DOUBLE 0-AUTHU

DATABASE MANAGEMENT SYSTEMS

PROJECT REVIEW

- AIM

To develop a double wall authentication system that is more secure than the current industry standard (0-Authu).

- ABSTRACT

In the database the password is not stored directly to secure the data of consumer from the database administrator. To do this there are different algorithms in market, the leading algorithm used by Google's database is 0-Authu. Basically, in this algorithm a hashed value or an encrypted value is stored in the database as User UID. This is then used as a token by the application to send to the database and then verified for login so that the password is cannot be seen by the administrator. But the User UID can be traced back to the password by different ways like brute force etc. hence this is where the modified algorithm comes into picture. What the application does is that, it takes the password generates a key for encryption of the password and then that encrypted password is in turn sent for 0-Authu processing so that there are two different layers of protection. Each time a user signs up the above process takes place in the main activity. When the user has already signed up, the Firebase authentication saves the email ID and the generated User UID in the database. Along with that the last sign in time and the ID created time is also saved. The user can also use existing google or some other social account to login using the Firebase Authentication. There can be multiple mail providers in the database. For now, there is no verified provider but that can be added as a functionality too in the further stages. The second part is the Login Activity, when tapping on the sign in option then an activity intent is used to transfer to the Login Activity. This is where a user a login when he/she has already signed in. When the user enters the login ID and the password. The password is then encrypted with the same algorithm and key as previously done while signup. And then is passed though 0-Authu and after that the value that is generated is sent to the database as a token to verify for User UID and if the value is same then the user is logged and sent to the home activity though another activity intent. Here a shared preference value is used to save the state of the login so that whenever the user has logged in once the user does not have to login every time. Only login once logged out of the application. The logout button is present on the Home Activity that takes user to the Sign-up screen though an activity intent. The whole application is 9.2 MB so can also be applied as a wrapper to existing application. Which makes it easier to implement the new algorithm on the existing algorithm. There is a detailed account if the resources used in then introduction section and the details about the existing and the new algorithms are in the Literary Review.

- INTRODUCTION

This project in an android application designed on Android Studio. The database used is firebase by google.

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development. The official language for Android development is Java. Large parts of Android are written in Java and its APIs are designed to be called primarily from Java. It is possible to develop C and C++ app using the Android Native Development Kit (NDK), however it isn't something that Google promotes. According to Google, "the NDK will not benefit most apps. As a developer, you need to balance its benefits against its drawbacks.

The job of JAVA virtual machines is to interpret the bytecode. Java is a programming language first released by sun microsystems back in 1995. It can be found on many different types of devices from smartphones, to mainframe computers. Java doesn't compile to native processor code but rather it relies on a "virtual machine" which understands an intermediate format called java bytecode. Each platform that runs java needs a virtual machine (vm) implementation. On android the original vm is called dalvik.

Layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of view and viewgroup objects.

There are two declare a layout:

- **Declare ui elements in xml**. Android provides a straightforward xml vocabulary that corresponds to the view classes and subclasses.
- **Instantiate layout elements at runtime**. Your app can create view and viewgroup objects (and manipulate their properties) programmatically.

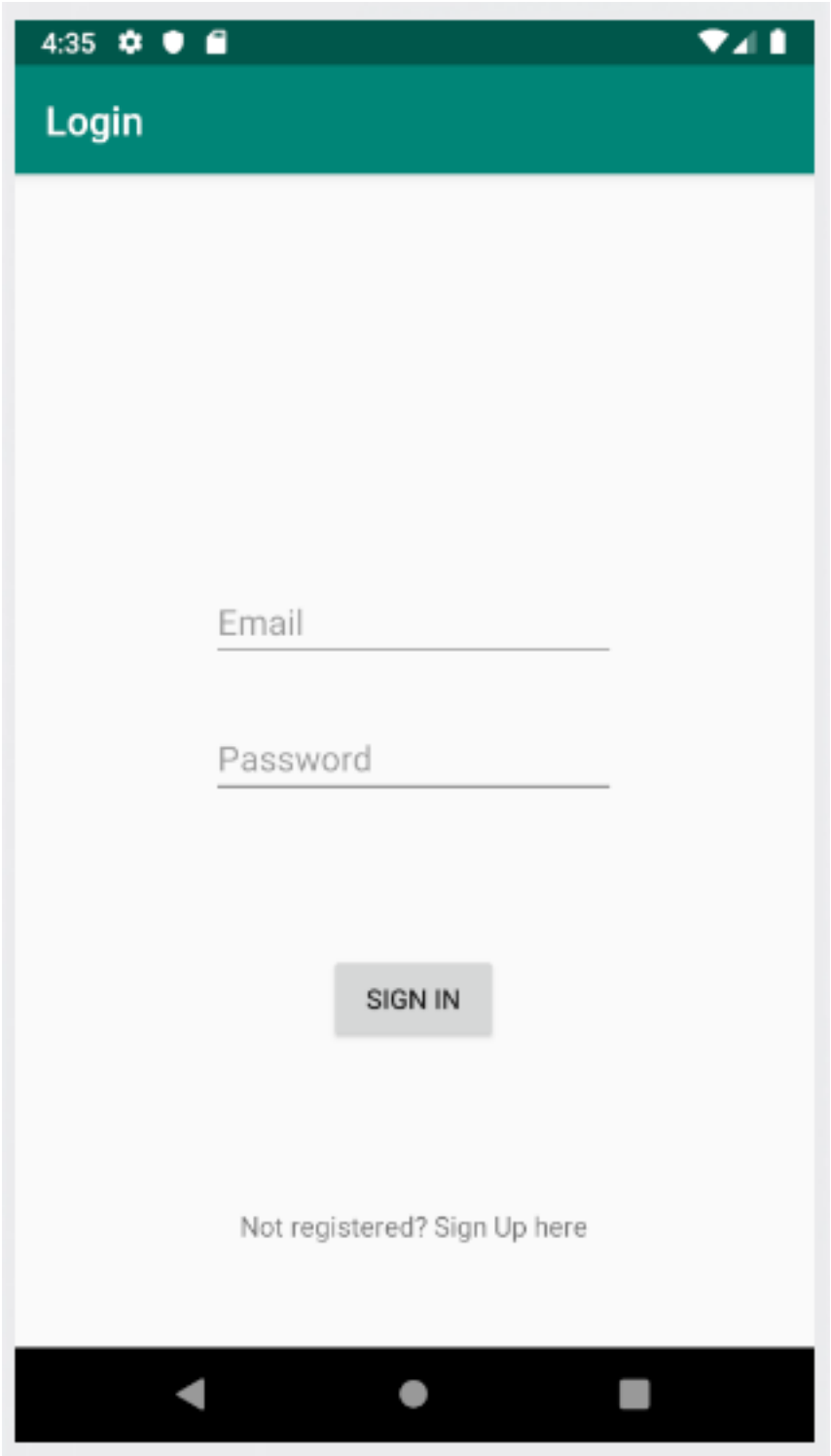
Firebase is a NoSQL database. Nosql is an approach to database design that can accommodate a wide variety of data models, including key-value, document, columnar and graph formats. Nosql, which stand for "not only sql," is an alternative to traditional relational databases in which data is placed in tables and data schema.

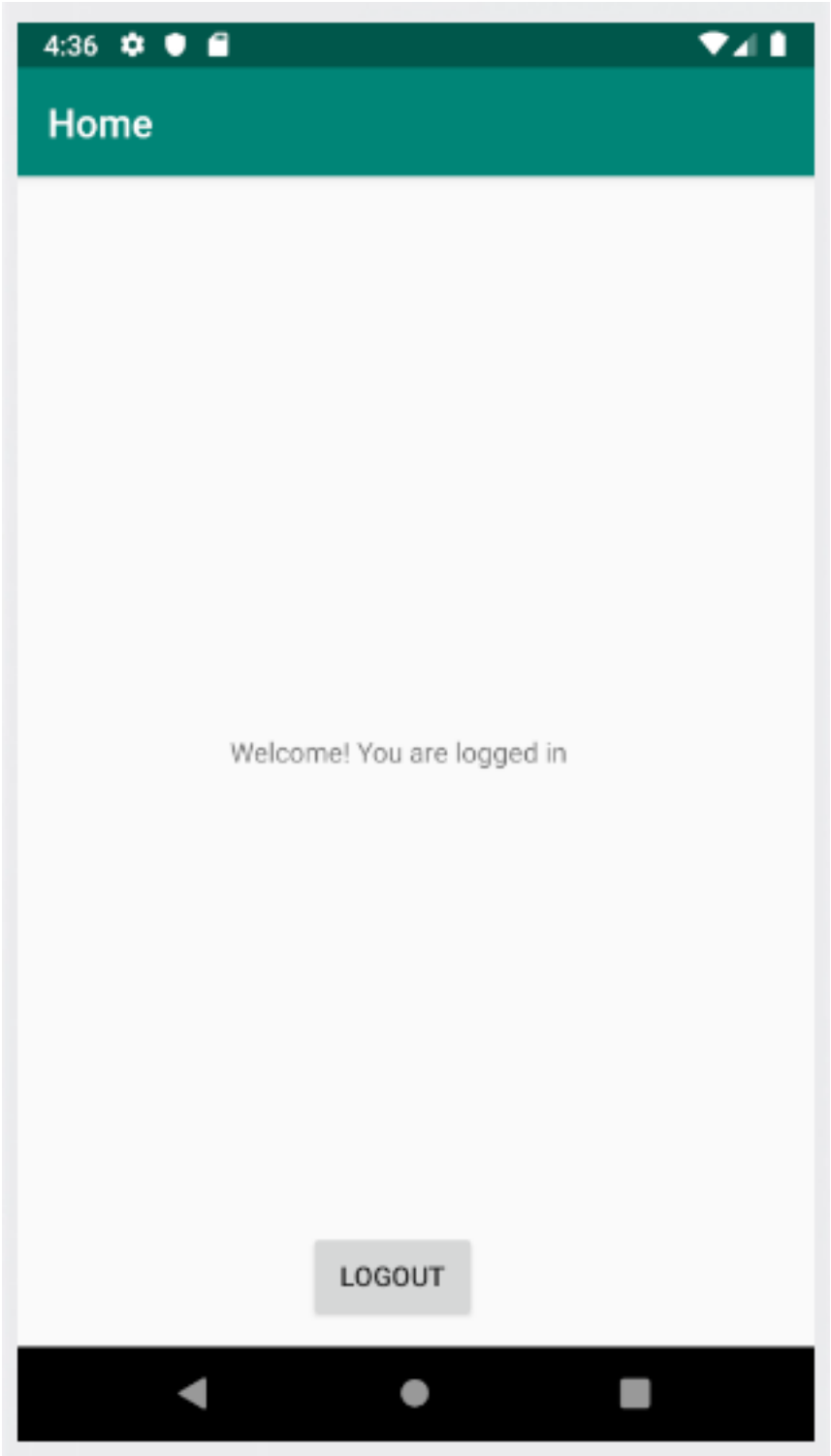
Firebase is a mobile and web application development platform developed by firebase, inc. In 2011, then acquired by google in 2014.

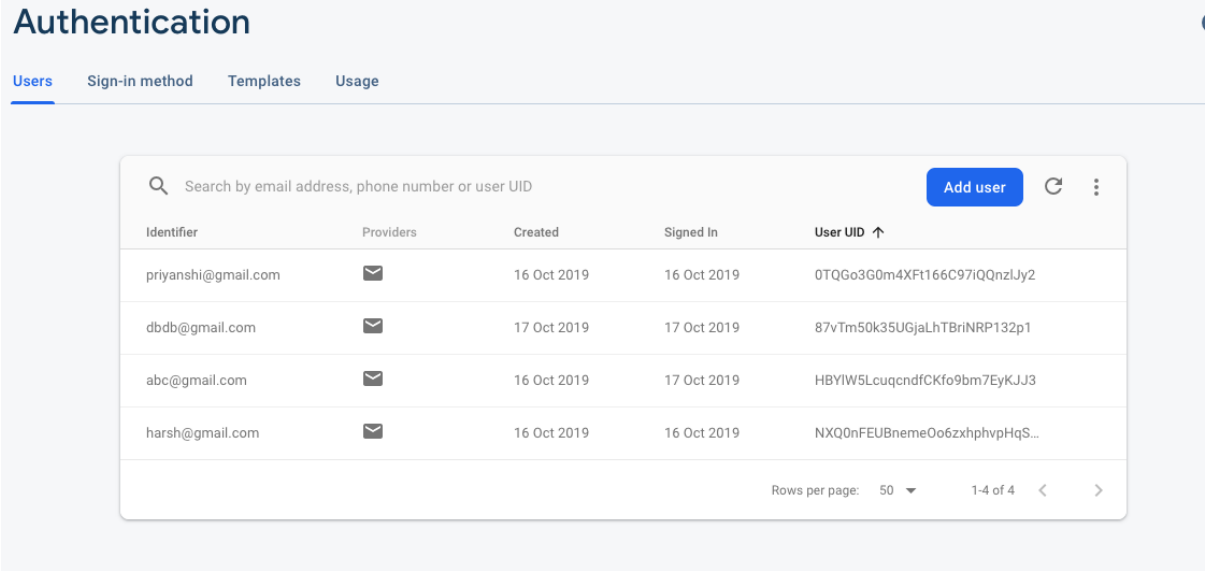
Firebase auth is a service that can authenticate users using only client-side code. It supports social login providers facebook, github, twitter and google (and google play games). Additionally, it includes a user management system whereby developers can enable user authentication with email and password login stored with firebase.

The above resources are used to get the modified algorithm for the purpose of this project. This is now followed by screenshots of the application.


4:35
DBMS PROJECT 2
abc@gmail.com
SIGN UP
Already have
an account?


4:35
Login
Email
Password
SIGN IN
Not registered? Sign Up here


4:36
Home
Welcome! You are logged in
LOGOUT



- **LITERARY REVIEW**

Firebase Authentication uses an internally modified version of scrypt to hash account passwords. Even when an account is uploaded with a password using a different algorithm, Firebase Auth will rehash the password the first time that account successfully logs in. Accounts downloaded from Firebase Authentication will only ever contain a password hash if one for this version of scrypt is available, or contain an empty password hash otherwise.

A simple password-based encryption utility is available as a demonstration of the scrypt library. It can be invoked as scrypt {key} {salt} {rounds} {memcost} [-P]. The utility will ask for a plain text password and output a hash upon success. This hash should be encoded to base64 and compared to the password hash of the exported user account.

- {key} - The signer key from the project's password hash parameters. This key must be decoded from base64 before being passed to the utility.
- {salt} - Concatenation of the password salt from the exported account and the salt separator from the project's password hash parameters. Each half must be decoded from base64 before concatenation.
- {rounds} - The rounds parameter from the project's password hash parameters.
- {memcost} - The mem_cost parameter from the project's password hash parameters.
- [-P] - An optional -P may also be supplied to allow for the raw text password to be read from STDIN.

Sample Password hash parameters from Firebase Console:

```
hash_config {
  algorithm: SCRYPT,
  base64_signer_key: jxspr8Ki0RYycVU8zykbdLGjFQ3McFUH0uiiTvC8pVMXAn210wjLNmdZJzxUECKbm0QsEmYUSDzZvpjeJ9WmXA==,
  base64_salt_separator: Bw==,
  rounds: 8,
  mem_cost: 14,
}
```

This is the existing hashing algorithm and the function used for authentication.

```
mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()) {
                    // Sign in success, update UI with the signed-in user's information
                    Log.d(TAG, "signInWithEmail:success");
                    FirebaseUser user = mAuth.getCurrentUser();
                    updateUI(user);
                } else {
                    // If sign in fails, display a message to the user.
                    Log.w(TAG, "signInWithEmail:failure", task.getException());
                    Toast.makeText(EmailPasswordActivity.this, "Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                    updateUI(null);
                }

                // ...
            }
        });
```

For the **PURPOSED ALGORITHM,** when the password is passed in the above function, it is encrypted using AES algorithm before passing so another layer of protection is added.

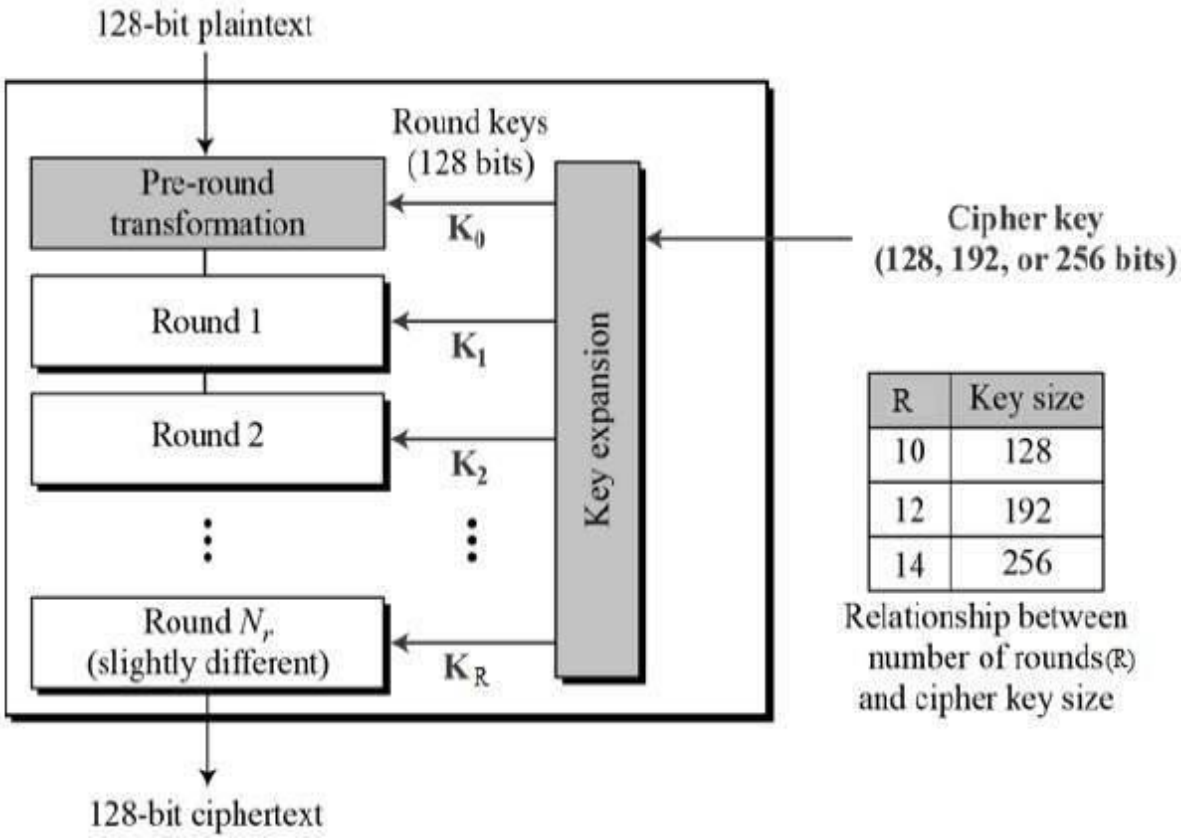
AES ENCRYPTION ALGORITHM

Aes is an iterative rather than feistel cipher. It is based on 'substitution–permutation network'. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, aes performs all its computations on bytes rather than bits. Hence, aes treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike des, the number of rounds in aes is variable and depends on the length of the key. Aes uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original aes key.

The schematic of aes structure is given in the following illustration −

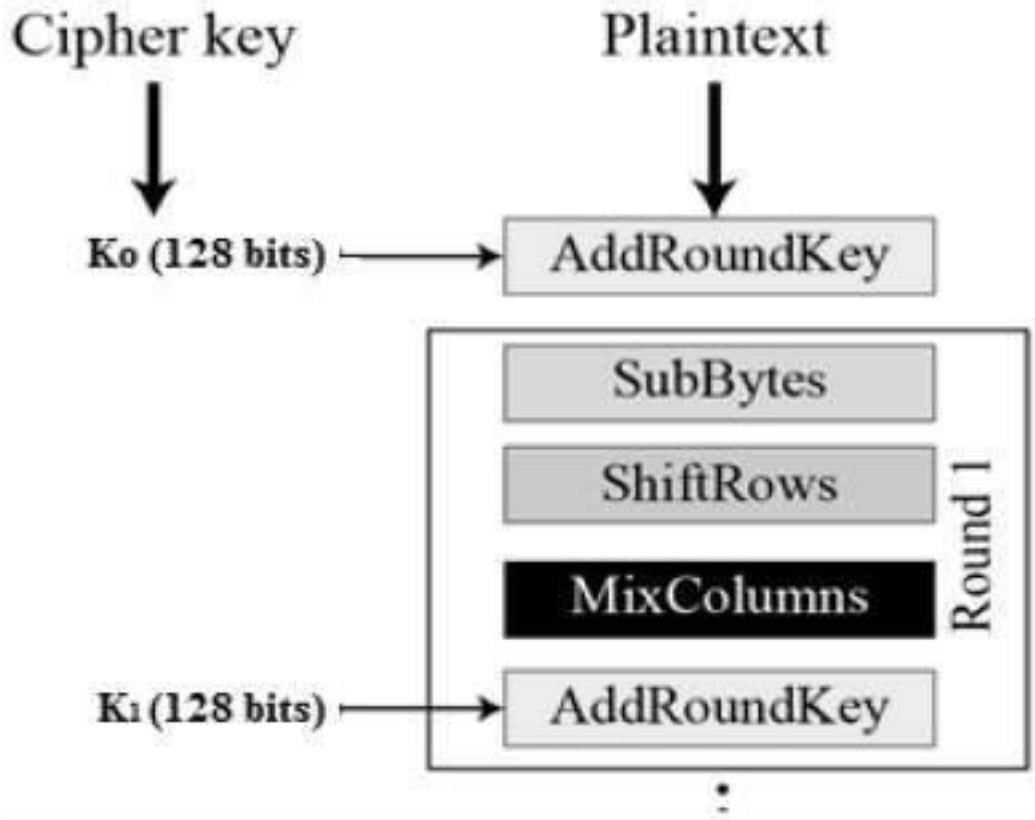


R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size

Encryption process

Here, we restrict to description of a typical round of aes encryption. Each round comprise of four sub-processes. The first round process is depicted below −



Byte substitution (subbytes)

The 16 input bytes are substituted by looking up a fixed table (s-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that 'fall off' are re-inserted on the right side of row. Shift is carried out as follows −

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

Mixcolumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are xored to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption process

The process of decryption of an aes ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

- Add round key
- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a feistel cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

NEW ALGORITHM I.E. FIREBASE AUTHNTICATION + AES ENCRYPTION

```
                //AES ENCRYPTION CALL
                try {
                    outputString = encrypt(pwd, email);
                } catch (Exception e) {
                    e.printStackTrace();
                }
                printcatAES(outputString);
                if(email.isEmpty()){
                    emailId.setError("Please enter email id");
                    emailId.requestFocus();
                }
                else  if(pwd.isEmpty()){
                    password.setError("Please enter your password");
                    password.requestFocus();
                }
                else  if(email.isEmpty() && pwd.isEmpty()){
                    Toast.makeText( context: LoginActivity.this, text: "Fields Are Empty!",Toast.LENGTH_SHORT).show();
                }
                else  if(!(email.isEmpty() && pwd.isEmpty())){
                    mFirebaseAuth.signInWithEmailAndPassword(email, pwd).addOnCompleteListener( activity: LoginActivity.this, (task) → {
                            if(!task.isSuccessful()){
                                Toast.makeText( context: LoginActivity.this, text: "Login Error, Please Login Again",Toast.LENGTH_SHORT).show();
                            }
                            else{
                                Intent intToHome = new Intent( packageContext: LoginActivity.this,HomeActivity.class);
                                startActivity(intToHome);
                            }
                    });
                }
                else{
                    Toast.makeText( context: LoginActivity.this, text: "Error Occurred!",Toast.LENGTH_SHORT).show();

                }

            }
        });
    public String encrypt(String Data, String passwords) throws UnsupportedEncodingException, NoSuchAlgorithmException, InvalidKeyException, BadPaddingException, I
        SecretKeySpec key = generateKeys(passwords);
        Cipher c = Cipher.getInstance(AES);
        c.init(Cipher.ENCRYPT_MODE,key);
        byte[] encVal = c.doFinal(Data.getBytes());
        String encryptesValue = Base64.encodeToString(encVal, Base64.DEFAULT);
        return encryptesValue;
    }
    public SecretKeySpec generateKeys(String passwords) throws NoSuchAlgorithmException, UnsupportedEncodingException {
        final MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] bytes = passwords.getBytes( charsetName: "UTF-8");
        digest.update(bytes,  offset: 0, bytes.length);
        byte[] key = digest.digest();
        SecretKeySpec secretKeySpec = new SecretKeySpec(key,  algorithm: "AES");
        return secretKeySpec;
    }
}
```

This way a new dynamic key is generated on the system and the protection is increased. Hence making it more difficult to get the password back.

- **CONCLUSION**

The new algorithm adds another layer of protection to the sign in application and then the value stored in database is first encrypted and then hashed. This adds to the integrity of the database. The following screenshot shows the new encrypted and hashed value which is generated after then encryption by the Firebase Authentication algorithm.

The password here is - abcdef

```
.dbmsproject2 I/BiChannelGoogleApi: [FirebaseAuth: ] getGoogleApiForMethod() returned Gms: com.google.firebase.auth.api.internal.zzao@f8e29d9
.dbmsproject2 D/LoginActivity: Encrypted Password by JAVA cipher = ?????I?? ???h?>
.dbmsproject2 D/LoginActivity: Encrypted Password by AES = 2n8Iql2WXbrFt/I/qhH6Qw==
.dbmsproject2 I/BiChannelGoogleApi: [FirebaseAuth: ] getGoogleApiForMethod() returned Gms: com.google.firebase.auth.api.internal.zzao@f8e29d9
.dbmsproject2 D/FirebaseAuth: Notifying id token listeners about user ( HBYlW5LcuqcndfCKfo9bm7EyKJJ3 ).
.dbmsproject2 D/FirebaseAuth: Notifying auth state listeners about user ( HBYlW5LcuqcndfCKfo9bm7EyKJJ3 ).
```

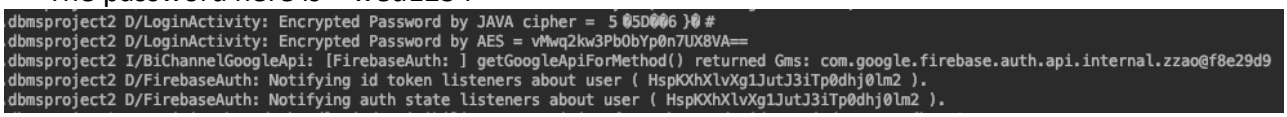
The password here is - naming

```
dbmsproject2 D/LoginActivity: Encrypted Password by JAVA cipher = p?23 ?3?? 5??i
dbmsproject2 D/LoginActivity: Encrypted Password by AES = skjUH82/dH2XSaM7iKqBDg==
dbmsproject2 I/BiChannelGoogleApi: [FirebaseAuth: ] getGoogleApiForMethod() returned Gms: com.google.firebase.auth.api.internal.zzao@f8e29d9
dbmsproject2 D/FirebaseAuth: Notifying id token listeners about user ( A3I5dNaf24MQ7q8JTSvmf5wd8X52 ).
dbmsproject2 D/FirebaseAuth: Notifying auth state listeners about user ( A3I5dNaf24MQ7q8JTSvmf5wd8X52 ).
```

The password here is – 1234567

```
dbmsproject2 D/LoginActivity: Encrypted Password by JAVA cipher = 9C?3?GtJ?v? U~??
dbmsproject2 D/LoginActivity: Encrypted Password by AES = jw0emQ56oDQ4BVdNPhzUXA==
dbmsproject2 I/BiChannelGoogleApi: [FirebaseAuth: ] getGoogleApiForMethod() returned Gms: com.google.firebase.auth.api.internal.zzao@f8e29d9
dbmsproject2 D/FirebaseAuth: Notifying id token listeners about user ( aGMjalKAMGX2EKr8tnztH1C6LkW2 ).
dbmsproject2 D/FirebaseAuth: Notifying auth state listeners about user ( aGMjalKAMGX2EKr8tnztH1C6LkW2 ).
```

The password here is – wed1234



The full project is available on the following github link.
https://github.com/Harsh1210/Double-0-Authu

- **Reference:**

1. https://firebase.google.com/docs/auth
2. https://firebaseopensource.com/projects/firebase/firebase-android-sdk/
3. https://www.tutorialspoint.com/android/android_studio.htm
4. https://developer.android.com/training/basics/firstapp
5. https://firebase.google.com/docs/auth/?gclid=Cj0KCQjwoqDtBRD-ARIsAL4pviDysOQO5kFBuJ0WBcnJzAqug0PhvA3V90tXZyueitRGrubjeJdlJT8aAocKEALw_wcB
6. https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard#targetText=The%20Advanced%20Encryption%20Standard%2C%20or,world%20to%20encrypt%20sensitive%20data.
7. https://www.comparitech.com/blog/information-security/what-is-aes-encryption/
8. https://developer.android.com/studio/run/managing-avds
9. https://howtodoinjava.com/security/java-aes-encryption-example/
10. https://howtodoinjava.com/security/aes-256-encryption-decryption/