STRV

ANDROID KEYSTORE SYSTEM

REAL LIFE - USE CASE

STRV

“The Android Keystore system lets you store cryptographic keys in a container to make it more difficult to extract from the device.“

developer.android.com

KEYSTORE SYSTEM API NOTES

SINCE API 18 - Keystore Provider

- Let individual app store its own credentials that only the app itself can access.

SINCE API 14 - KeyChain

- Allows several apps to use the same set of credentials with user consent.

SINCE API 1 - Keystore

- SpongyCastle - repackaged BouncyCastle for Android

KEYSTORE PROVIDER API NOTES

SINCE API 18

- Known vulnerability without known patches.

SINCE API 19

- Still needs custom handling of LockScreen. App needs Admin privileges to force lock-screen.

SINCE API 21

- Still needs to force LockScreen manually, but using standard KeyguardManager .

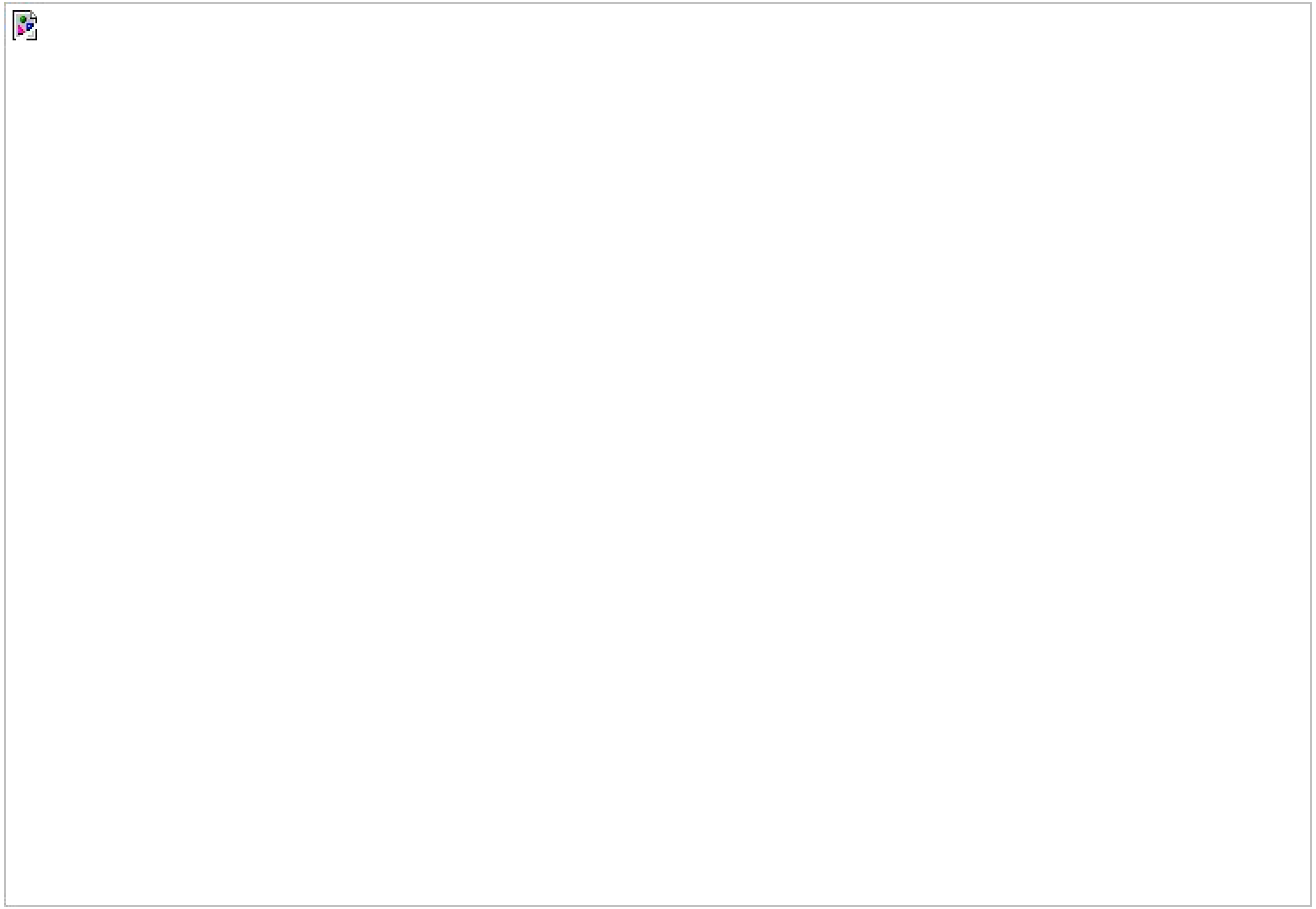
SINCE API 23

- Ability to define LockScreen force during key-pair generation.
- Addition symmetric cryptography (AES,HMAC)
- Enhancement for hardware-backed Keystore and many others...

KEYSTORE
USED
USE-CASE

ENCRYPT
DECRYPT
~~SIGN~~
~~VERIFY~~

Encrypt/Decrypt secret using Android Keystore

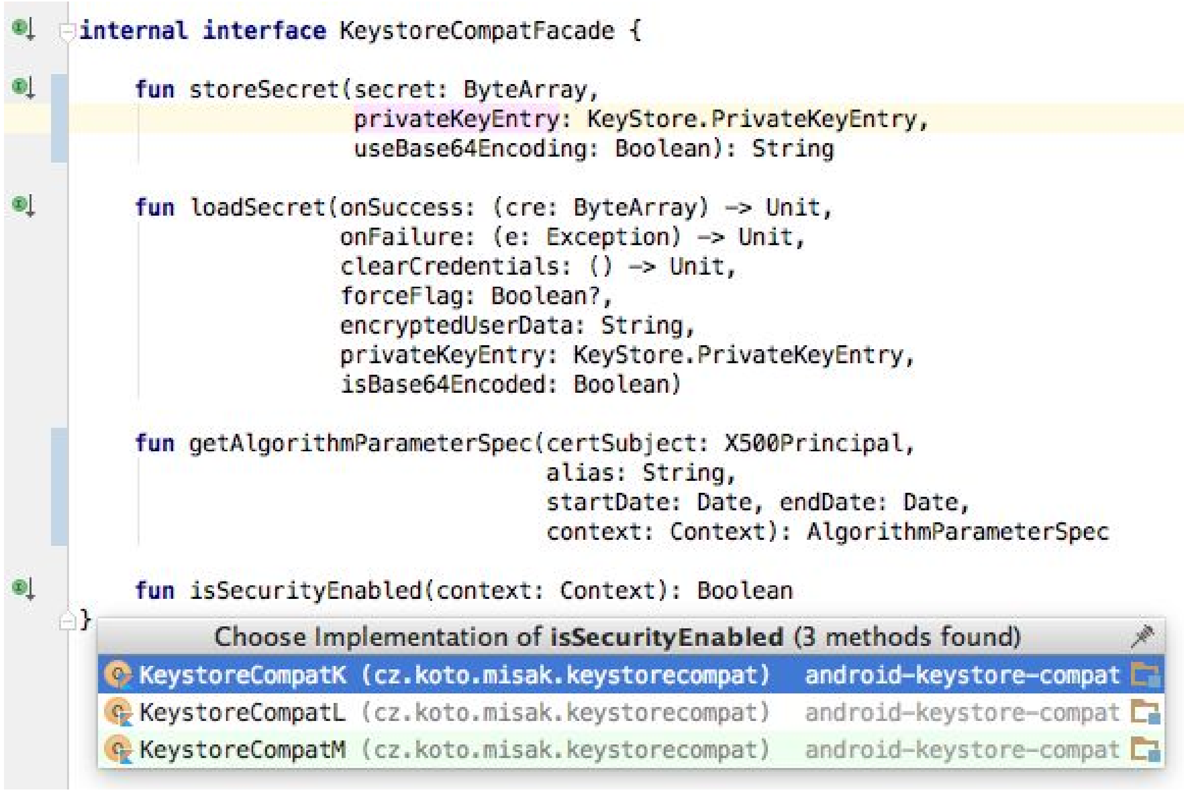


WHAT? WHY THE LIBRARY?

Separate Encryption/Decryption mechanism and make following features (including all future improvements) reusable as the one mechanism:

- Android-version specific crypto handling
- Android-version specific lock-screen handling
- Root detection handling
- Additional intent/hashing utilities

ANDROID VERSION-SPECIFIC CRYPTO HANDLING 1/3

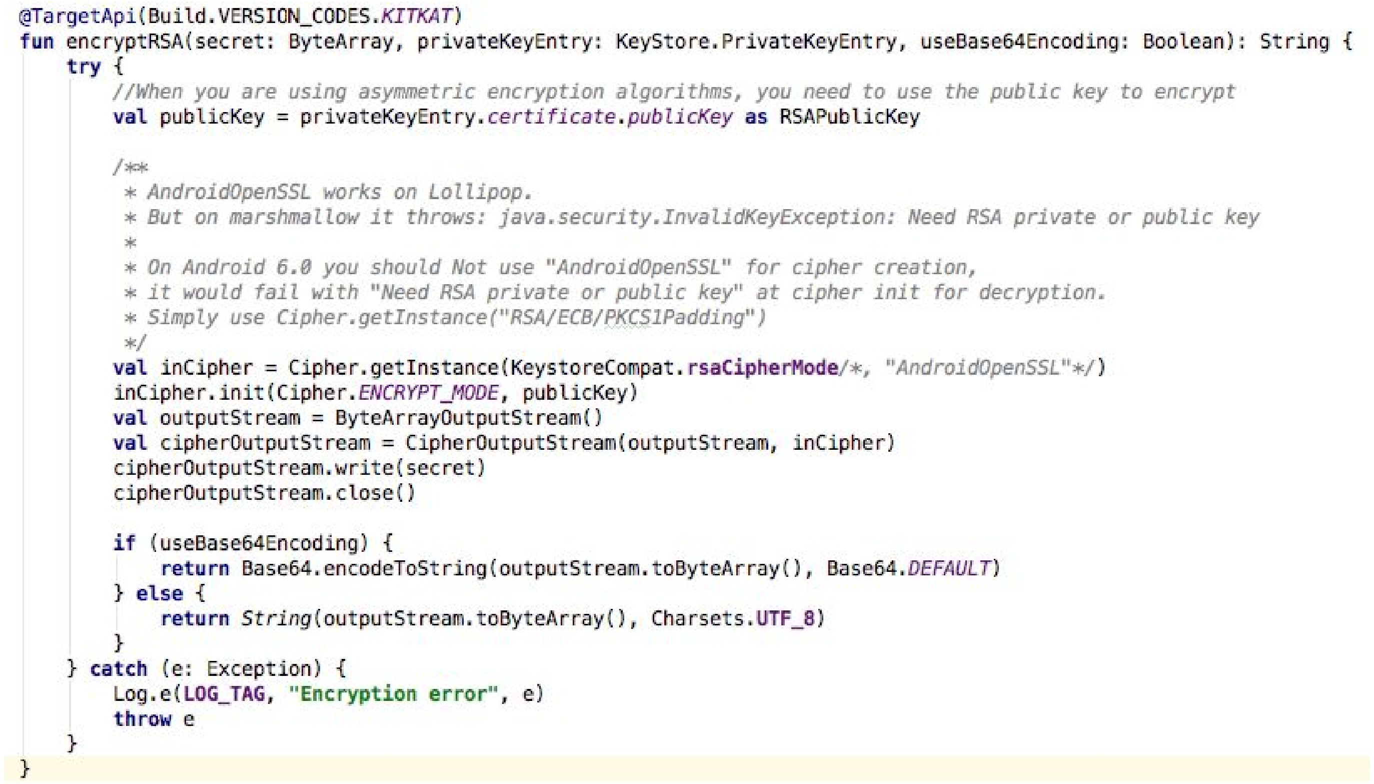


```
internal interface KeystoreCompatFacade {

    fun storeSecret(secret: ByteArray,
                    privateKeyEntry: KeyStore.PrivateKeyEntry,
                    useBase64Encoding: Boolean): String

    fun loadSecret(onSuccess: (cre: ByteArray) -> Unit,
                   onFailure: (e: Exception) -> Unit,
                   clearCredentials: () -> Unit,
                   forceFlag: Boolean?,
                   encryptedUserData: String,
                   privateKeyEntry: KeyStore.PrivateKeyEntry,
                   isBase64Encoded: Boolean)

    fun getAlgorithmParameterSpec(certSubject: X500Principal,
                                  alias: String,
                                  startDate: Date, endDate: Date,
                                  context: Context): AlgorithmParameterSpec

    fun isSecurityEnabled(context: Context): Boolean
}
```

Choose Implementation of **isSecurityEnabled** (3 methods found)

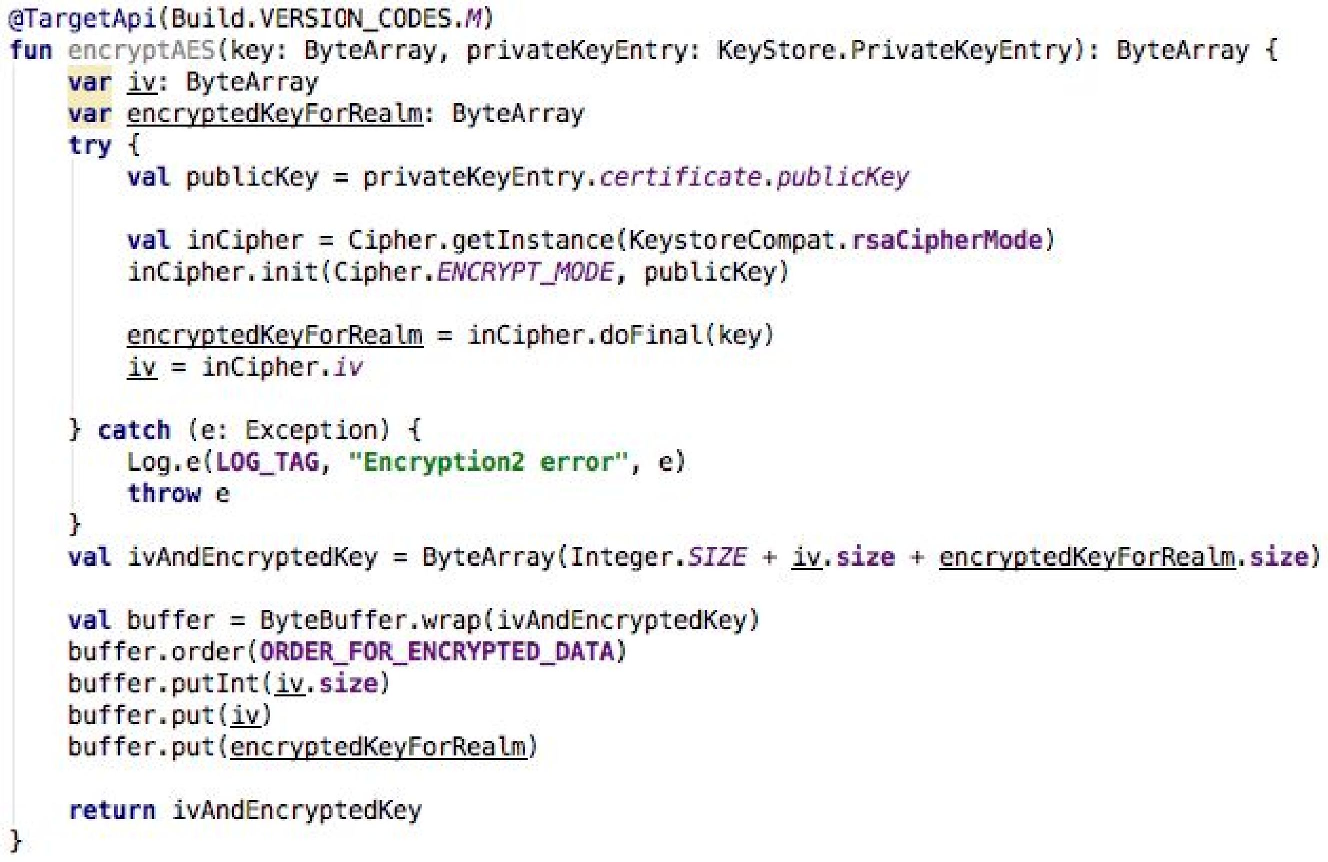
```
KeystoreCompatK (cz.koto.misak.keystorecompat)   android-keystore-compat
KeystoreCompatL (cz.koto.misak.keystorecompat)   android-keystore-compat
KeystoreCompatM (cz.koto.misak.keystorecompat)   android-keystore-compat
```

ANDROID VERSION-SPECIFIC CRYPTO HANDLING 2/3



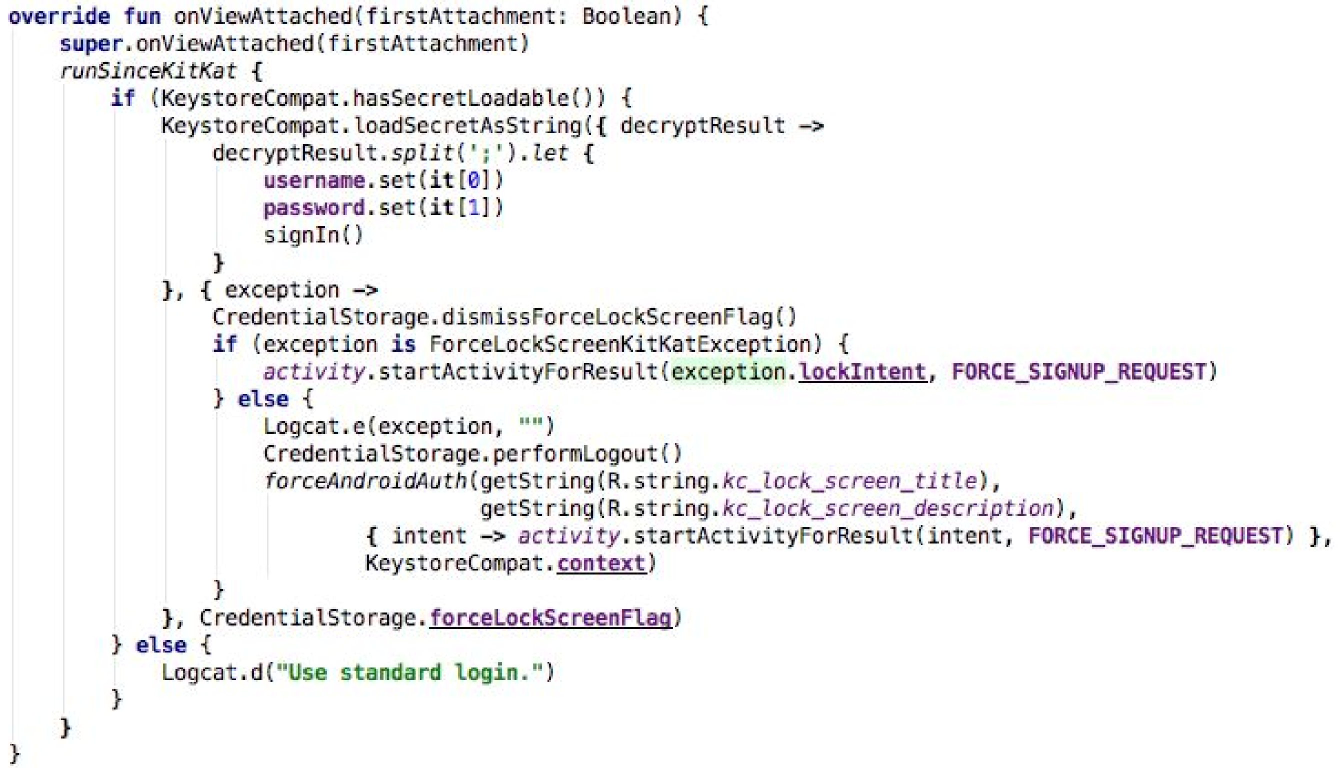
```kotlin
@TargetApi(Build.VERSION_CODES.KITKAT)
fun encryptRSA(secret: ByteArray, privateKeyEntry: KeyStore.PrivateKeyEntry, useBase64Encoding: Boolean): String {
    try {
        //When you are using asymmetric encryption algorithms, you need to use the public key to encrypt
        val publicKey = privateKeyEntry.certificate.publicKey as RSAPublicKey

        /**
         * AndroidOpenSSL works on Lollipop.
         * But on marshmallow it throws: java.security.InvalidKeyException: Need RSA private or public key
         *
         * On Android 6.0 you should Not use "AndroidOpenSSL" for cipher creation,
         * it would fail with "Need RSA private or public key" at cipher init for decryption.
         * Simply use Cipher.getInstance("RSA/ECB/PKCS1Padding")
         */
        val inCipher = Cipher.getInstance(KeystoreCompat.rsaCipherMode/*, "AndroidOpenSSL"*/)
        inCipher.init(Cipher.ENCRYPT_MODE, publicKey)
        val outputStream = ByteArrayOutputStream()
        val cipherOutputStream = CipherOutputStream(outputStream, inCipher)
        cipherOutputStream.write(secret)
        cipherOutputStream.close()

        if (useBase64Encoding) {
            return Base64.encodeToString(outputStream.toByteArray(), Base64.DEFAULT)
        } else {
            return String(outputStream.toByteArray(), Charsets.UTF_8)
        }
    } catch (e: Exception) {
        Log.e(LOG_TAG, "Encryption error", e)
        throw e
    }
}
```

ANDROID VERSION-SPECIFIC CRYPTO HANDLING 3/3



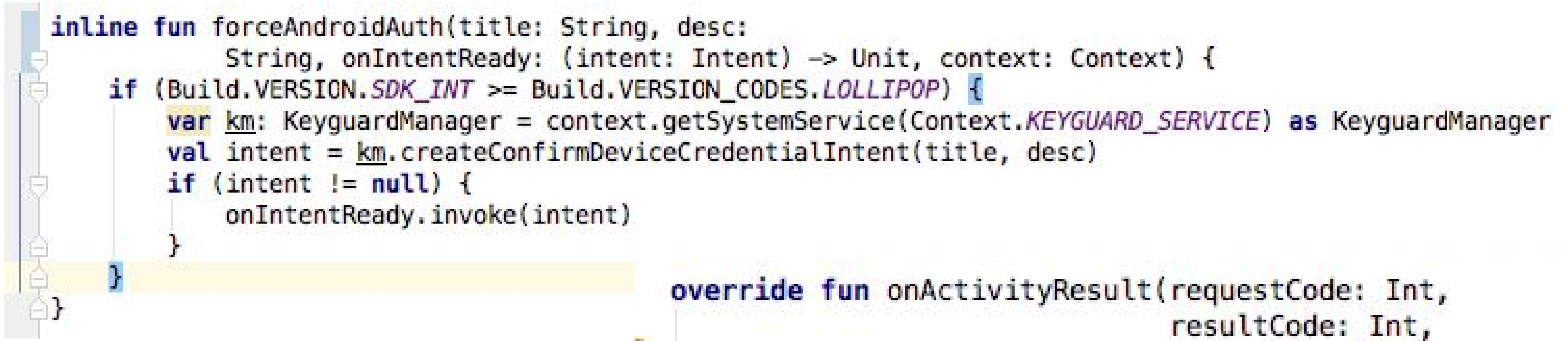
```
@TargetApi(Build.VERSION_CODES.M)
fun encryptAES(key: ByteArray, privateKeyEntry: KeyStore.PrivateKeyEntry): ByteArray {
    var iv: ByteArray
    var encryptedKeyForRealm: ByteArray
    try {
        val publicKey = privateKeyEntry.certificate.publicKey

        val inCipher = Cipher.getInstance(KeystoreCompat.rsaCipherMode)
        inCipher.init(Cipher.ENCRYPT_MODE, publicKey)

        encryptedKeyForRealm = inCipher.doFinal(key)
        iv = inCipher.iv

    } catch (e: Exception) {
        Log.e(LOG_TAG, "Encryption2 error", e)
        throw e
    }
    val ivAndEncryptedKey = ByteArray(Integer.SIZE + iv.size + encryptedKeyForRealm.size)

    val buffer = ByteBuffer.wrap(ivAndEncryptedKey)
    buffer.order(ORDER_FOR_ENCRYPTED_DATA)
    buffer.putInt(iv.size)
    buffer.put(iv)
    buffer.put(encryptedKeyForRealm)

    return ivAndEncryptedKey
}
```

ANDROID VERSION-SPECIFIC LOCK-SCREEN HANDLING 1/2

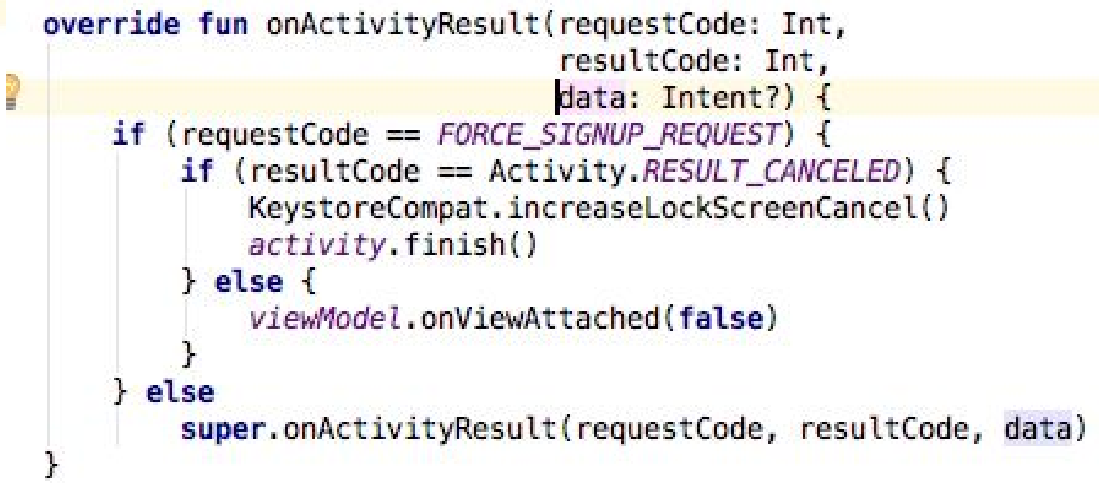


```kotlin
override fun onViewAttached(firstAttachment: Boolean) {
    super.onViewAttached(firstAttachment)
    runSinceKitKat {
        if (KeystoreCompat.hasSecretLoadable()) {
            KeystoreCompat.loadSecretAsString({ decryptResult ->
                decryptResult.split(';').let {
                    username.set(it[0])
                    password.set(it[1])
                    signIn()
                }
            }, { exception ->
                CredentialStorage.dismissForceLockScreenFlag()
                if (exception is ForceLockScreenKitKatException) {
                    activity.startActivityForResult(exception.lockIntent, FORCE_SIGNUP_REQUEST)
                } else {
                    Logcat.e(exception, "")
                    CredentialStorage.performLogout()
                    forceAndroidAuth(getString(R.string.kc_lock_screen_title),
                                     getString(R.string.kc_lock_screen_description),
                            { intent -> activity.startActivityForResult(intent, FORCE_SIGNUP_REQUEST) },
                            KeystoreCompat.context)
                }
            }, CredentialStorage.forceLockScreenFlag)
        } else {
            Logcat.d("Use standard login.")
        }
    }
}
```

ANDROID VERSION-SPECIFIC LOCK-SCREEN HANDLING 2/2

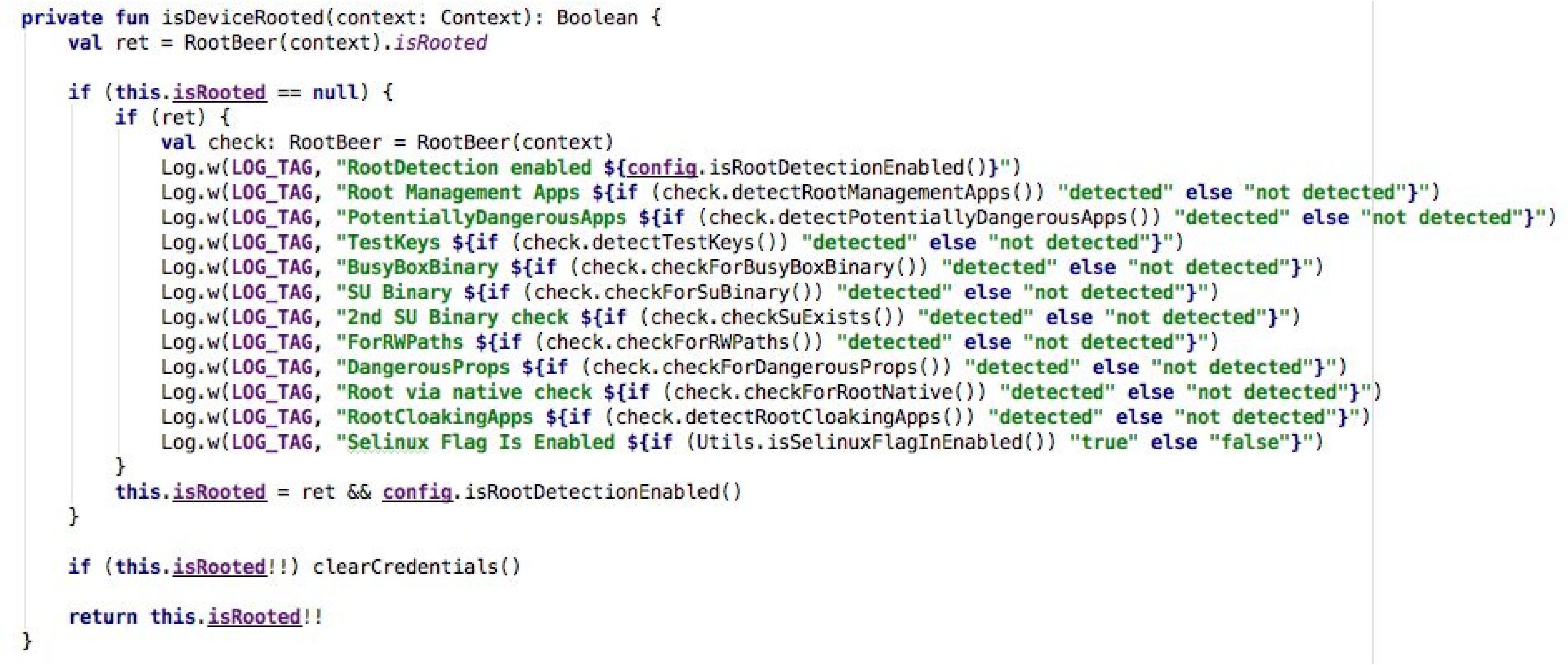


```kotlin
inline fun forceAndroidAuth(title: String, desc:
            String, onIntentReady: (intent: Intent) -> Unit, context: Context) {
    if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.LOLLIPOP) {
        var km: KeyguardManager = context.getSystemService(Context.KEYGUARD_SERVICE) as KeyguardManager
        val intent = km.createConfirmDeviceCredentialIntent(title, desc)
        if (intent != null) {
            onIntentReady.invoke(intent)
        }
    }
}
```



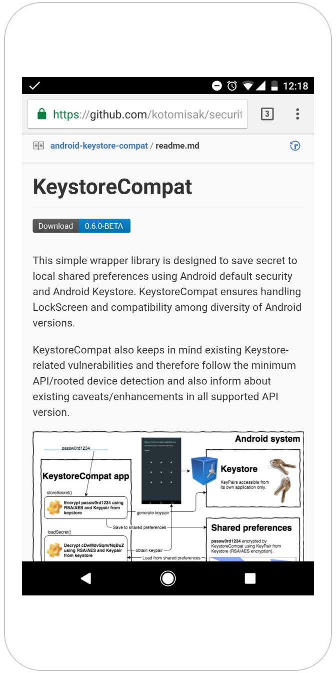
```kotlin
override fun onActivityResult(requestCode: Int,
                              resultCode: Int,
                              data: Intent?) {
    if (requestCode == FORCE_SIGNUP_REQUEST) {
        if (resultCode == Activity.RESULT_CANCELED) {
            KeystoreCompat.increaseLockScreenCancel()
            activity.finish()
        } else {
            viewModel.onViewAttached(false)
        }
    } else
        super.onActivityResult(requestCode, resultCode, data)
}
```

ROOT DETECTION HANDLING



```kotlin
private fun isDeviceRooted(context: Context): Boolean {
    val ret = RootBeer(context).isRooted

    if (this.isRooted == null) {
        if (ret) {
            val check: RootBeer = RootBeer(context)
            Log.w(LOG_TAG, "RootDetection enabled ${config.isRootDetectionEnabled()}")
            Log.w(LOG_TAG, "Root Management Apps ${if (check.detectRootManagementApps()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "PotentiallyDangerousApps ${if (check.detectPotentiallyDangerousApps()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "TestKeys ${if (check.detectTestKeys()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "BusyBoxBinary ${if (check.checkForBusyBoxBinary()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "SU Binary ${if (check.checkForSuBinary()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "2nd SU Binary check ${if (check.checkSuExists()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "ForRWPaths ${if (check.checkForRWPaths()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "DangerousProps ${if (check.checkForDangerousProps()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "Root via native check ${if (check.checkForRootNative()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "RootCloakingApps ${if (check.detectRootCloakingApps()) "detected" else "not detected"}")
            Log.w(LOG_TAG, "Selinux Flag Is Enabled ${if (Utils.isSelinuxFlagInEnabled()) "true" else "false"}")
        }
        this.isRooted = ret && config.isRootDetectionEnabled()
    }

    if (this.isRooted!!) clearCredentials()

    return this.isRooted!!
}
```

KeystoreCompat

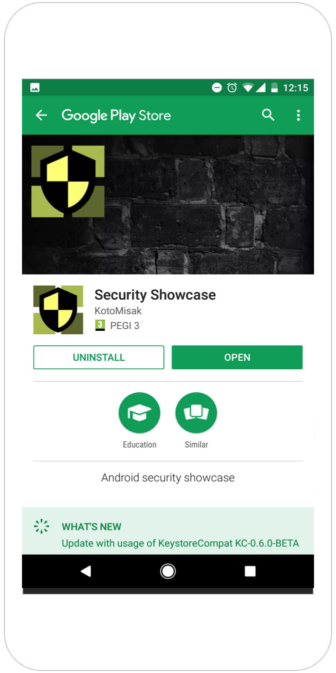

12:18
https://github.com/kotomisak/securit
android-keystore-compat / readme.md

KeystoreCompat

This simple wrapper library is designed to save secret to local shared preferences using Android default security and Android Keystore. KeystoreCompat ensures handling LockScreen and compatibility among diversity of Android versions.

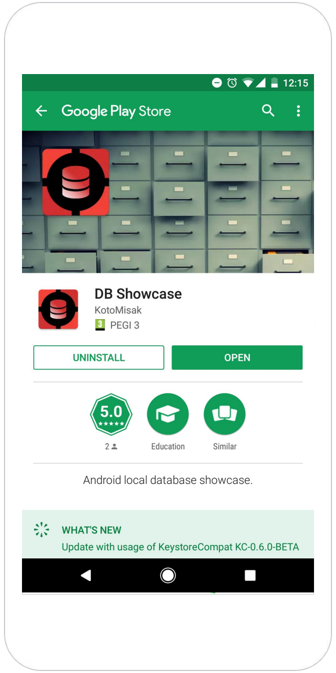
KeystoreCompat also keeps in mind existing Keystore-related vulnerabilities and therefore follow the minimum API/rooted device detection and also inform about existing caveats/enhancements in all supported API version.

Android system
passw0rd1234
KeystoreCompat app
storeSecret()
Encrypt passw0rd1234 using RSA/AES and Keypair from keystore
loadSecret()
Decrypt cDwMdvSqmrNqBuZ using RSA/AES and Keypair from keystore
generate keypair
Save to shared preferences
obtain keypair
Load from shared preferences
Keystore
KeyPairs accessible from its own application only.
Shared preferences
passw0rd1234 encrypted by KeystoreCompat using KeyPair from Keystore (RSA/AES encryption).


12:15
Google Play Store
Security Showcase
KotoMisak
PEGI 3
UNINSTALL
OPEN
Education
Similar

Android security showcase

WHAT'S NEW
Update with usage of KeystoreCompat KC-0.6.0-BETA



Android local database showcase.

WHAT'S NEW
Update with usage of KeystoreCompat KC-0.6.0-BETA

https://github.com/kotomisak/security-showcase-android/blob/develop/android-keystore-compat/readme.md

THANK YOU

michal.jenicek@strv.com

STRV

QUESTIONS

STRV