EncryptZip:
Compressed, Encrypted Containers on Android
Project Milestone

Alex Shah

4/7/18

1 Abstract

This project sets out to implement encryption on zip based containers on the Android platform. This was accomplished with tools available in the Android SDK and AES encryption implemented manually. Containers are created by zipping a folder's contents on the user's device. Zipping a folder compresses it to save file size. The zip container is then encrypted with AES as a serial object and the unpacked folder and insecure zip are removed from the device, unrecoverable due to Android's implementation of solid state storage on ext4 file systems.

2 Introduction

Encrypting files is a common practice to prevent unauthorized or unintended access to sensitive information. However, encrypting each file is only one facet of security in a system. Encrypting the container which sensitive information is held can prove more robust than encrypting on a per file basis, or in addition to it, in applications where the system cannot be entirely secured or in addition to measures such as disk based encryption. These three levels, file level, container level, and system/disk level encryption ensure that intended access is compartmentalized in nature and can ensure that a system is truly secure.

3 Background

While most desktop systems have applications to accomplish the container level of encryption, such as Veracrypt, Android does not have a native application and third party implementations use proprietary methods. The goal of this project is to use built in features and standard systems to create a container level encryption scheme on the Android mobile platform.

4 Methodology

Utilizing the Android SDK and available APIs provided by Google and within Java, it's perfectly feasible to encrypt files, zip them into a container, and encrypt the container itself. This also has an intended side effect of reducing the file size by using compression, as well as ensuring that containers are dynamically sized. In addition, it would then be trivial to compute the hash of a container for authenticating the contents of a container as an added measure of security. Users can also use built in Google drive support to store, share, or keep an encrypted zip container synchronized between two remote parties.

5 Experiments

Various examples of compression in Android can be found using Java's built in zip functionality. To implement directory unzipping in Android it is necessary to use a buffer and an InputStream. For example:

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.InputStream;
import java.util.zip.ZipEntry;
import java.util.zip.ZipInputStream;
import android.content.Context;
import android.util.Log;

public class Decompress {
    private static final int BUFFER_SIZE = 1024 * 10;

    public static void unzipFromAssets(Context context, String
        zipFile, String destination) {
               destination = context.getFilesDir().getAbsolutePath
                   ();
               InputStream stream = context.getAssets().open(
                   zipFile);
               unzip(stream, destination);
    }


    public static void unzip(InputStream stream, String
        destination) {
        byte[] buffer = new byte[BUFFER_SIZE];
        ZipInputStream zin = new ZipInputStream(stream);
               .
               .
               .
         zin.close();
    }
}
```

While Java contains its own implementation of AES, as well as easy to access Cipher key making and padding, we have created an implementation of AES in class, so it is necessary to use some aspects of Java's built in functions while also overriding some with functions we have created in class.

```
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;
```

6 Conclusion

Thus far, implementing components of the project have been successful for the parts individually. Combining these parts to create a cohesive application will be more difficult, but rewarding. The project is on track, and all that's left is to assemble the pieces, benchmark the performance, and optimize.