Data Encryption Application Whitepaper

Peng Wang, Atom Software Applications Engineer

Application Origination:
Intel SSG

Introduction

Mobile security is getting personal. Smartphones and tablets contain some of our most precious memories and personal details, including access to financial transactions and other sensitive information. Cybercriminals now develop five new threats every second. With the sharp growth in smartphones and tablets, the bad guys are aggressively attacking these devices and your information with a new range of insidious threats.

Here at Intel, we've begun creating small software applications to provide to independent software vendors (ISVs) as examples of how to use the security features on the platforms based on Intel® Architecture.

The purpose of this paper is to give developers a quick start by describing the basic Android* data encryption API and showing how to encrypt application data for the Android OS.

We recommend that you try out the features and compile the code as you read through the paper.

Data Encryption Code and Explanations

The DataEncrytion app has three major features: data encryption, progress bar, and performance indicators.

DataEncryption Class declaration

DataEncryption class extends the standard Activity class. It has private attributes for the logging, dialog, and UI components.

```java
public class DataEncryption extends Activity {

      private final String TAG = "SECURITY_DATA_ENCRYPTION";
      private final String FLAG_SUC = "SECURITY_DATA_ENCRYPTION_SUCCESSFULLY";
      private final String FLAG_FAIL = "SECURITY_DATAENCRYPTION_FAILED";

      private Boolean mDoneEncryption = false;
      private long mTimeTook = 0;

      ProgressThread progThread;
      ProgressDialog progDialog;
      int delay = 40;                   // Milliseconds of delay in the update loop
      int maxBarValue = 200;            // Maximum value of horizontal progress bar
      int typeBar;

      private Handler mHandler = new Handler();
      private String mIndicatorText;
      private TextView mIndicator;
      private SDPUtility mUtility;
      private ImageButton encyptionButton;
```

onCreate Member Function

The onCreate member function is called by the framework after an activity is created. Since the DataEncryption class extends the Activity class as you saw above, it is treated like an Activity when started. The initialization for the application occurs here.

```
	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.encryption);

		mUtility = new SDPUtility();
		mHandler.postDelayed(mUpdateTimeTask, 100);
		mIndicator = (TextView) findViewById(R.id.encryptionIndicator);

		typeBar = 0;
		encyptionButton = (ImageButton) findViewById(R.id.encryptionButton);
		encyptionButton.setOnClickListener(new View.OnClickListener() {
			public void onClick(View v) {
				showDialog(typeBar);
				new Thread(new Runnable() {
					public void run() {
						encryptVideo();

		} catch (Exception e) {
			Log.d(TAG, "Exception occurs in security encryption test.", e);
		}

	}
```

encrypt() function

The encrypt function does the actual work of data encryption. It first creates an input and output file stream. It then creates the “AES” cipher object and encryption key. Finally, the encryption function reads from the input file stream and writes the encrypted data to the output file stream.

```
	private void encryptVideo() {
		FileInputStream fis;
		try {
			fis = new FileInputStream(new
File("/sdcard/h264.mp4"));

			File outfile = new File("/sdcard/h264_enc.mp4");
			int read = 0;
			if (!outfile.exists())
				outfile.createNewFile();
			File decfile = new File("/sdcard/h264_dec.mp4");
			if (!decfile.exists())
				decfile.createNewFile();
			FileOutputStream fos = new FileOutputStream(outfile);
			FileInputStream encfis = new FileInputStream(outfile);
			FileOutputStream decfos = new
FileOutputStream(decfile);
			Cipher encipher = Cipher.getInstance("AES");
			Cipher decipher = Cipher.getInstance("AES");
			KeyGenerator kgen = KeyGenerator.getInstance("AES");

			SecretKey skey = kgen.generateKey();
			encipher.init(Cipher.ENCRYPT_MODE, skey);
			CipherInputStream cis = new CipherInputStream(fis,
encipher);
			decipher.init(Cipher.DECRYPT_MODE, skey);
			CipherOutputStream cos = new
CipherOutputStream(decfos, decipher);
			long start = System.nanoTime();
			Log.d("security", String.valueOf(start));
			while ((read = cis.read()) != -1) {
				fos.write((char) read);
				fos.flush();
			}
			long stop = System.nanoTime();
			Log.d("security", String.valueOf(stop));
			long seconds = (stop - start) / 1000000000;// for
seconds
			Log.d("security", String.valueOf(seconds));
			fos.close();
			mTimeTook = seconds;
			mDoneEncryption = true;

		} catch (FileNotFoundException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (IOException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchAlgorithmException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (NoSuchPaddingException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		} catch (InvalidKeyException e) {
			// TODO Auto-generated catch block
			e.printStackTrace();
		}

	}
```

Progress Bar

It often takes a while to encrypt big data files, so it is very useful to have a progress bar to inform the user when the encryption is complete. The progress bar is on a different thread and has both “Spinner” and “horizontal” styles.

```
     @Override
     protected Dialog onCreateDialog(int id) {
          switch(id) {
          case 0:                        // Spinner
               progDialog = new ProgressDialog(this);
               progDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
               progDialog.setMessage("encrypting data...");
               progThread = new ProgressThread(handler);
               progThread.start();
               return progDialog;
          case 1:                        // Horizontal
               progDialog = new ProgressDialog(this);

progDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);
               progDialog.setMax(maxBarValue);
               progDialog.setMessage("encrypting data...");
               progThread = new ProgressThread(handler);
               progThread.start();
               return progDialog;
          default:
               return null;
          }
     }

     final Handler handler = new Handler() {
          public void handleMessage(Message msg) {
               int total = msg.getData().getInt("total");
               progDialog.setProgress(total);
               if (mDoneEncryption){
                    dismissDialog(typeBar);
                    progThread.setState(ProgressThread.DONE);
               }
          }
     };
```

Performance Indicator

Performance is also very important for the data encryption. The DataEncryption app also provides some key indicators such as CPU utilization, memory utilization, and total time spent.

```
      private Runnable mUpdateTimeTask = new Runnable() {
            public void run() {
                  DecimalFormat df = new DecimalFormat("#.##");
                  String cpuUsage = df.format(mUtility.readUsage() * 100) + "%";
                  String memUsage = Long
                              .toString(mUtility
                                          .readMem((ActivityManager)
getSystemService(ACTIVITY_SERVICE)))
                              + "M";

                  mIndicatorText = "CPU usage: " + cpuUsage + "\nMemory usage: "
                              + memUsage;
                  if(mDoneEncryption){
                        mIndicatorText = mIndicatorText + "\nTotal Time Spent: " +
mTimeTook + " Seconds";
                  }

                  mIndicator.setText(mIndicatorText);

                  mHandler.postAtTime(this, SystemClock.uptimeMillis() + 2000);
            }
      };
```

Conclusion

By implementing code like the samples described in this paper, you can quickly learn how to encrypt data for Intel Android platforms and know how to measure the basic encryption performance.

About the author

Peng Wang is a member of the Intel Software and Solutions Group (SSG), Developer Relations Division, Atom High Touch Software Enabling team. Before joining SSG, Peng was leading the integration and validation team for the Ultra Mobile Group.