

# Lecture #7

# Securing Mobile Apps

Mobile Applications  
Fall 2023

# Android Security Strategy

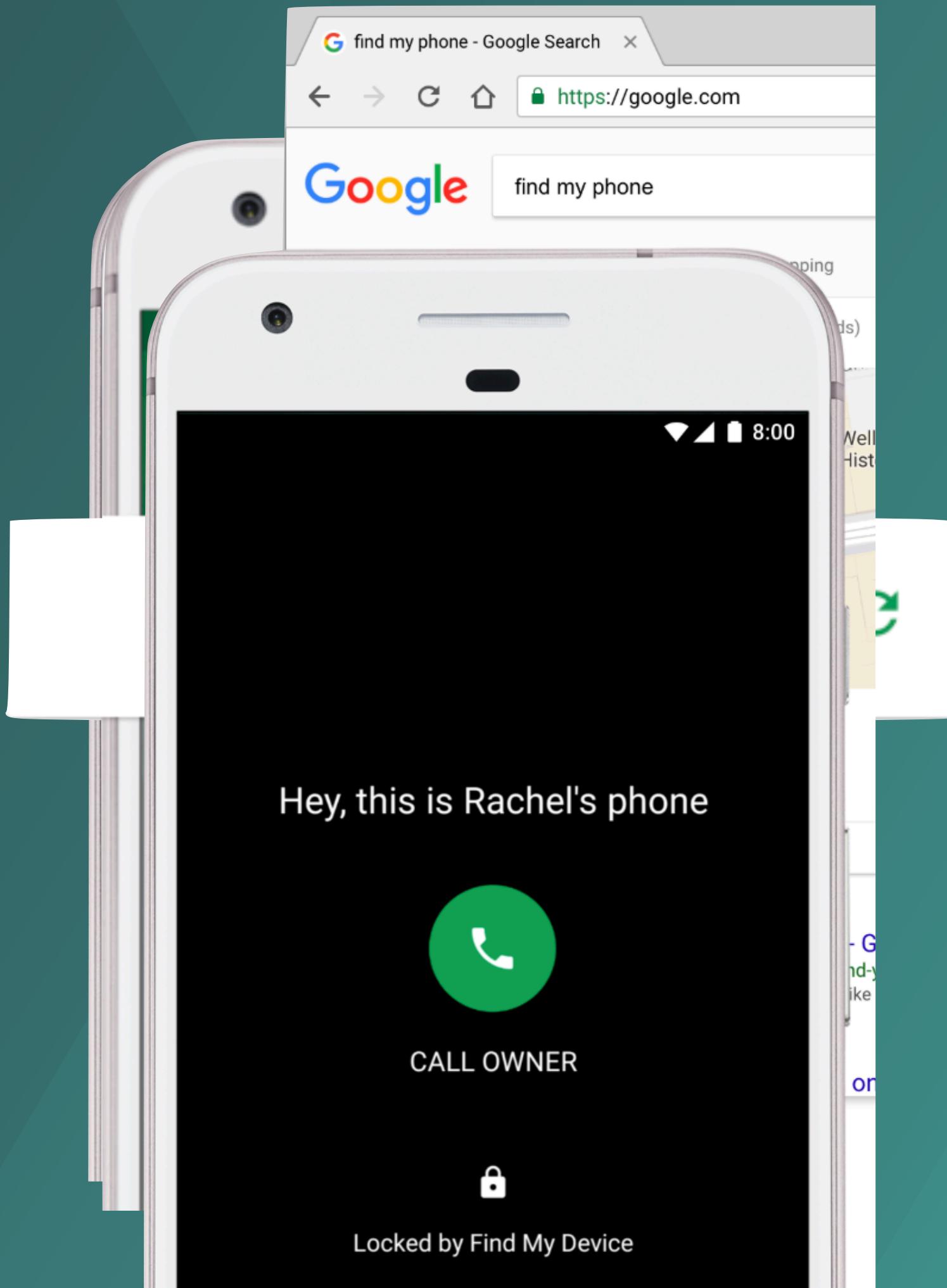
- Google Play Protect
  - Defend against Internet-borne threats.
  - User experience that offers security CCC (comprehension, control, confidence).
- Platform Engineering
  - Feature dev.
  - OS hardening, leverage HW.
- SDLC - Security Development Life Cycle
  - Vulnerability management.
  - Full cycle.



# Google Play Protect



- Keeping your device safe, 24/7
- Scanning and verifying over 100 billion apps every day
- Securing your device, even if it's lost
- Helping you surf on the safe side



<https://www.android.com/play-protect/>

# Platform Engineering

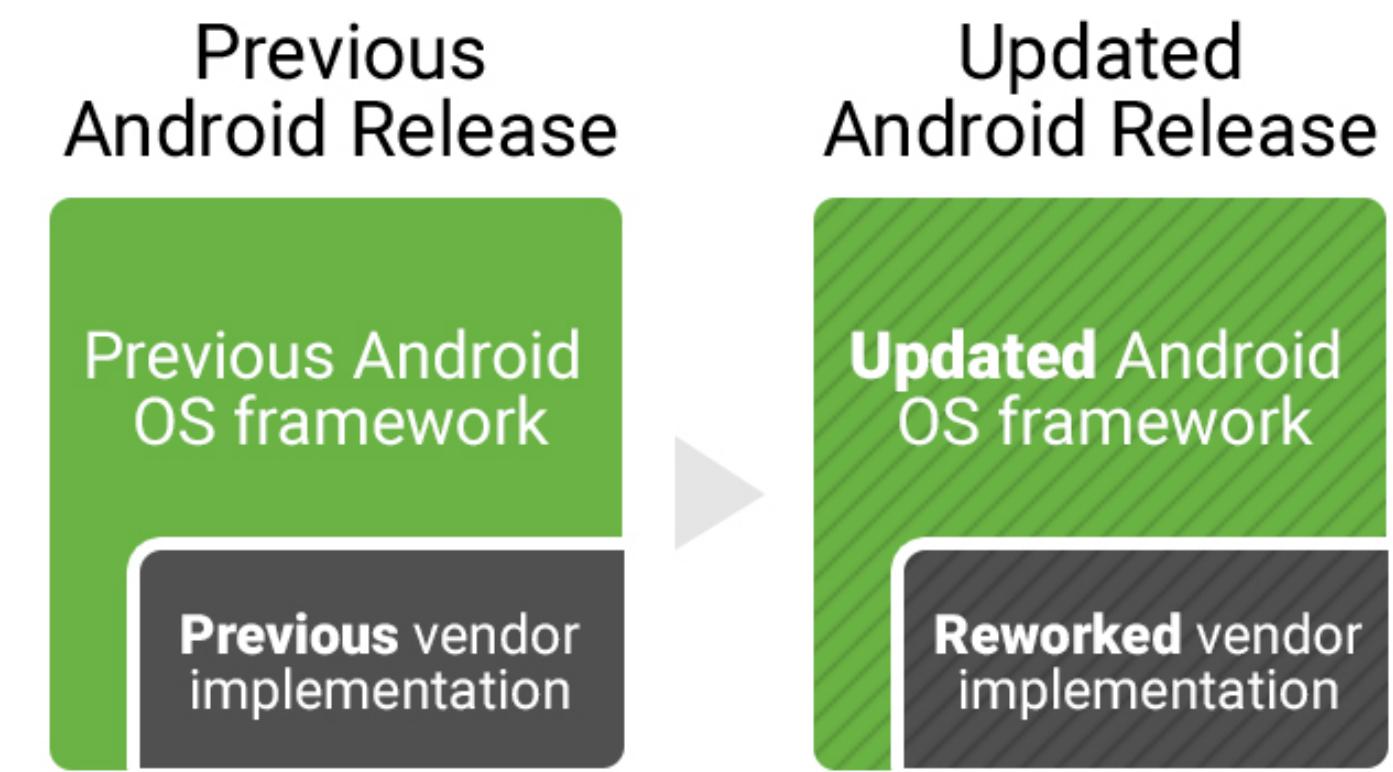
- SELinux
  - Allows users and administrators more control over access control.
  - Access can be constrained, as which users and applications can access which resources.
  - Adds finer granularity to access controls.
- Control Flow Integrity
  - Protecting against code reuse attacks.
  - Implementing in the Linux kernel.
- Verify Boot
  - Ensure all executed code comes from a trusted source.



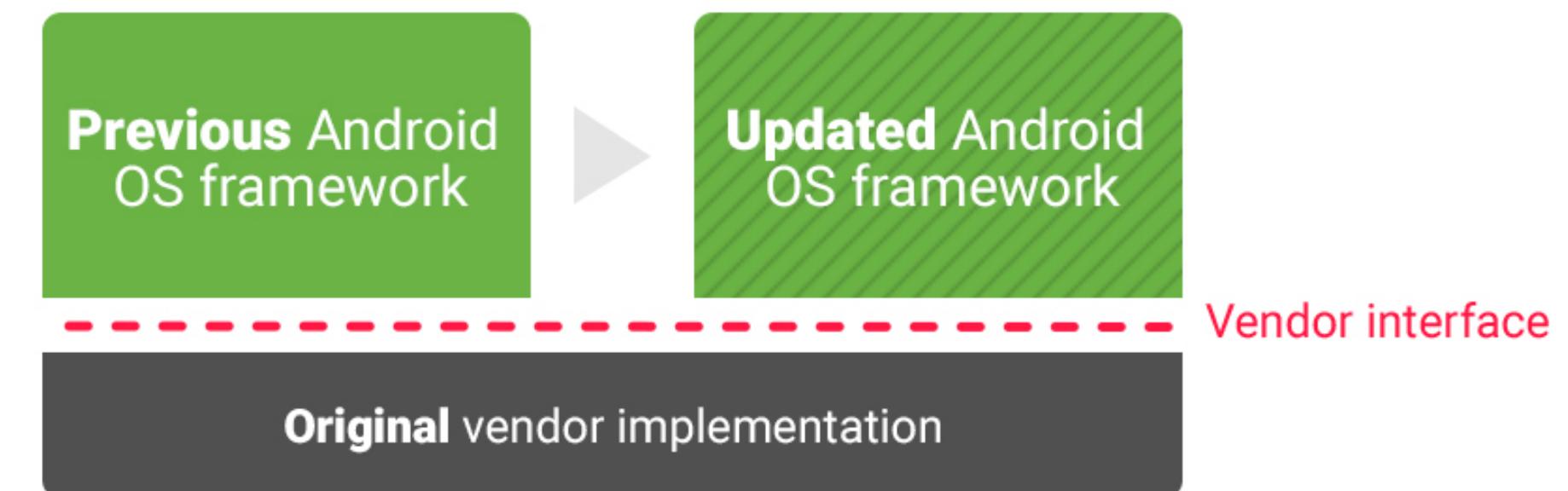
# Security Development Lifecycle

- Testing infrastructure.
- Security patching program.
- HAL interface definition language (HIDL)
- Treble: A modular base for Android

## Before Treble



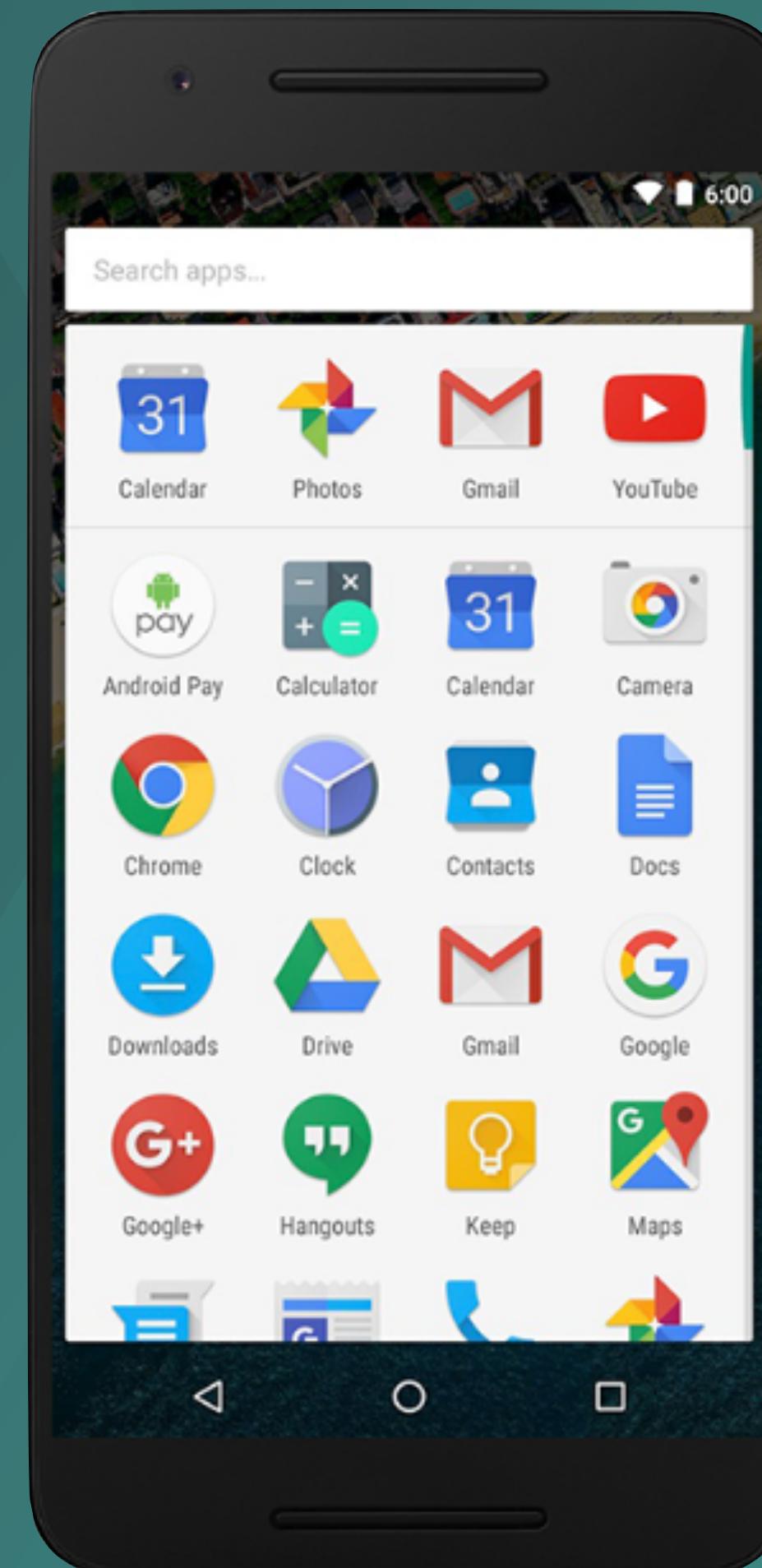
## With Treble



# Security for Android Developers



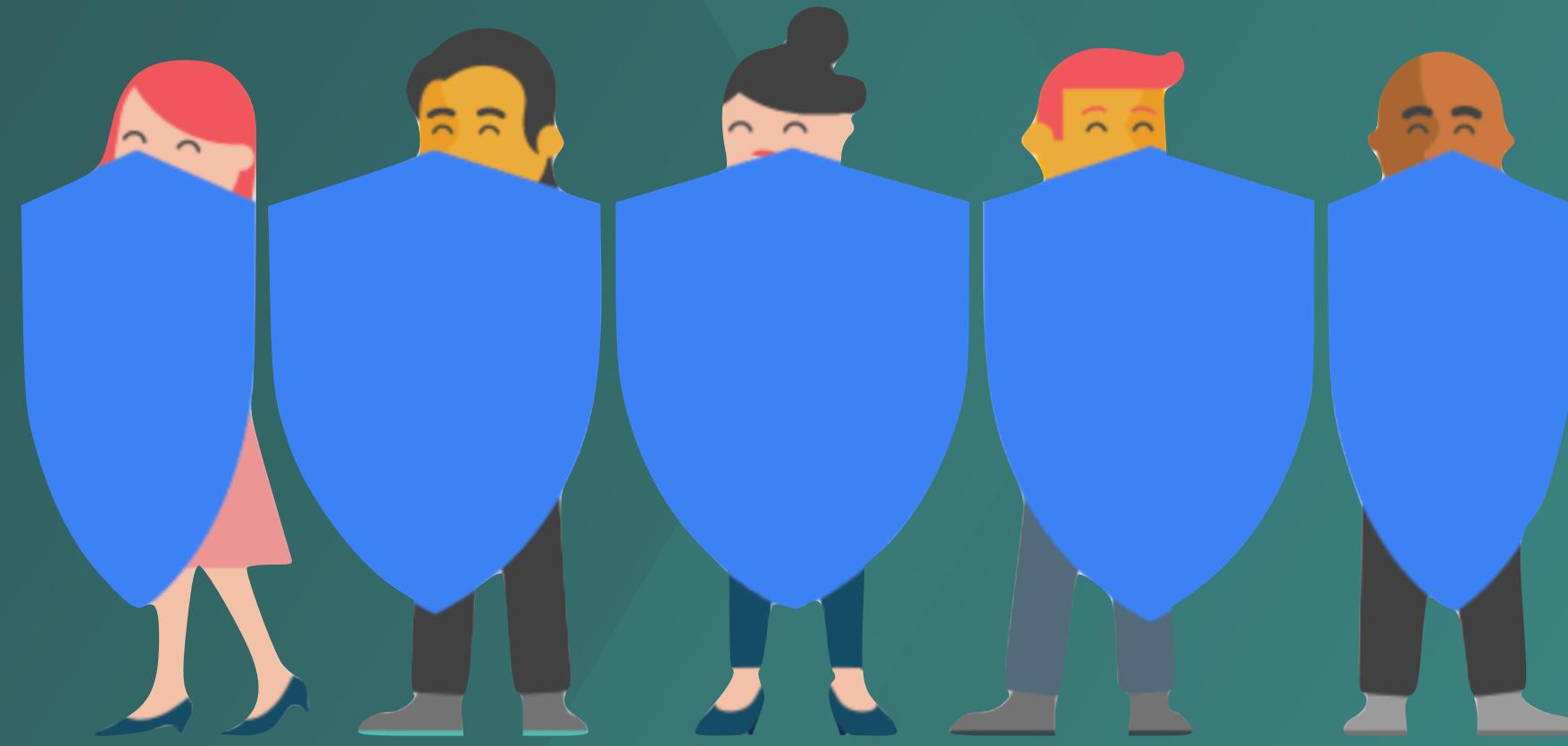
- Store data safely.
- Enforce secure communication.
- Update security provider.
- Pay attention to permissions.



# Store Data Safely



- Minimize the use of APIs that access sensitive or personal user data.
- Consider using hash or non-reversible form of the data to represent the user's sensitive details.



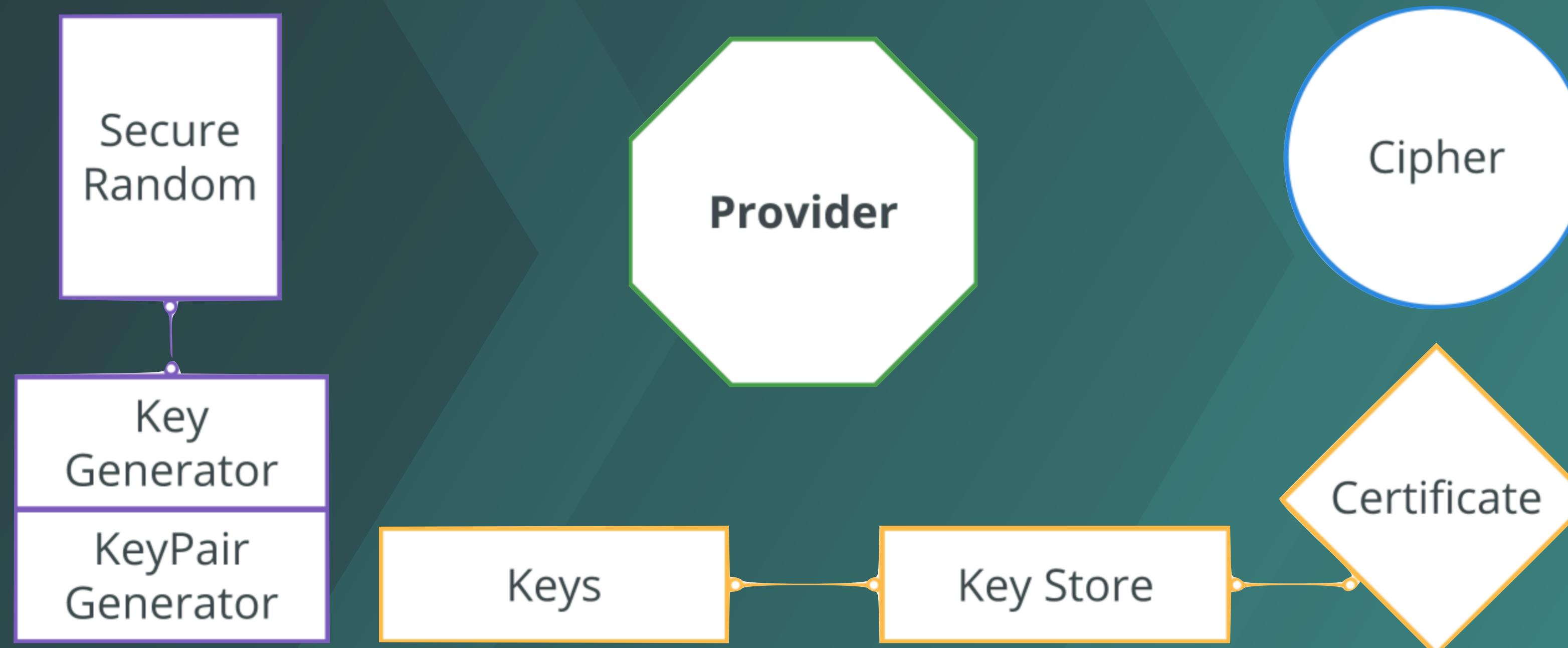
# Store Data Safely



- ✓ Internal storage.
- ✗ MODE\_WORLD\_WRITEABLE
- ✗ MODE\_WORLD\_READABLE
- ✗ External storage.
- ✓ Content providers.



# Encrypt Content



# Choose a Key

## Key Storage

```
private val keyStore: KeyStore = createAndroidKeyStore()  
private fun createAndroidKeyStore(): KeyStore {  
    val keyStore = KeyStore.getInstance("AndroidKeyStore")  
    keyStore.load(null) //loads parameters  
    return keyStore  
}
```

## Key Generation

```
fun createAndroidKeyStoreAsymmetricKey(alias: String): KeyPair {  
    val generator = KeyPairGenerator.getInstance("RSA", "AndroidKeyStore")  
    if (SystemServices.hasMarshmallow()) {  
        initGeneratorWithKeyGenParameterSpec(generator, alias)  
    } else {  
        initGeneratorWithKeyPairGeneratorSpec(generator, alias)  
    }  
    // Generates Key with given spec and saves it to the KeyStore  
    return generator.generateKeyPair()  
}
```

# Choose a Key

## Key Initialization

```
@TargetApi(Build.VERSION_CODES.M)
private fun initGeneratorWithKeyGenParameterSpec(
    generator: KeyPairGenerator,
    alias: String) {
    val builder = KeyGenParameterSpec.Builder(
        alias, KeyProperties.PURPOSE_ENCRYPT or KeyProperties.PURPOSE_DECRYPT)
        .setBlockModes(KeyProperties.BLOCK_MODE_ECB)
        .setEncryptionPaddings(KeyProperties.ENCRYPTION_PADDING_RSA_PKCS1)
    generator.initialize(builder.build())
}
```

## Key Management

```
fun getAndroidKeyStoreAsymmetricKeyPair(alias: String): KeyPair? {
    val privateKey = keyStore.getKey(alias, null) as PrivateKey?
    val publicKey = keyStore.getCertificate(alias)?.publicKey
    return if (privateKey != null && publicKey != null) {
        KeyPair(publicKey, privateKey)
    } else { null }
}
fun removeAndroidKeyStoreKey(alias: String) = keyStore.deleteEntry(alias)
```

# Encrypt & Decrypt

```
companion object {  
    var TRANSFORMATION_ASYMMETRIC = "RSA/ECB/PKCS1Padding"  
}
```

```
val cipher: Cipher = Cipher.getInstance(transformation)
```

```
fun encrypt(data: String, key: Key?): String {  
    cipher.init(Cipher.ENCRYPT_MODE, key)  
    val bytes = cipher.doFinal(data.toByteArray())  
    return Base64.encodeToString(bytes, Base64.DEFAULT)  
}
```

```
fun decrypt(data: String, key: Key?): String {  
    cipher.init(Cipher.DECRYPT_MODE, key)  
    val encryptedData = Base64.decode(data, Base64.DEFAULT)  
    val decodedData = cipher.doFinal(encryptedData)  
    return String(decodedData)  
}
```

## Algorithm

AES/CBC/NoPadding

AES/CBC/PKCS7Padding

AES/CTR/NoPadding

AES/ECB/NoPadding

AES/ECB/PKCS7Padding

AES/GCM/NoPadding

RSA/ECB/NoPadding

RSA/ECB/PKCS1Padding

RSA/ECB/OAEPWithSHA-1AndMGF1Padding

RSA/ECB/OAEPWithSHA-224AndMGF1Padding

RSA/ECB/OAEPWithSHA-256AndMGF1Padding

RSA/ECB/OAEPWithSHA-384AndMGF1Padding

RSA/ECB/OAEPWithSHA-512AndMGF1Padding

RSA/ECB/OAEPPadding

API 18+

API 23+

# Encrypt & Decrypt Example

```
var message = "Hello Word"
```

```
// Creates Android Key Store and provides manage functions  
private val keyStoreWrapper = KeyStoreWrapper(context)
```

```
// Create and Save asymmetric key  
keyStoreWrapper.createAndroidKeyStoreAsymmetricKey("MASTER_KEY")
```

```
aB9Ce9d5oM0/yloLQik0z8RovWHLmoQf3ovlCiz+D9+0/y7ZDfx6SpPYsKFIK3df079DNVIGVXIW  
63CIUrrc7zLPMCCCHCnzoeNJMqj2z0mFc1luXzr5mCDJYfU/63yPeUpCPuo3y1SfXPPPNYJKhz2pq  
TugVE+rWoql9019BwTKtBy80n0E4RDQnMe6M9FwCsV/k6NyFtml9iwwtGVuRGXpSgh9humMWT0Cu  
MxzHusdIaRaviY4mQLFS+iIyRC3Riu00xbkgTWpDs937Vfv3LSs1JSo2CvwqFEnMGhkGvMdjtNhJ  
vGnpzMYN/rYWt/cer8nreURscXN7o3IR8ZtPkA==
```

```
// Encrypt message with the key, using public key
```

```
var encryptedData = cipherWrapper.encrypt(message, masterKey?.public)
```

```
// Decrypt message with the key, using private key
```

```
var decryptedData = cipherWrapper.decrypt(data, masterKey?.private)
```

# Lock Screen

```
private val keyguardManager: KeyguardManager  
  
keyguardManager =  
    context.getSystemService(Context.KEYGUARD_SERVICE)  
  
fun isDeviceSecure(): Boolean =  
    if (hasMarshmallow()) keyguardManager.isDeviceSecure  
    else keyguardManager.isKeyguardSecure  
  
fun hasMarshmallow() =  
    Build.VERSION.SDK_INT >= Build.VERSION_CODES.M
```



# Prevent the app from starting

```
private var deviceSecurityAlert: AlertDialog? = null
```

```
override fun onStart() {
    super.onStart()
    if (!isDeviceSecure()) {
        deviceSecurityAlert = showDeviceSecurityAlert()
    }
}
// Used to block application if no lock screen is setup.
fun showDeviceSecurityAlert(): AlertDialog {
    return AlertDialog.Builder(context)
        .setTitle(R.string.lock_title)
        .setMessage(R.string.lock_body)
        .setPositiveButton(R.string.lock_settings, { _, _ ->
            context.openLockScreenSettings()
        })
        .setNegativeButton(R.string.lock_exit, { _, _ -> System.exit(0) })
        .setCancelable(BuildConfig.DEBUG)
        .show()
}
```

## Lock Screen

Secure lock screen hasn't set up. To continue working with this app, please go to Settings and set a lock screen method.

EXIT    SETTINGS

# Jetpack Security

- Based on Google Tink ([github.com/google/tink](https://github.com/google/tink)).
- Provides abstractions for encrypting Files and SharedPreferences objects.
- Strong security that balances great encryption and good performance.
- Maximum security.



# Setup

```
dependencies {  
    implementation "androidx.security:security-crypto:1.0.0"  
  
    // For Identity Credential APIs  
    implementation "androidx.security:security-identity-credential:1.0.0-alpha03"  
}
```

# Read Files

```
val mainKey = MasterKey.Builder(applicationContext)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM).build()

val fileToRead = "my_sensitive_data.txt"
val encryptedFile = EncryptedFile.Builder(applicationContext, File(DIRECTORY, fileToRead),
    mainKey, EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB).build()

val inputStream = encryptedFile.openFileInput()
val byteArrayOutputStream = ByteArrayOutputStream()
var nextByte: Int = inputStream.read()
while (nextByte != -1) {
    byteArrayOutputStream.write(nextByte)
    nextByte = inputStream.read()
}

val plaintext: ByteArray = byteArrayOutputStream.toByteArray()
```

# Write Files

```
val mainKey = MasterKey.Builder(applicationContext)
    .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
    .build()
```

```
val fileToWrite = "my_sensitive_data.txt"
val encryptedFile = EncryptedFile.Builder(
    applicationContext,
    File(DIRECTORY, fileToWrite),
    mainKey,
    EncryptedFile.FileEncryptionScheme.AES256_GCM_HKDF_4KB
).build()
```

```
val fileContent = "MY SUPER-SECRET INFORMATION"
    .toByteArray(StandardCharsets.UTF_8)
```

```
encryptedFile.openFileOutput().apply {
    write(fileContent)
    flush()
    close()
}
```

# Edit shared preferences

```
val sharedPrefsFile: String = FILE_NAME
val sharedPreferences: SharedPreferences = EncryptedSharedPreferences.create(
    applicationContext,
    sharedPrefsFile,
    mainKey,
    EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
    EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
)
with (sharedPreferences.edit()) {
    // Edit the user's shared preferences...
    apply()
}
```

# BiometricPrompt

DEMO

```
// Create BiometricPrompt instance in onCreate
val biometricPrompt = BiometricPrompt(
    this, // Activity
    ContextCompat.getMainExecutor(this),
    authenticationCallback
)

private val authenticationCallback = object : AuthenticationCallback() {
    override fun onAuthenticationSucceeded(
        result: AuthenticationResult
    ) {
        super.onAuthenticationSucceeded(result)
        // Unlocked -- do work here.
    }

    override fun onAuthenticationError(
        errorCode: Int, errString: CharSequence
    ) {
        super.onAuthenticationError(errorCode, errString)
        // Handle error.
    }
}
```

```
// To use
val promptInfo = PromptInfo.Builder()
    .setTitle("Unlock?")
    .setDescription("Would you like to unlock this key?")
    .setDeviceCredentialAllowed(true)
    .build()
biometricPrompt.authenticate(promptInfo)
```



# Secure Communication

```
// Load CAs from an InputStream
// (could be from a resource or ByteArrayInputStream or ...)
val cf: CertificateFactory = CertificateFactory.getInstance("X.509")
// From https://www.washington.edu/itconnect/security/ca/load-der.crt
val caInput: InputStream = BufferedInputStream(FileInputStream("load-der.crt"))
val ca:X509Certificate = URL("https://wikipedia.org").getCaInput.use {
    cf.generateCertificate(it) as X509Certificate
}
url.openConnection()
// Create a KeyStore containing our trusted CAs
val keyStoreType: KeyStoreType = KeyStore.getDefaultType()
val keyStore: KeyStore = KeyStore.getInstance(keyStoreType).apply {
    load(inputStream, null, System.out)
    setCertificateEntry("ca", ca)
}
// Create a TrustManager that trusts the CAs in our KeyStore
val tmfAlgorithm: String = TrustManagerFactory.getDefaultAlgorithm()
val tmf: TrustManagerFactory = TrustManagerFactory.getInstance(tmfAlgorithm).apply {
    init(keyStore)
}
// Create an SSLContext that uses our TrustManager
val context: SSLContext = SSLContext.getInstance("TLS").apply {
```



# SSL

```
// Open SSLSocket directly to gmail.com
val socket: SSLSocket = SSLSocketFactory.getDefault().run {
    createSocket("gmail.com", 443) as SSLSocket
}
val session = socket.session

// Verify that the certificate hostname is for mail.google.com
HttpsURLConnection.getDefaultHostnameVerifier().run {
    if (!verify("mail.google.com", session)) {
        throw SSLHandshakeException("Expected mail.google.com, found ${session.peerPrincipal}")
    }
}

// At this point SSLSocket performed certificate verification and
// we have performed hostname verification, so it is safe to proceed.

// ... use socket ...

socket.close()
```

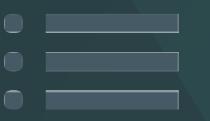




# Permissions

- Only use the permissions necessary for your app to work.
- Pay attention to permissions required by libraries.
- Be transparent.
- Make system accesses explicit.

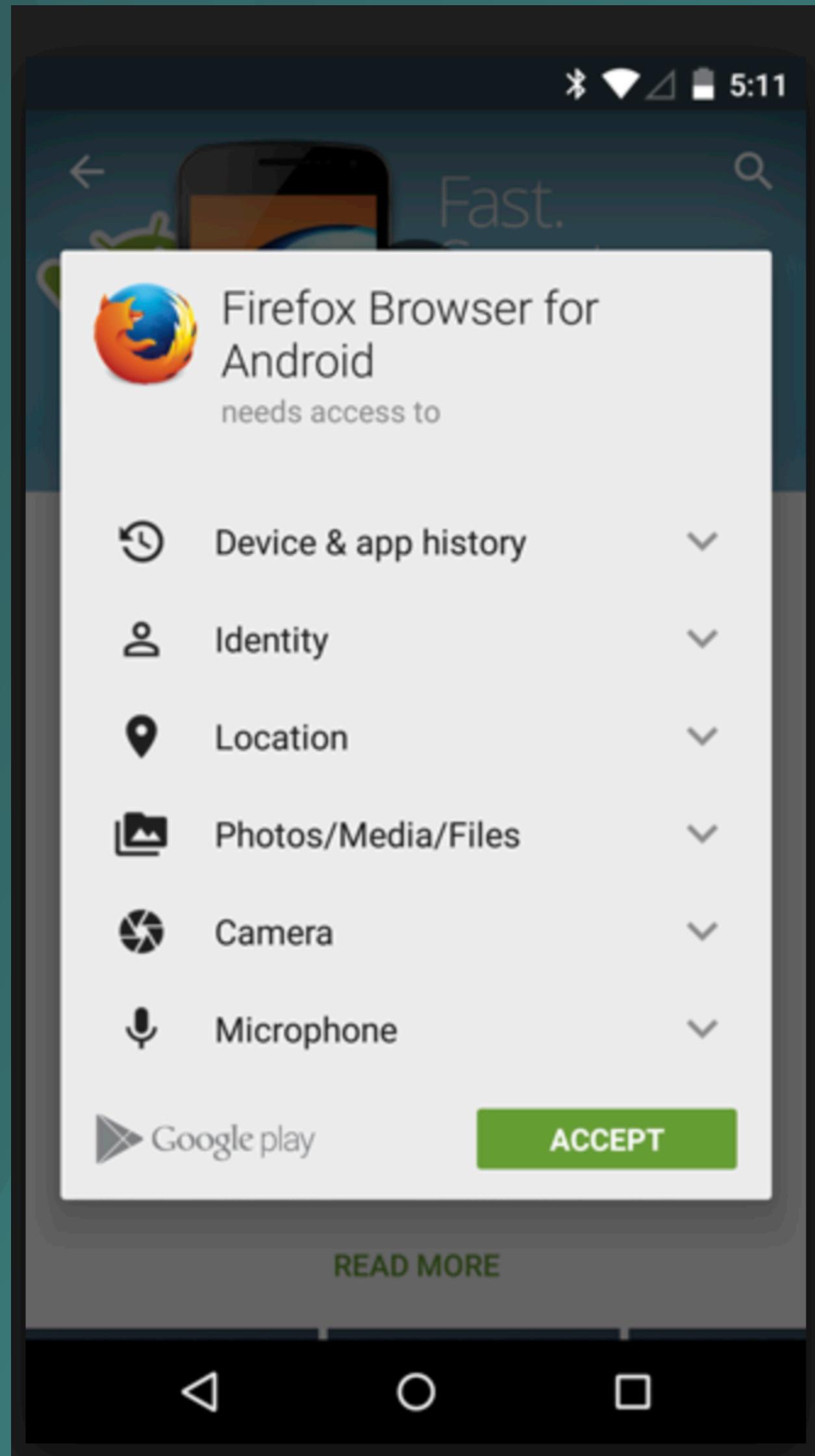
```
<manifest ...  
    package="com.example.snazzyapp">  
  
    <uses-permission  
        android:name=  
            "android.permission.SEND_SMS"/>  
    <!-- other permissions go here -->  
  
    <application ...>  
        ...  
    </application>  
</manifest>
```



# Permissions

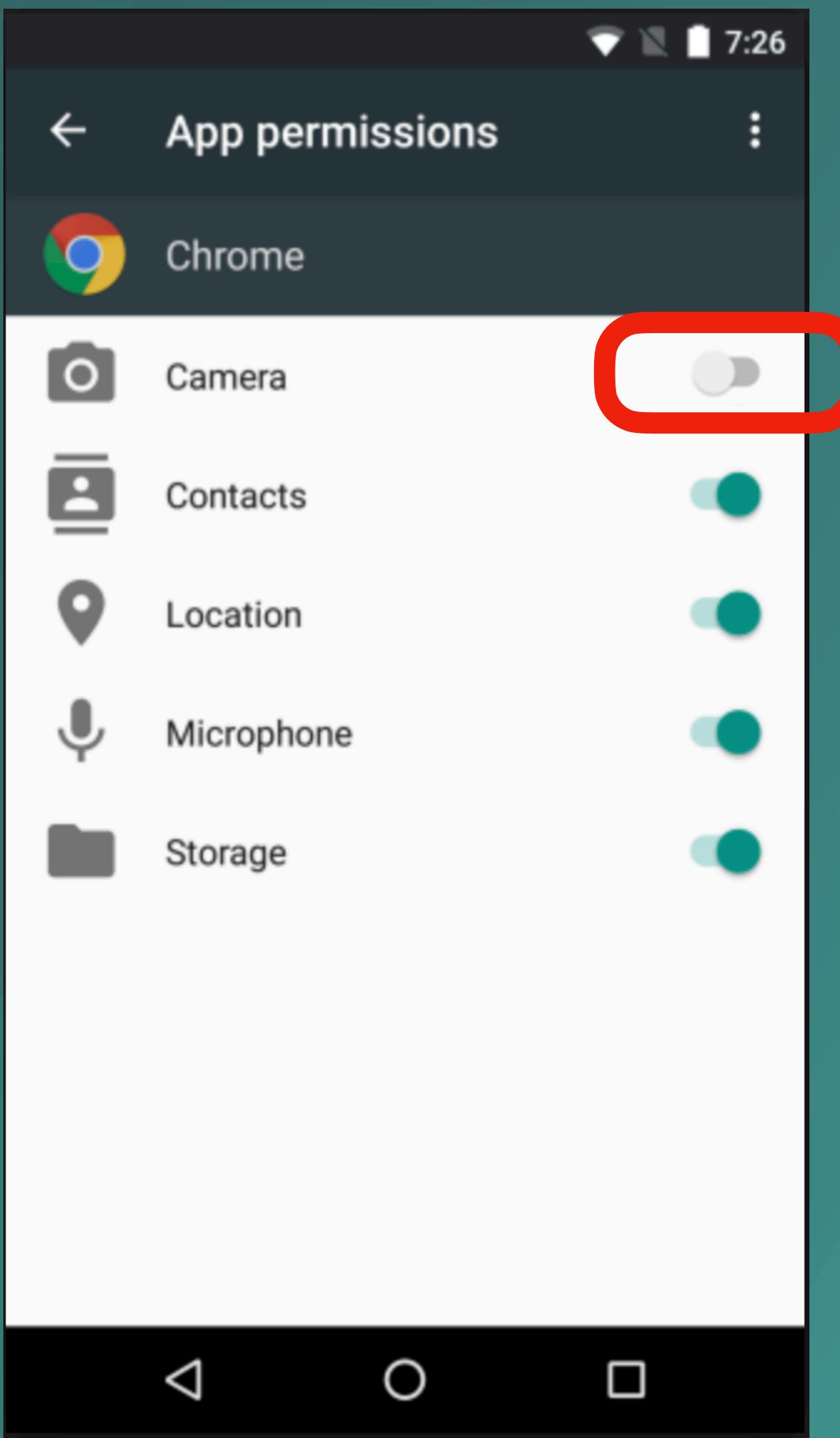
**Before:** Android 6.0

```
<manifest ...  
    package="com.mozilla.firefox">  
    ...  
    <uses-permission  
        android:name=  
            "android.permission.CAMERA"/>  
    <uses-permission  
        android:name=  
            "android.permission.MICROPHONE"/>  
    ...  
    <application ... >  
        ...  
    </application>  
</manifest>
```



# After: Android 6.0

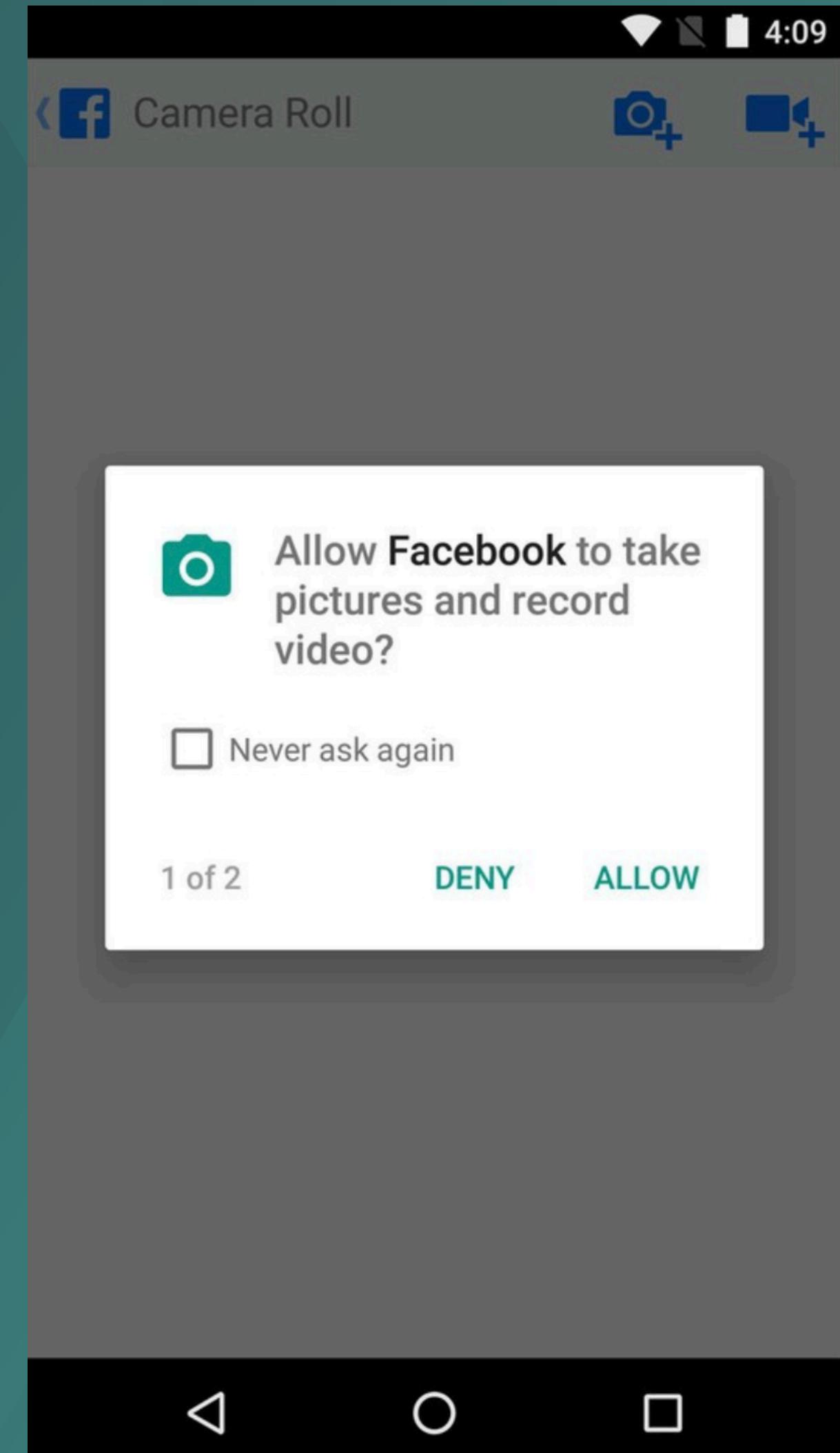
```
if (ContextCompat.checkSelfPermission(  
    thisActivity,  
    Manifest.permission.CAMERA)  
!= PackageManager.PERMISSION_GRANTED) {  
    // Permission is not granted  
}
```



```

if (ContextCompat.checkSelfPermission(
    thisActivity,
    Manifest.permission.CAMERA)
!= PackageManager.PERMISSION_GRANTED) {
    // Should we show an explanation?
    if (ActivityCompat.
        shouldShowRequestPermissionRationale(
            thisActivity,
            Manifest.permission.CAMERA)) {
        // Show an explanation to the user *asynchronously*
        // -- don't block this thread waiting for user's
        // response! After the user sees the explanation,
        // try again to request the permission.
    } else {
        // No explanation needed,
        // we can request the permission.
        ActivityCompat.requestPermissions(thisActivity,
            arrayOf(Manifest.permission.CAMERA),
            MY_PERMISSIONS_REQUEST_CAMERA)
        // MY_PERMISSIONS_REQUEST_CAMERA is an
        // app-defined int constant.
        // The callback method gets the result of the request.
    }
}

```



# Permissions Request Response

```
override fun onRequestPermissionsResult(requestCode: Int,
    permissions: Array<String>, grantResults: IntArray) {
    when (requestCode) {
        MY_PERMISSIONS_REQUEST_CAMERA -> {
            // If request is cancelled, the result arrays are empty.
            if ((grantResults.isNotEmpty() &&
                    grantResults[0] == PackageManager.PERMISSION_GRANTED)) {
                // permission was granted, yay!
                // Do the camera-related task you need to do.
            } else {
                // permission denied, boo!
                // Disable the functionality that depends on this permission.
            }
            return
        }
        // Add other 'when' lines to check for other
        // permissions this app might request.
    else -> {
        // Ignore all other requests.
    }
    }
}
```

# Dangerous Permissions

DEMO

Permission Group	Permissions
CALENDAR	<ul style="list-style-type: none"><li>• READ_CALENDAR</li><li>• WRITE_CALENDAR</li></ul>
CALL_LOG	<ul style="list-style-type: none"><li>• READ_CALL_LOG</li><li>• WRITE_CALL_LOG</li><li>• PROCESS_OUTGOING_CALLS</li></ul>
CAMERA	<ul style="list-style-type: none"><li>• CAMERA</li></ul>
CONTACTS	<ul style="list-style-type: none"><li>• READ_CONTACTS</li><li>• WRITE_CONTACTS</li><li>• GET_ACCOUNTS</li></ul>
LOCATION	<ul style="list-style-type: none"><li>• ACCESS_FINE_LOCATION</li><li>• ACCESS_COARSE_LOCATION</li></ul>
MICROPHONE	<ul style="list-style-type: none"><li>• RECORD_AUDIO</li></ul>
PHONE	<ul style="list-style-type: none"><li>• READ_PHONE_STATE</li><li>• READ_PHONE_NUMBERS</li><li>• CALL_PHONE</li><li>• ANSWER_PHONE_CALLS</li><li>• ADD_VOICEMAIL</li></ul>

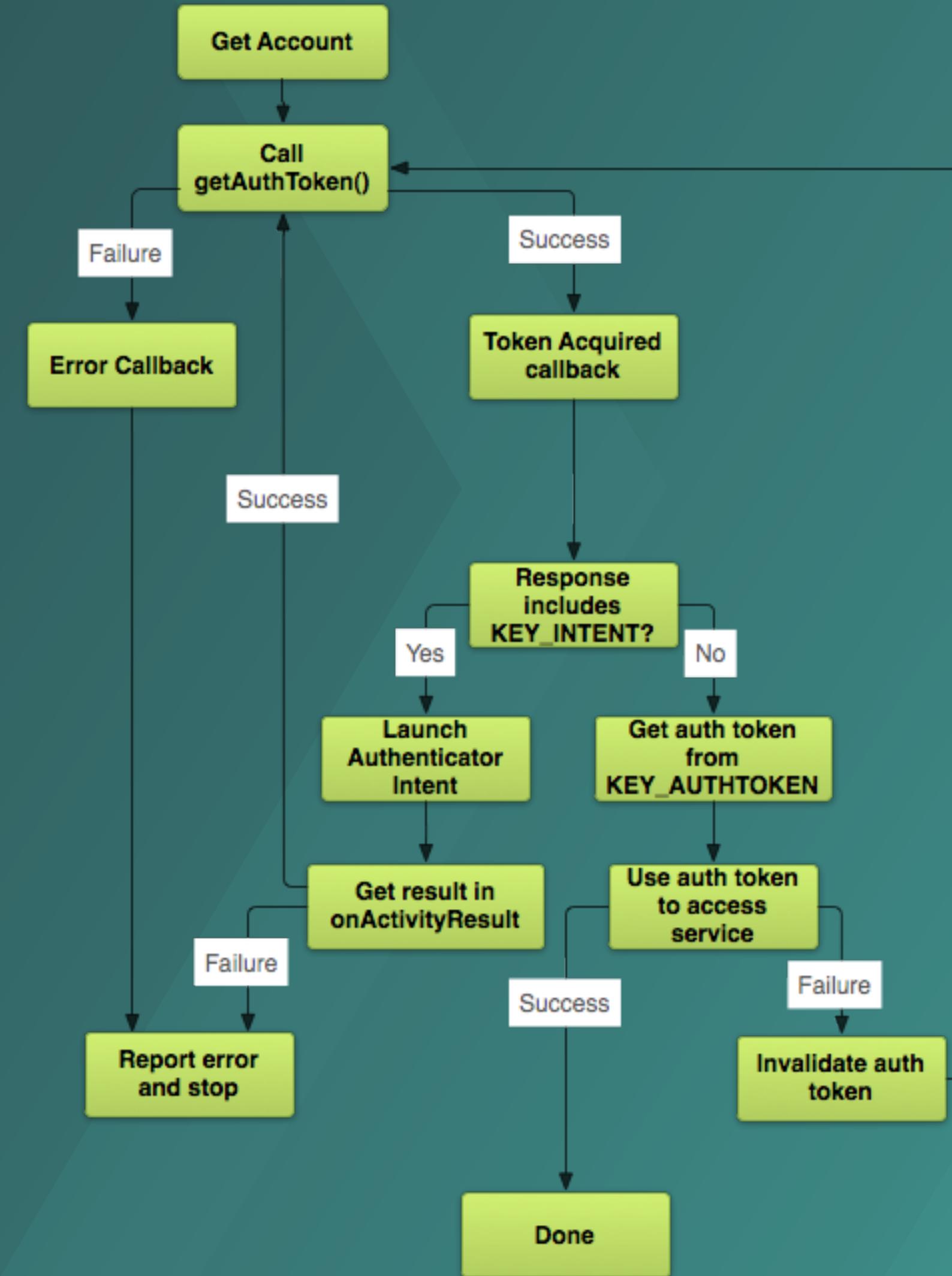
# OAuth2

- Industry-standard protocol for authorization.
- Focuses on client developer simplicity.
- Specific authorization flows for:
  - Web applications.
  - Desktop applications.
  - Mobile phones.
  - Others, eg: living room devices.



# Request an Auth Token

```
<manifest ...>
    <uses-permission
        android:name=
            "android.permission.ACCOUNT_MANAGER" />
    <uses-permission
        android:name=
            "android.permission.INTERNET" />
    ...
</manifest>
```



# Get the Auth Token

```
AccountManager am = AccountManager.get(this);
Bundle options = new Bundle();

am.getAuthToken(
    myAccount, // Account retrieved using getAccountsByType()
    "Manage your tasks", // Auth scope
    options, // Authenticator-specific options
    this, // Your activity
    new OnTokenAcquired(), // Callback called when a token
                           // is successfully acquired
    new Handler(new OnError())); // Callback called if an error occurs
```

# Get the Auth Token

```
AccountManager am = AccountManager.get(this);
```

```
Bundle options = new Bundle();
```

```
am.getAuthToken(  
    myAccount, // Account retrieved using getAccountsByType()  
    "Manage your tasks", // Auth scope  
    options, // Authenticator-specific options  
    this, // Your activity  
    new OnTokenAcquired(), // Callback called when a token  
                           // is successfully acquired  
    new Handler(new OnError())); // Callback called if an error occurs
```

```
private class OnTokenAcquired  
    implements AccountManagerCallback<Bundle> {  
    @Override  
    public void run(AccountManagerFuture<Bundle> result) {  
        // Get the result of the operation from the AccountManagerFuture.  
        Bundle bundle = result.getResult();  
  
        // The token is a named value in the bundle. The name of the value  
        // is stored in the constant AccountManager.KEY_AUTHTOKEN.  
        token = bundle.getString(AccountManager.KEY_AUTHTOKEN);  
        ...  
    }  
}
```

# Using the Auth Token

DEMO

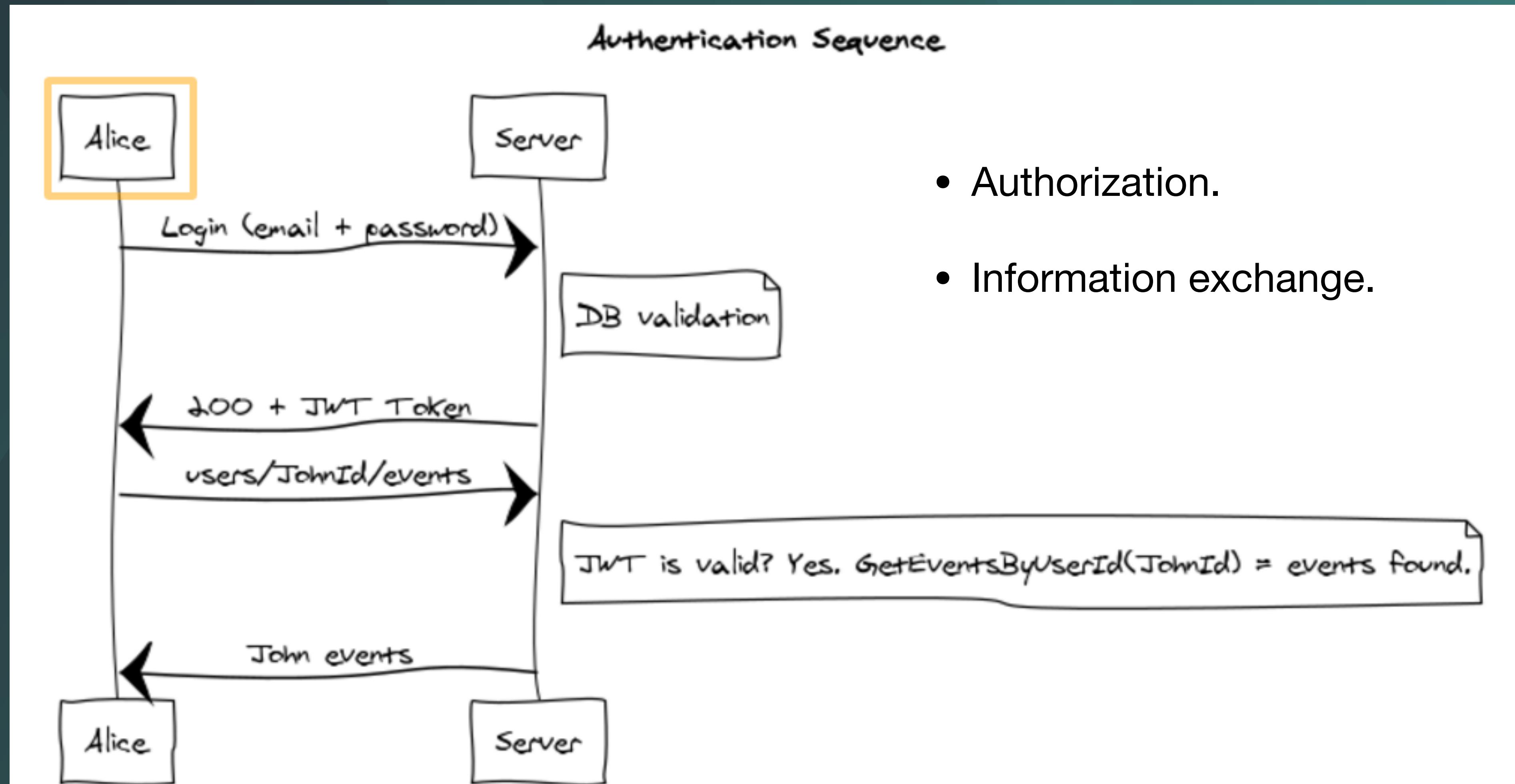
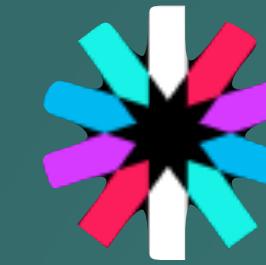
```
URL url = new URL(  
    "https://www.googleapis.com/tasks/v1/users/@me/lists?key=" + your_api_key);  
URLConnection conn = (HttpURLConnection) url.openConnection();  
conn.addRequestProperty("client_id", your client id);  
conn.addRequestProperty("client_secret", your client secret);  
conn.setRequestProperty("Authorization", "OAuth " + token);
```

# JWT (JSON Web Token)

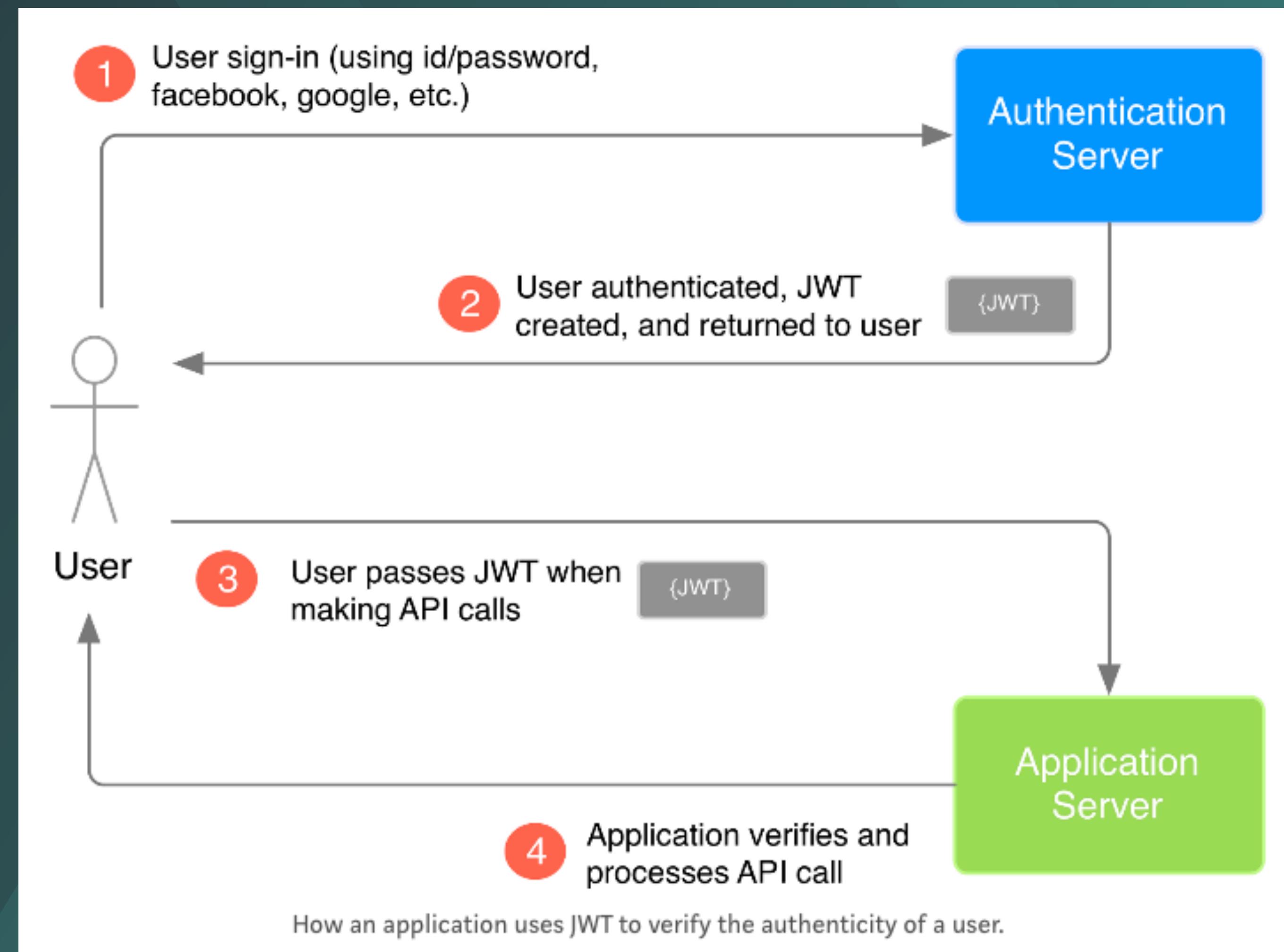
- Open standard, part of RFC 7519.
- Compact.
- Self-contained.
- Secure transmission.
- JSON objects.



# JWT - Usage



# JWT - Separate Server



# JWT - Model

DEMO

## Header:

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

## Payload:

```
{  
  "userId": "b08f86af-35da-48f2-8fab-cef3904660bd"  
}
```

## Signature:

```
// signature algorithm  
data = base64urlEncode(header) + “.” + base64urlEncode(payload)  
hashedData = hash(data, secret)  
signature = base64urlEncode(hashedData)
```

# Lecture outcomes

- Encrypt/Decrypt user's data.
- Establish secure connections.
- Understand security permissions.
- Using OAuth2.
- Using JWT.

