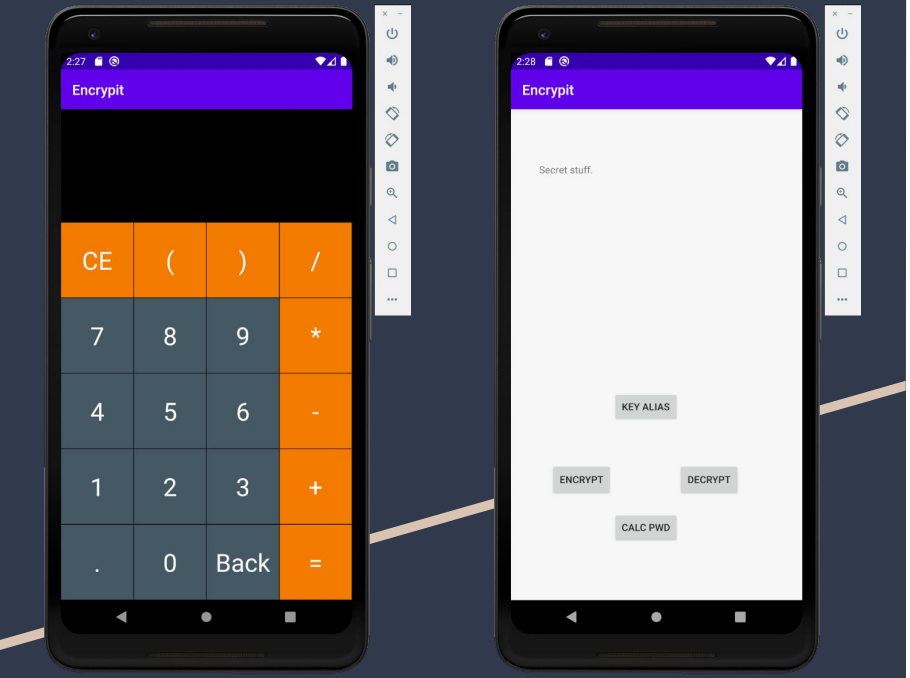
Encrypit

Hidden Calculator Encryption Vault

Steven Guarino

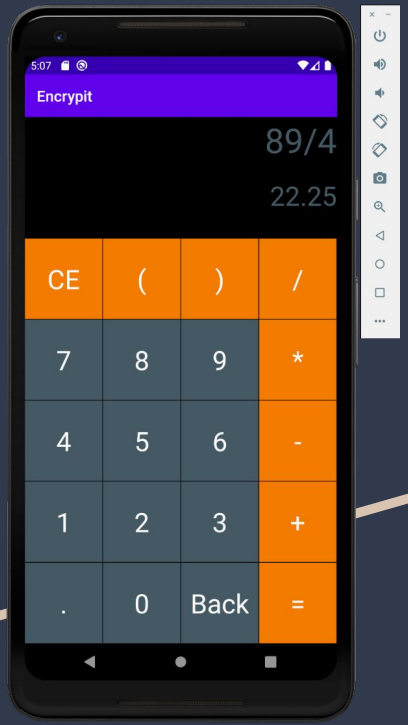
Overview



- A calculator app that hides a secret encrypted writing pad
- Functions as regular calculator until a specific key sequence is entered
- Hitting “=” with correct number sequence in calculator display reveals “hidden” app
- Calculator built with XML and resolves simple mathematical expressions
- Before calculations are executed the expression is checked against the hidden passcode, if correct encryption handler is launched

```
Equals.setOnClickListener { it: View!
    val expressionString = topDisplay.text.toString()
    if (expressionString == calcPasscode) {
        loadNewActivity()
    }
```

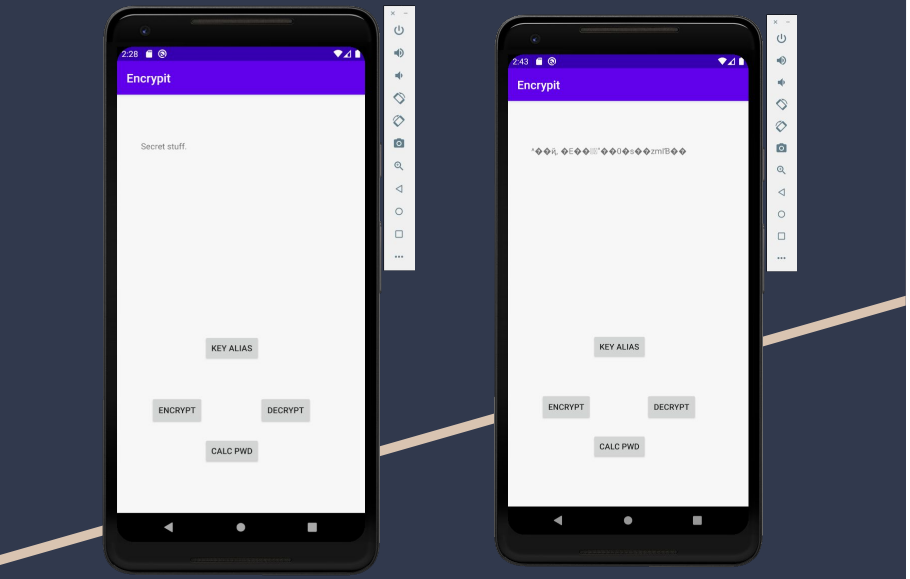
Calculator



- Assembly of textviews in a column of rows for buttons
- ExpressionBuilder library resolves simple mathematical expressions if passcode is not matched

```
val expression = ExpressionBuilder(topDisplay.text.toString()).build()
val result = expression.evaluate()
val longResult = result.toLong()
if (result == longResult.toDouble())
    bottomDisplay.text = longResult.toString()
else
    bottomDisplay.text = result.toString()
```

Encryption via Cipher



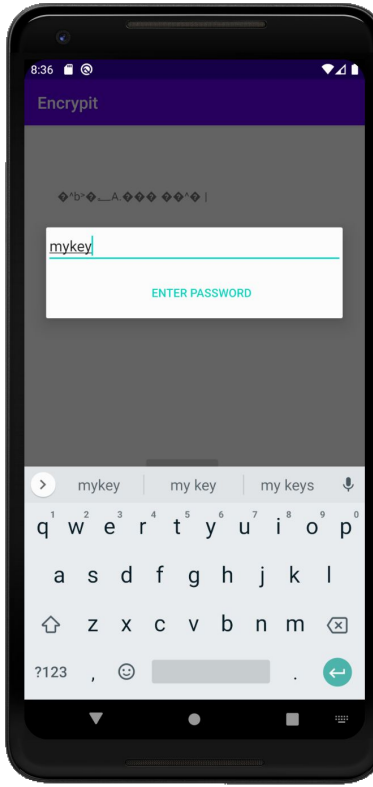
- Stores persistent file of encrypted text data using Android's Cipher class
- Cipher class instantiated with cryptographic algorithm, feedback mode, and padding scheme
- AES/GCM/NoPadding
- Key generation and storage using KeyStore
- Cipher is then instantiated with user key
- If no user key - dialogue pop up with encryption failure, otherwise success
- Define cipher IV reference

```
val cipher = Cipher.getInstance( transformation: "AES/GCM/NoPadding")
cipher.init(Cipher.ENCRYPT_MODE, secretKey)

// Passing IV to outer scope so it can by used by decryption
iv = cipher.iv
```

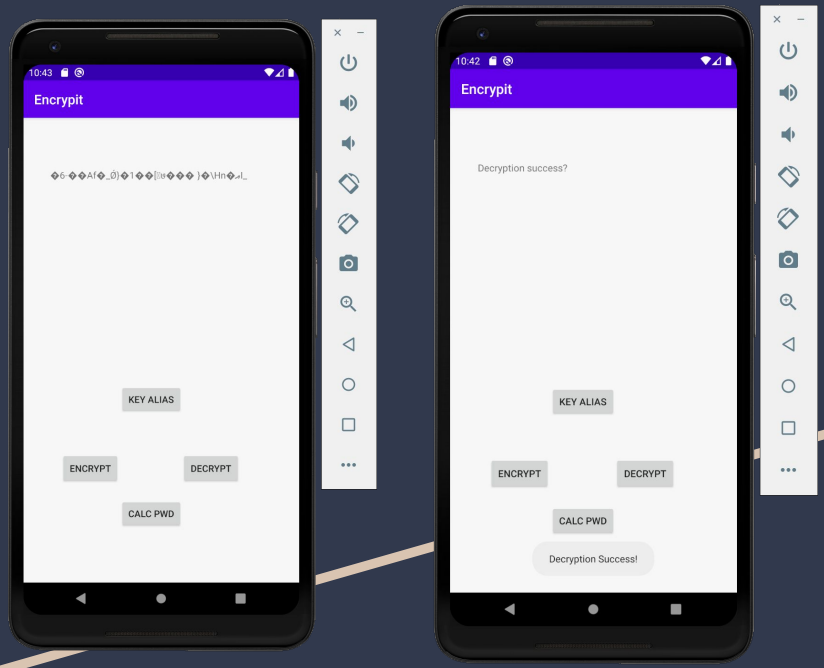
Key Generation via KeyStore

- Using KeyStore to generate and store keys used in cipher instantiation
- Key generator built with KeyGenParameterSpec, passing in same block and padding mode as cipher
- Key generator generates key with tag bit-length and encryption Cipher IV
- Key stored under “Key Alias” to be retrieved on decryption

```
// Instance of Androids KeyGenerator
val keyGenerator = KeyGenerator.getInstance(KeyProperties.KEY_ALGORITHM_AES,  provider: "AndroidKeyStore")

// build KeyGenParameterSpec with parameters of key, storing in keystore using keystore alias
val keyGenParameterSpec = KeyGenParameterSpec.Builder(
    // aliasHolder is user defined key alias
    aliasHolder,  purposes: KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT)
    .setBlockModes(KeyProperties.BLOCK_MODE_GCM)
    .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_NONE)
    .build()

keyGenerator.init(keyGenParameterSpec)
val secretKey = keyGenerator.generateKey()

val cipher = Cipher.getInstance( transformation: "AES/GCM/NoPadding")
cipher.init(Cipher.ENCRYPT_MODE, secretKey)
```



Decryption



- Generates KeyStore instance to retrieve user key used in decryption cipher generation
- Secret key generated through secretKeyEntry and unique key alias ID
- GCM Parameter Spec object requires total bit length of block cipher and same IV from encryption cipher instantiation
- Cipher is initialized with GCM spec and secret key in decryption mode and passed bytes from the file

```
// Creating keystore instance
val keyStore = KeyStore.getInstance( type: "AndroidKeyStore")
keyStore.load( param: null)

// Gets our secret key from keystore with alias
val secretKeyEntry = keyStore
    .getEntry(aliasHolder,  protParam: null) as KeyStore.SecretKeyEntry

val decrsecretKey: SecretKey = secretKeyEntry.secretKey
val cipher =
    Cipher.getInstance( transformation: "AES/GCM/NoPadding")
// Passing encryption IV to decryption cipher
val spec = GCMParameterSpec( tLen: 128, iv)
cipher.init(Cipher.DECRYPT_MODE, decrsecretKey, spec)

val encryption = cipher.doFinal(testfile.readBytes())
```

Limi