

# Buffed-Up Bionmetrics

Brian Gardner  
Big Nerd Ranch

# What are Bionmetrics?

# Types of Biometrics<sup>1</sup>

- Fingerprint
- Face
- Iris
- DNA
- Gait
- Typing rythm

<sup>1</sup> Examples from <https://en.wikipedia.org/wiki/Biometrics>

# Security<sup>2</sup>



Knowledge



Possession



Biometric

<sup>2</sup> Image from Better Biometrics in Android P

# New Threats

DeepMasterPrints: Generating MasterPrints for Dictionary Attacks via Latent Variable Evolution<sup>3</sup>

Train Generative Adversarial Network with real fingerprints

<sup>3</sup> Research paper available here: <https://arxiv.org/pdf/1705.07386.pdf>

# Old News

FingerprintManager

FingerprintManagerCompat

# Why introduce something new?

FingerprintManager only works with fingerprint authentication



# BiometricPrompt!



# Permissions

```
<uses-permission android:name="android.permission.USE_BIOMETRIC" />
```

# BiometricPrompt Object

```
private fun showAuthenticationDialog() {  
    val title = getString(R.string.authentication_title)  
    val cancel = getString(R.string.cancel_button)  
    val biometricPrompt = BiometricPrompt.Builder(this)  
        .setTitle(title)  
        .setNegativeButton(cancel, mainExecutor, negativeListener)  
        .build()  
}
```

# BiometricPrompt Object

```
private fun showAuthenticationDialog() {  
    val title = getString(R.string.authentication_title)  
    val cancel = getString(R.string.cancel_button)  
    val biometricPrompt = BiometricPrompt.Builder(this)  
        .setTitle(title)  
        .setNegativeButton(cancel, mainExecutor, negativeListener)  
        .build()  
}
```

# BiometricPrompt Object

```
private fun showAuthenticationDialog() {  
    val title = getString(R.string.authentication_title)  
    val cancel = getString(R.string.cancel_button)  
    val biometricPrompt = BiometricPrompt.Builder(this)  
        .setTitle(title)  
        .setNegativeButton(cancel, mainExecutor, negativeListener)  
        .build()  
}
```

# BiometricPrompt Object

```
private fun showAuthenticationDialog() {  
    val title = getString(R.string.authentication_title)  
    val cancel = getString(R.string.cancel_button)  
    val biometricPrompt = BiometricPrompt.Builder(this)  
        .setTitle(title)  
        .setNegativeButton(cancel, mainExecutor, negativeListener)  
        .build()  
}
```

# Cancel Listener

```
private val negativeListener =  
    DialogInterface.OnClickListener { _, _ ->  
        // Biometric auth canceled, authenticate a different way  
    }
```

# Trigger Authentication

```
// Regular authentication
biometricPrompt.authenticate(
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
// Unlock crypto object
biometricPrompt.authenticate(
    cryptoObject
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
```

# Trigger Authentication

```
// Regular authentication
biometricPrompt.authenticate(
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
// Unlock crypto object
biometricPrompt.authenticate(
    cryptoObject
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
```



# Trigger Authentication

```
// Regular authentication
biometricPrompt.authenticate(
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
// Unlock crypto object
biometricPrompt.authenticate(
    cryptoObject
    cancellationSignal,
    mainExecutor,
    authenticationCallback
)
```

# Cancellation signal

```
val cancellationSignal = CancellationSignal()  
  
// To cancel later  
cancellationSignal.cancel()
```

# CryptoObject

```
val signatureObject = BiometricPrompt.CryptoObject(signature)
```

```
val cipherObject = BiometricPrompt.CryptoObject(cipher)
```

```
val macObject = BiometricPrompt.CryptoObject(mac)
```

# Crypto Object

```
new KeyGenParameterSpec.Builder(  
    "key1", KeyProperties.PURPOSE_SIGN)  
    // Only permit the private key to be used if the user authenticated  
    .setUserAuthenticationRequired(true)  
    .build()
```

# Crypto Object

```
new KeyGenParameterSpec.Builder(  
    "key1", KeyProperties.PURPOSE_SIGN)  
    // Only permit the private key to be used if the user authenticated  
    .setUserAuthenticationRequired(true)  
    .build()
```

# Callbacks

```
private val authenticationCallback = object: AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: AuthenticationResult?) {  
    }  
  
    override fun onAuthenticationError(errorCode: Int,  
                                       errString: CharSequence?) {  
    }  
  
    override fun onAuthenticationFailed() {  
    }  
  
    override fun onAuthenticationHelp(helpCode: Int,  
                                       helpString: CharSequence?) {  
    }  
}
```

# Callbacks

```
private val authenticationCallback = object: AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: AuthenticationResult?) {  
    }  
  
    override fun onAuthenticationError(errorCode: Int,  
                                       errString: CharSequence?) {  
    }  
  
    override fun onAuthenticationFailed() {  
    }  
  
    override fun onAuthenticationHelp(helpCode: Int,  
                                       helpString: CharSequence?) {  
    }  
}
```

# Callbacks

```
private val authenticationCallback = object: AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: AuthenticationResult?) {  
    }  
  
    override fun onAuthenticationError(errorCode: Int,  
                                       errString: CharSequence?) {  
    }  
  
    override fun onAuthenticationFailed() {  
    }  
  
    override fun onAuthenticationHelp(helpCode: Int,  
                                       helpString: CharSequence?) {  
    }  
}
```



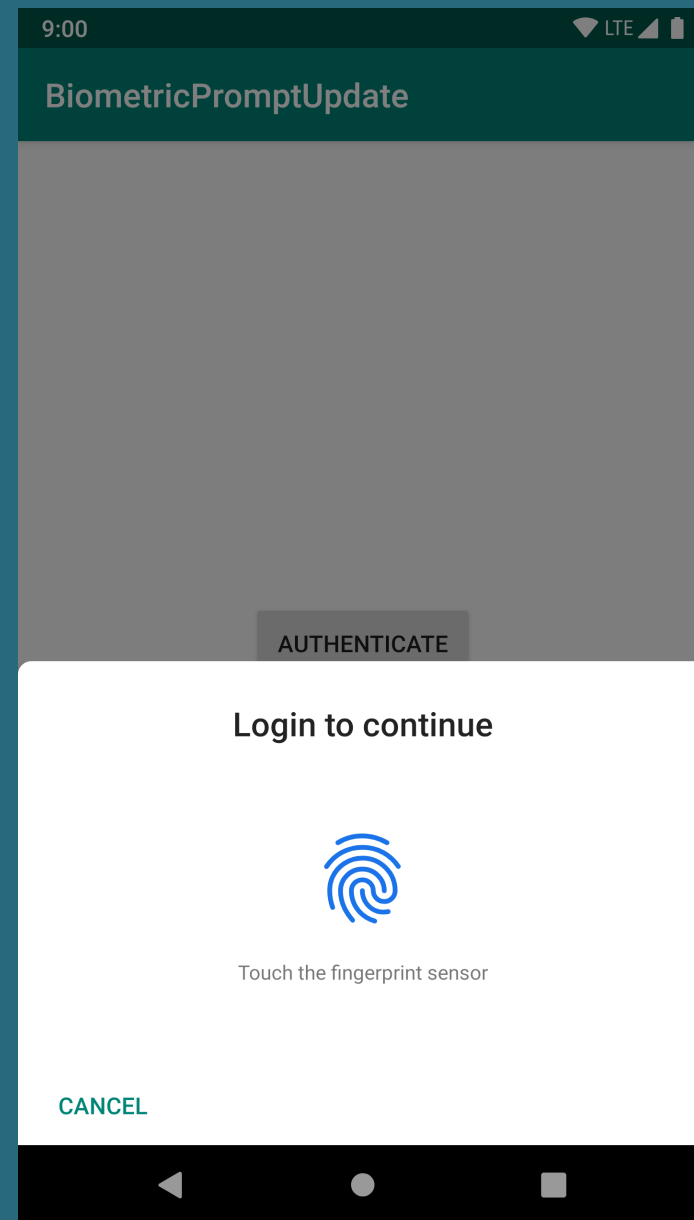
# Callbacks

```
private val authenticationCallback = object: AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: AuthenticationResult?) {  
    }  
  
    override fun onAuthenticationError(errorCode: Int,  
                                       errString: CharSequence?) {  
    }  
  
    override fun onAuthenticationFailed() {  
    }  
  
    override fun onAuthenticationHelp(helpCode: Int,  
                                       helpString: CharSequence?) {  
    }  
}
```

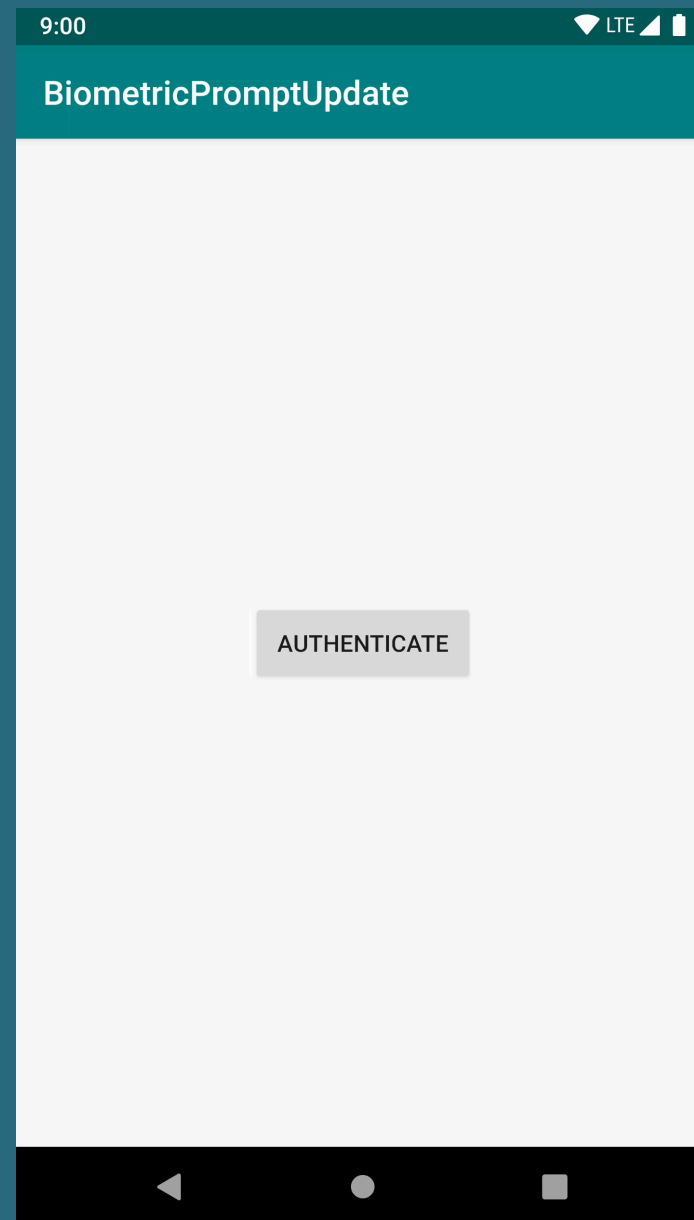
# Callbacks

```
private val authenticationCallback = object: AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: AuthenticationResult?) {  
    }  
  
    override fun onAuthenticationError(errorCode: Int,  
                                       errString: CharSequence?) {  
    }  
  
    override fun onAuthenticationFailed() {  
    }  
  
    override fun onAuthenticationHelp(helpCode: Int,  
                                       helpString: CharSequence?) {  
    }  
}
```

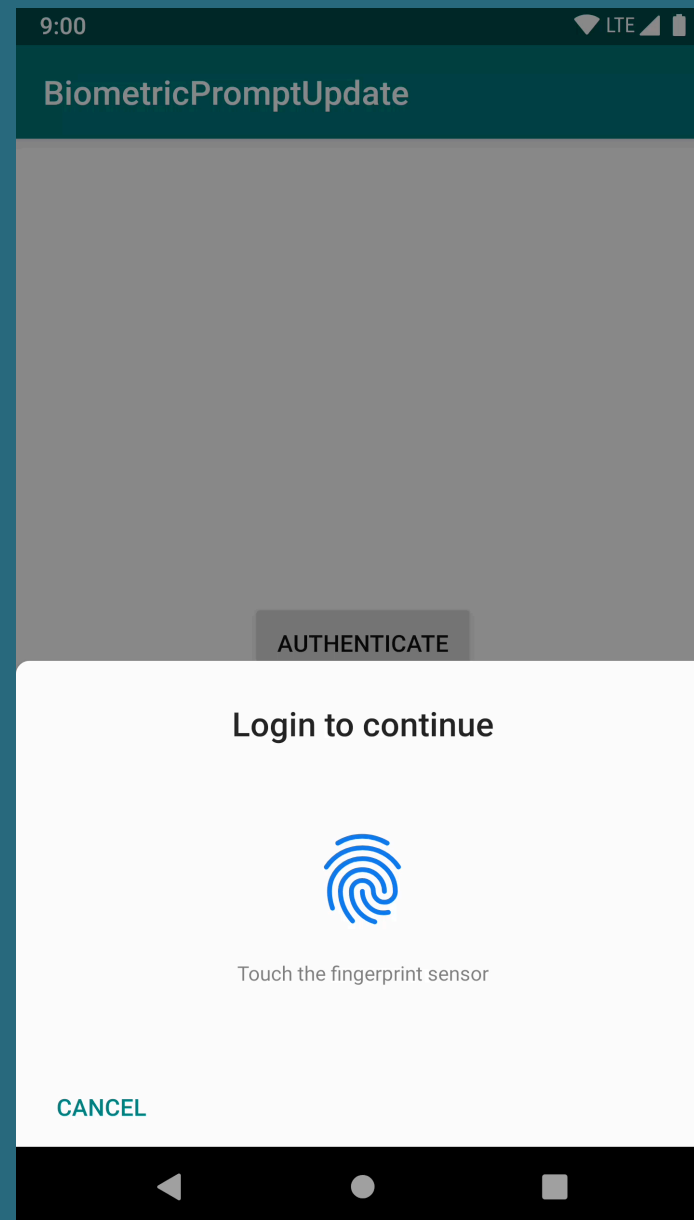
# Default Prompt



# Sad Prompt



# More Sad Prompts



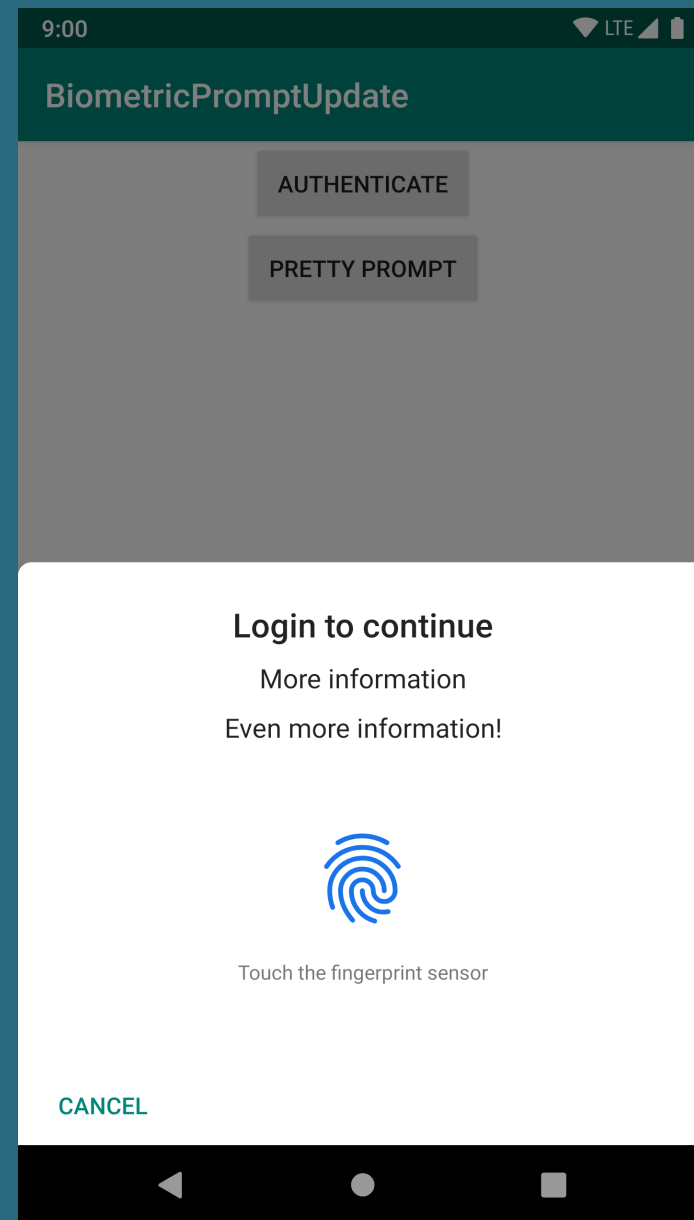
# Prompt configuration

```
val title = getString(R.string.authentication_title)
val cancel = getString(R.string.cancel_button)
val subtitle = getString(R.string.authentication_subtitle)
val description = getString(R.string.authentication_description)
val biometricPrompt = BiometricPrompt.Builder(this)
    .setTitle(title)
    .setNegativeButton(cancel, mainExecutor, negativeListener)
    .setSubtitle(subtitle)
    .setDescription(description)
    .build()
```

# Prompt configuration

```
val title = getString(R.string.authentication_title)
val cancel = getString(R.string.cancel_button)
val subtitle = getString(R.string.authentication_subtitle)
val description = getString(R.string.authentication_description)
val biometricPrompt = BiometricPrompt.Builder(this)
    .setTitle(title)
    .setNegativeButton(cancel, mainExecutor, negativeListener)
    .setSubtitle(subtitle)
    .setDescription(description)
    .build()
```

# Pretty Prompt





# Problem

BiometricPrompt API is available on API 28+

# AndroidX Compatibility 🎉

androidx.biometric:biometric:1.0.0-alpha02

# BiometricPrompt

```
val biometricPrompt = BiometricPrompt(  
    this,  
    executor,  
    authenticationCallback)
```

# Executor

```
val executor = if (Build.VERSION.SDK_INT >= 28) {  
    mainExecutor  
} else {  
    Executor {  
        fun execute(runnable: Runnable) {  
            runnable.run()  
        }  
    }  
}
```

# Executor

```
val executor = if (Build.VERSION.SDK_INT >= 28) {  
    mainExecutor  
} else {  
    Executor {  
        fun execute(runnable: Runnable) {  
            runnable.run()  
        }  
    }  
}
```

# Executor

```
val executor = if (Build.VERSION.SDK_INT >= 28) {  
    mainExecutor  
} else {  
    Executor {  
        fun execute(runnable: Runnable) {  
            runnable.run()  
        }  
    }  
}
```

# AuthenticationCallback

```
private val authenticationCallback = object : BiometricPrompt.AuthenticationCallback() {  
    override fun onAuthenticationSucceeded(result: BiometricPrompt.AuthenticationResult) {  
        Log.d(TAG, "onAuthenticationSucceeded")  
    }  
  
    override fun onAuthenticationError(errorCode: Int, errString: CharSequence) {  
        cancellationSignal.cancel()  
        Log.d(TAG, "onAuthenticationError $errString")  
    }  
  
    override fun onAuthenticationFailed() {  
        Log.d(TAG, "onAuthenticationFailed")  
    }  
}
```

# Prompt Configuration

```
val title = getString(R.string.authentication_title)
val cancel = getString(R.string.cancel_button)
// Configure data for the prompt
val promptInfo = BiometricPrompt.PromptInfo.Builder()
    .setTitle(title)
    .setNegativeButtonText(cancel)
    .build()
```



# Prompt Configuration

```
val title = getString(R.string.authentication_title)
val cancel = getString(R.string.cancel_button)
// Configure data for the prompt
val promptInfo = BiometricPrompt.PromptInfo.Builder()
    .setTitle(title)
    .setNegativeButtonText(cancel)
    .build()
```

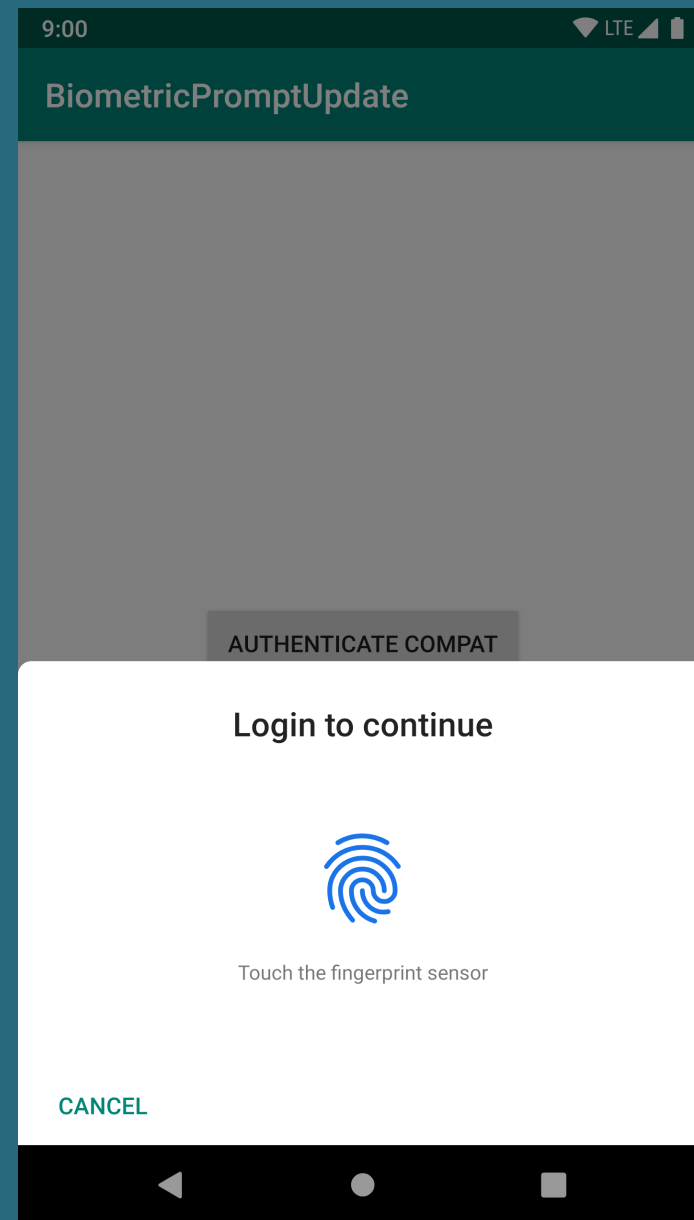
# Trigger Authentication

```
biometricPrompt.authenticate(promptInfo)
```

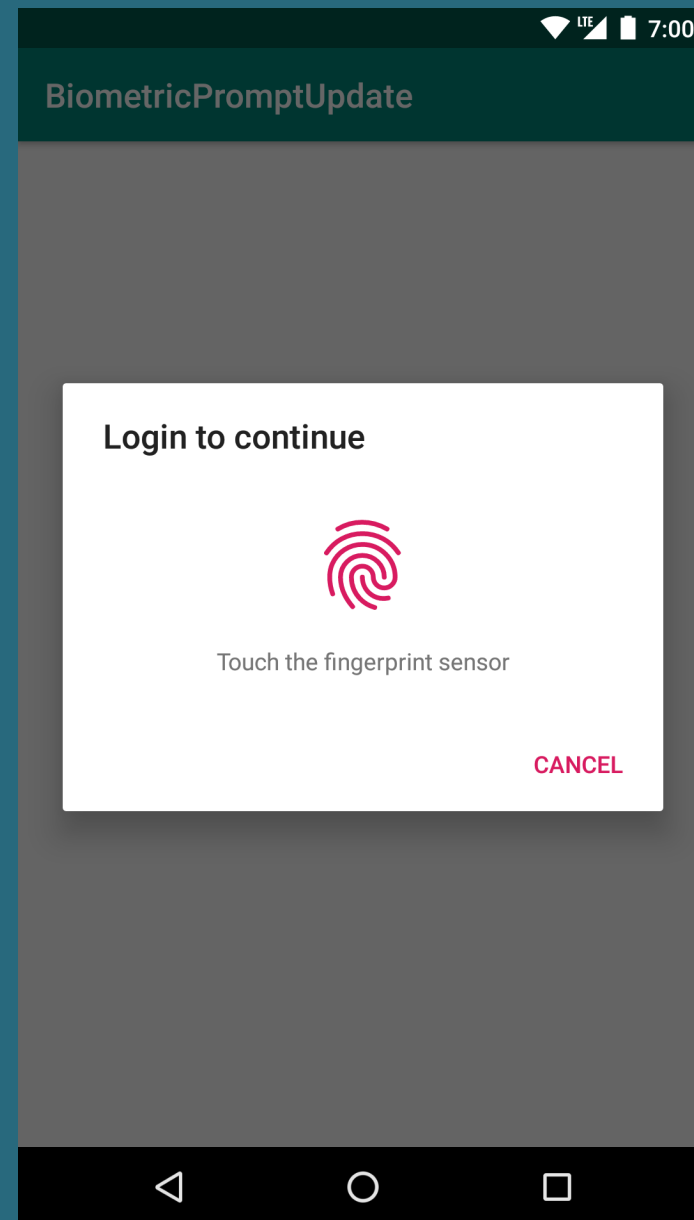
```
// or
```

```
biometricPrompt.authenticate(promptInfo, cryptoObject)
```

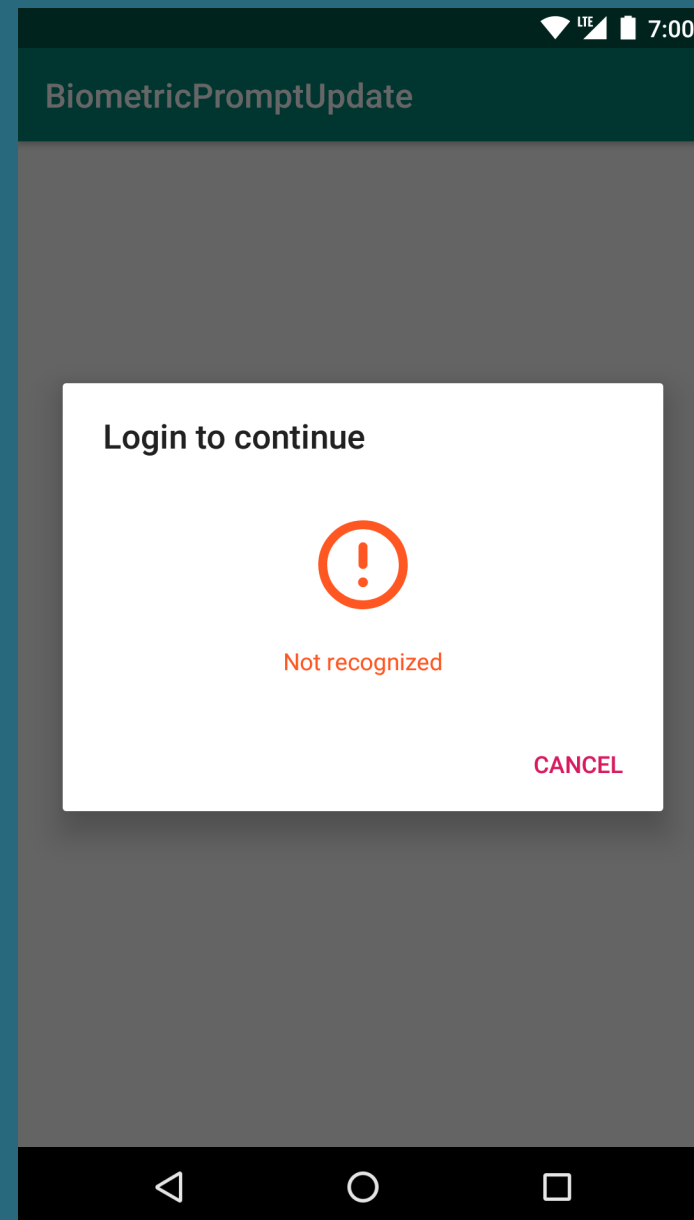
# Compat Prompt (API 28)



# (Old) Compat Prompt



# (Old) Compat Error



# What about the CancellationSignal?

```
biometricPrompt.cancelAuthentication()
```

# Authentication Flow

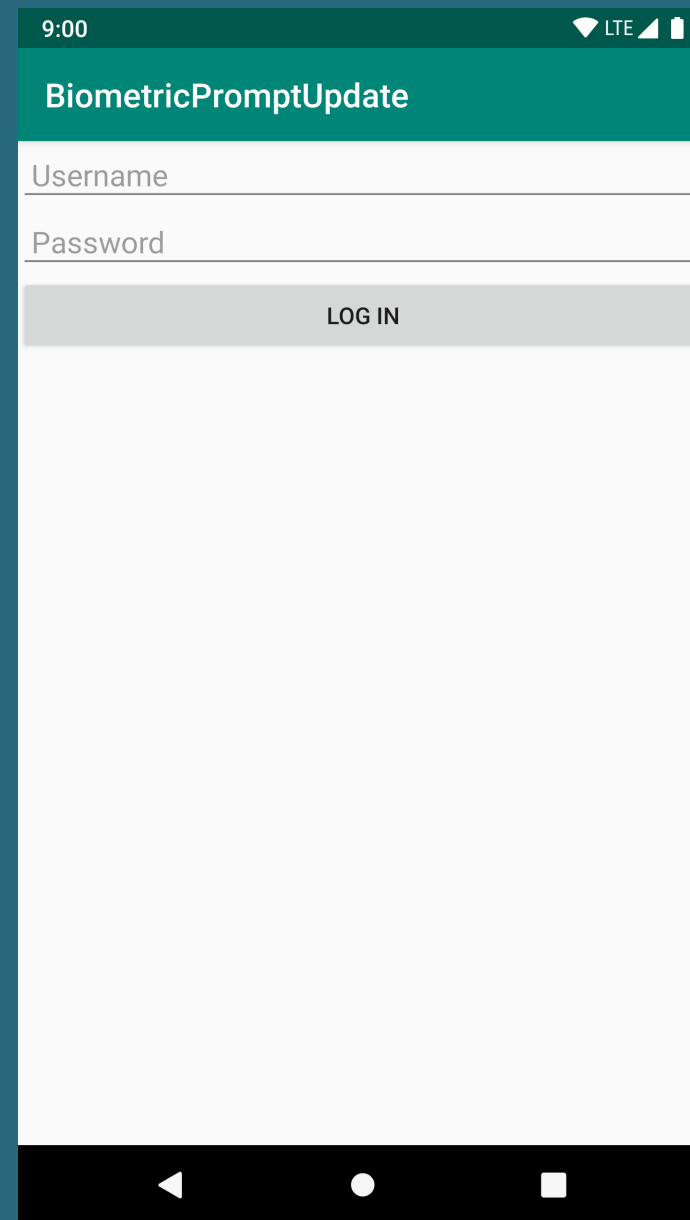
# Is user already using biometrics?

If no -> First-time flow

If yes -> Main flow



# First-time flow

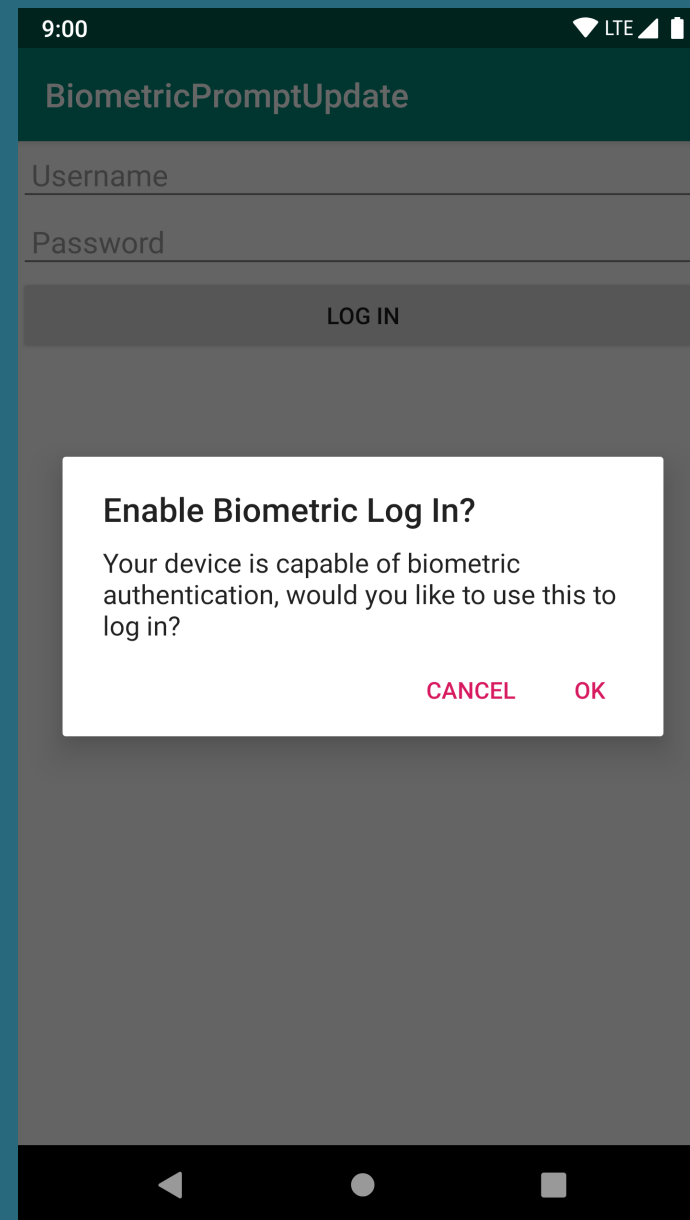


A mobile application login screen mockup. At the top, a dark green status bar shows the time 9:00, LTE signal, and battery level. Below this is a teal header bar with the text "BiometricPromptUpdate". The main content area is white and contains two text input fields: "Username" and "Password". Below the password field is a grey button labeled "LOG IN". The bottom of the screen features a black navigation bar with three white icons: a back arrow, a home circle, and a recent apps square.

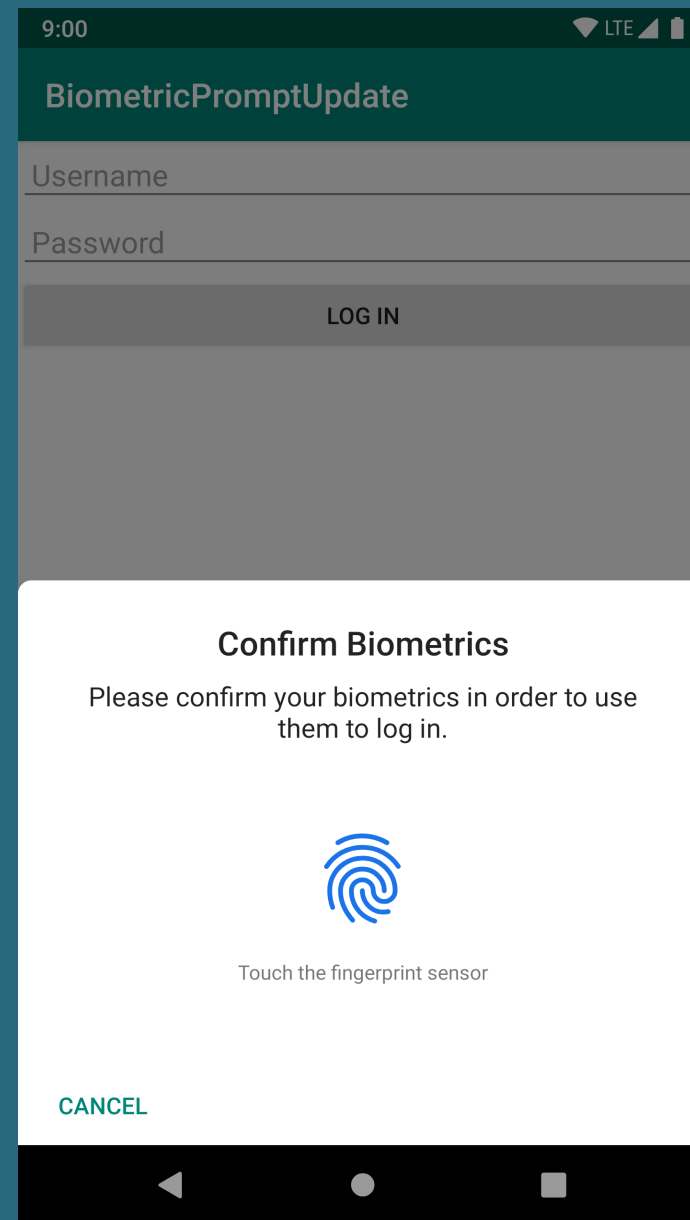
# Check device capability

```
private fun hasBiometrics(): Boolean {  
    return if (Build.VERSION.SDK_INT >= 23) {  
        packageManager.hasSystemFeature(  
            PackageManager.FEATURE_FINGERPRINT  
        )  
    } else {  
        false  
    }  
}
```

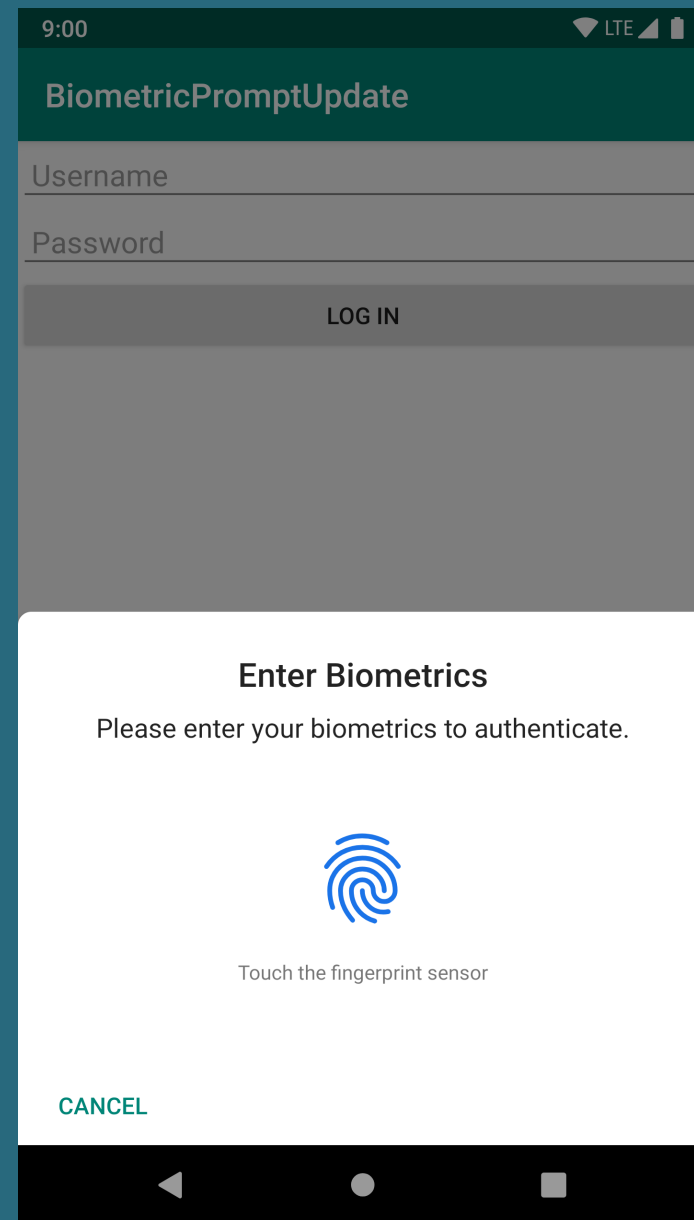
# First-time flow



# First-time flow



# Main flow



# Main flow

9:00 LTE

## BiometricPromptUpdate

Username

Password

LOG IN

Unrecognized biometrics, please log in with username and password

# Future plans

# Resources

## Biometrics

### Better Biometrics in Android P

### Measuring Biometric Unlock Security



# About Me

Brian Gardner

@BrianGardnerAtl

Big Nerd Ranch

Sample project & slides:  
[bit.ly/dcsf-bio-prompt](https://bit.ly/dcsf-bio-prompt)