigumus, B. Iyer, C. Li, and S. Mehrotra, "Executing SQL over encrypted data in the database service provider model," in *Proc. the ACM SIGMOD Conference*, 2002, pp. 216-227.
[4] H. A. Park, D. Lee, J. Zhan, and G. Blosser, "Efficient keyword index search over encrypted documents of Groups," *Intelligence and Security Informatics*, vol. 2, 2008.
[5] E. J. Goh, "Secure indexes," *Journal of Software*, pp. 1-19, 2004.
[6] Z. Wang, A. Tang, and W. Wang, "Fast query over encrypted data based on b+ tree," in *Proc. International Conference on Apperceiving Computing and Intelligence Analysis (ICACIA)*, 2009, pp. 23-25.
[7] H. Zhu, J. Cheng, and R. Jin, "Execution query over encrypted character strings in databases," *Frontier of Computer Science and Technology*, 2007, pp. 90-97.
[8] M. Alhanjouri and A. M. A. Deraw, "A new method of query over encrypted data in database using hash map," *International Journal of Computer Applications*, vol. 41, no. 4, March 2012.
[9] HTC One X Specifications. [Online]. Available: http://www.htc.com/in/smartphones/htc-o ne-x/#/
[10] Micromax Unite 2 Specifications. [Online]. Available: http://www.micromaxinfo.com/mobiles/smartphones/canvas/Unite-2-A106
[11] Motorola Moto E Specifications. [Online]. Available: http://www.motorola.com/us/consumers/shop-all-mobile-phones/Moto-E-pdp/moto-e.html

A. Vivek Chandra received his bachelor degree in information technology from JNT University, Hyderabad in 2012.
Presently he is working as a project engineer at Centre for Development of Advanced Computing. His research interests include mobile security and linux.

K. Harish Kumar received his bachelor degree in information technology from JNT University, Hyderabad in 2012.
Presently he is working as a project engineer at Centre for Development of Advanced Computing. His research interests include android platform security.

M. K. Chaithanya is currently working as a technical officer at C-DAC, Hyderabad. He received his B.Tech (ECE) degree and M.Tech (CS) from JNT University, Hyderabad in 2005 and 2010 respectively. He holds certifications in CEH and GSSP-Java.
He also holds a significant expertise in the area of Android application development, enterprise java application development, windows system programming, apache module development. His areas of interest include network security, mobile security, ubiquitous computing, network protocol design and Tizen OS.

Poonguzhali P. is currently working as a senior technical officer at C-DAC, Hyderabad. She received her B.E (ECE) degree from Anna University and MS in electronics and communication from JNT University, Hyderabad in 2005 and 2011 respectively. She holds certifications in CEH, ECSA, GSSP-Java.
She also holds a significant expertise in the area of Android application development, enterprise java application development, network simulators, Wireless sensor networks and ASIC & FPGA Design. Her areas of interest include mobile security, network security, reconfigurable computing systems, FPGA & ASIC designs and ubiquitous computing.

Mahesh U Patil received his master degree in electronics and communication.
Presently he is working as a principal technical officer at Centre for Development of Advanced Computing. His research interests include mobile security and embedded systems.