Android Fingerprint Authentication In Action

Ben Oberkfell

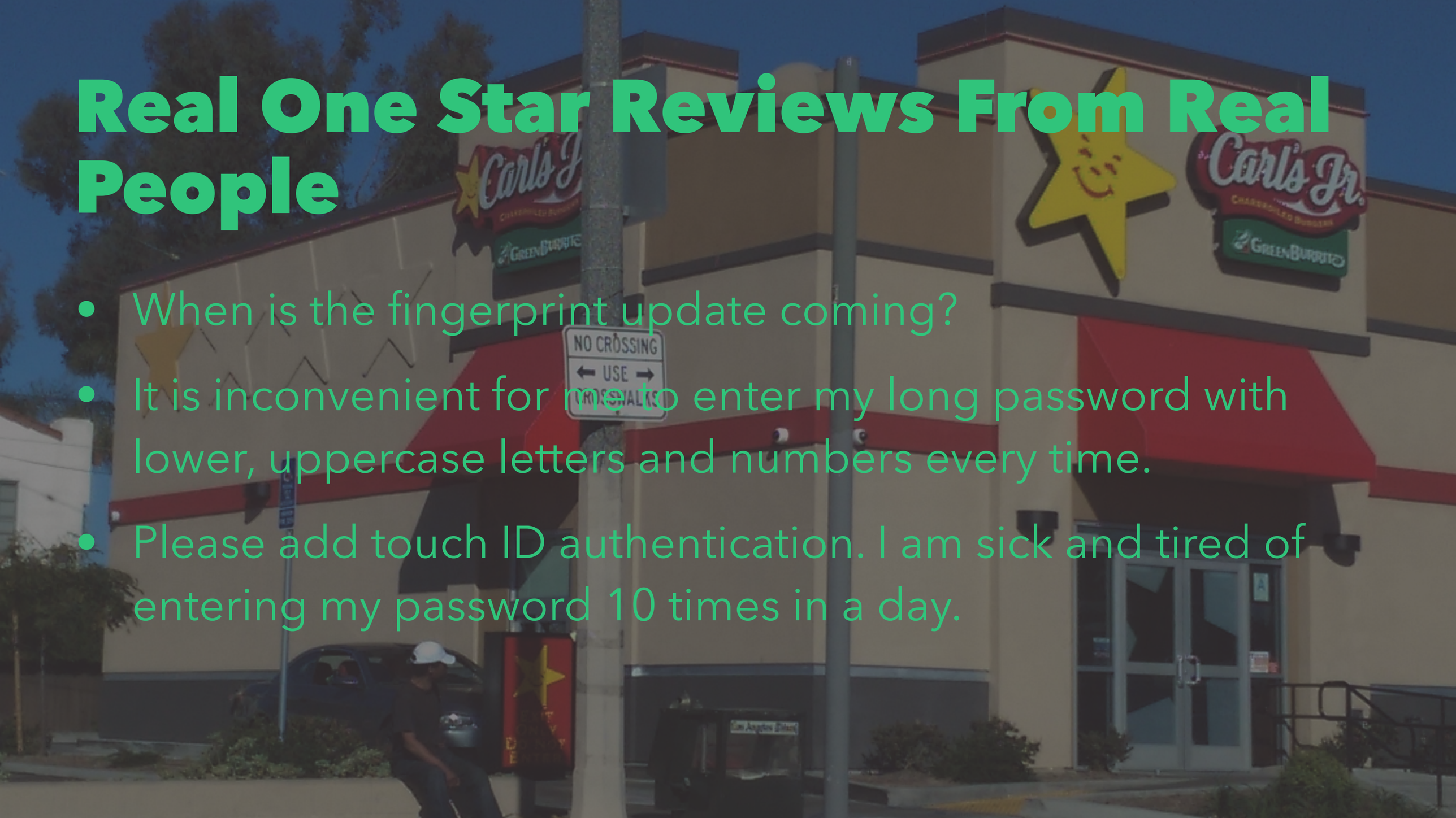
@benlikestocode
http://benlikestoco.de

On Tap

- Why/Where to Use Fingerprint
- How to Authenticate
- Designing It Right

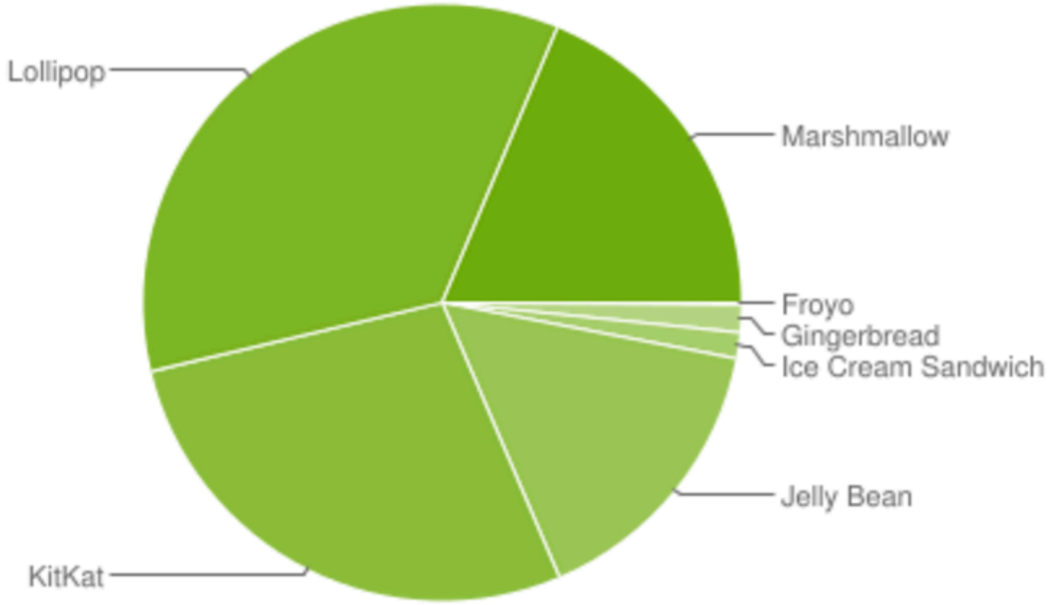

Carl's Jr.
GREEN BURRITO
NO CROSSING
USE
CROSSWALKS
EXIT
ONLY
DO NOT
ENTER

Real One Star Reviews From Real People



- When is the fingerprint update coming?
- It is inconvenient for me to enter my long password with lower, uppercase letters and numbers every time.
- Please add touch ID authentication. I am sick and tired of entering my password 10 times in a day.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	1.5%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.4%
4.1.x	Jelly Bean	16	5.6%
4.2.x		17	7.7%
4.3		18	2.3%
4.4	KitKat	19	27.7%
5.0	Lollipop	21	13.1%
5.1		22	21.9%
6.0	Marshmallow	23	18.7%



Data collected during a 7-day period ending on September 5, 2016.

Any versions with less than 0.1% distribution are not shown.

Bottom Line

This stuff is becoming table stakes, so let's learn how to do it!

Where Would You Want To Use Fingerprint?

Logins

Save the Trouble of Entering Passwords

Protect Critical User Flows

Things that Cost Money, Personal Security, etc

The Hard Part

How Do You Communicate a Successful Fingerprint Scan?

Would You Trust This?

```
POST /dominos
{
    "item": "Pepperoni Pizza",
    "quantity": 500,
    "deliveryAddress" : "1600 Pennsylvania Avenue",
    "fingerprintValidated" : true
}
```

If the goal is to protect the order flow, nope! 🍕


PATENT PENDING

What's In The Fingerprint Scanner?

If a device has a fingerprint reader and provides a developer-facing API, per the Compatibility Definition Document it has to

- have a hardware backed KeyStore
- gate keys usage via fingerprint

What Does This Mean For Us?

We can create encryption keys where the key cannot be used without authenticating with the fingerprint reader.

What Does This Mean For Us?

We can create encryption keys where the key cannot be used without authenticating with the fingerprint reader.

... which means if we can use the key, we know we successfully scanned an authorized fingerprint.

What Does This Mean For Us?

We can create encryption keys where the key cannot be used without authenticating with the fingerprint reader.

... which means if we can use the key, we know we successfully scanned an authorized fingerprint.

... so how do we use this for authentication?

Step 1

Create a Public/Private Key Pair

Make it require authentication to use

Step 2

Register the Public Key with Your Backend

And associate it with your user

Step 3

Sign Your Critical Requests with the Private Key

Step 4

Your Backend Verifies the Signed Request

Step 5

Happy Customers

Create the Key Pair

```
keyPairGenerator = KeyPairGenerator.getInstance(KeyProperties.KEY_ALGORITHM_RSA, "AndroidKeyStore");
keyPairGenerator.initialize(
            new KeyGenParameterSpec.Builder("DemoKey", PURPOSE_SIGN)
                    .setKeySize(2048)
                    .setDigests(DIGEST_SHA256)
                    .setSignaturePaddings(KeyProperties.SIGNATURE_PADDING_RSA_PKCS1)
                    // Here's the most important part. This enforces authentication before the private
                    // key can be used.
                    .setUserAuthenticationRequired(true)
                .build());

keyPairGenerator.generateKeyPair();
```

Signing A Request

```
public String signString(String data) {
    KeyStore keyStore = KeyStore.getInstance("AndroidKeyStore");
    keyStore.load(null);
    PublicKey publicKey = keyStore.getCertificate(KEY_NAME).getPublicKey();
    PrivateKey privateKey = keyStore.getKey(KEY_NAME, null);

    Signature signature = Signature.getInstance("SHA256withRSA");
    signature.initSign(getPrivateKey());
    signature.update(dataToSign.getBytes("UTF8"))

    byte[] bytes = cryptoObject.getSignature().sign();
    return Base64.encodeToString(bytes, Base64.NO_WRAP);
}
```

... but this won't work since we're not authenticated!

Authenticating First

Some Prereqs

- We'd use the `FingerprintManager` class to manage the fingerprint sensor.
- But to make life easy for pre-API 23, use `FingerprintManagerCompat`.
- Call for the `USE_FINGERPRINT` permission in your manifest. It'll be auto-granted.

Checking hardware

- Check for whether fingerprint reader exists

```
boolean hasHardware = fingerprintManager.isHardwareDetected();
```

- Check for fingerprints added to the device

```
boolean hasFingerprints = fingerprintManager.hasEnrolledFingerprints();
```

Authenticating First

```
public void requestFingerprintAuth(Signature signature,
                                   FingerprintManagerCompat.AuthenticationCallback callback) {

    CryptoObject cryptoObject = new CryptoObject(signature);
    CancellationSignal cancellationSignal = new CancellationSignal();

    FingerprintManagerCompat fingerprintManager = FingerprintManagerCompat.from(context);

    fingerprintManager.authenticate(cryptoObject, flags, cancellationSignal, callback, null);

}
```

Authenticating First

- This will start the scanner listening for fingerprints.
- `CryptoObject` wraps your crypto purpose (signing, encrypting, or message authentication code)
- `CancellationSignal` is a means of telling the fingerprint scanner to stop listening
- The `AuthenticationCallback` tells us what happened.

Caveat Emptor

Stop listening when your app is not foregrounded & active.
You can interfere with other apps or the lock screen if you keep listening.

Reacting to Successful Authentication

```
FingerprintManagerCompat.AuthenticationCallback callback =
    new FingerprintManagerCompat.AuthenticationCallback() {
            @Override
            public void onAuthenticationSucceeded(FingerprintManagerCompat.AuthenticationResult result) {
                super.onAuthenticationSucceeded(result);
                CryptoObject cryptoObject = result.getCryptoObject();

                // NOW we can use this Signature to sign!
                Signature signature = cryptoObject.getSignature();
            }

            // ....
}
```

Handling Unsuccessful Authentication

There are a few kinds of unsuccessful authentication:

- Failed. The wrong finger was scanned.
- Hard errors. The scanner gives up. E.g. hardware or fingerprint lockout errors.
- Soft errors. The scanner is still live. Fingerprint scanned too fast, etc.

Error callbacks

- Your `AuthenticationCallback` implementation deals with this too
- You get callbacks that give a message ID (which is referred to by a constant) and also some help/error text text.
- See `FingerprintManager`'s javadoc for these constants if you don't want to use Google's text

Unsuccessful Authentication

```
FingerprintManagerCompat.AuthenticationCallback callback =
    new FingerprintManagerCompat.AuthenticationCallback() {

        // ....
            // Bad Fingerprint
            @Override
            public void onAuthenticationFailed() {
                super.onAuthenticationFailed();
                fingerprintView.onError("Fingerprint Not Recognized");
            }

            // Hard Error
            public void onAuthenticationError(int errMsgId, CharSequence errString) {
                super.onAuthenticationHelp(errMsgId, errString);
                if (!canceled) {
                    fingerprintView.onError(errString.toString());
                }
            }

            // Soft Error
            @Override
            public void onAuthenticationHelp(int helpMsgId, CharSequence helpString) {
                super.onAuthenticationHelp(helpMsgId, helpString);
                if (!canceled) {
                    fingerprintView.onError(helpString.toString());
                }
            }
    }
}
```

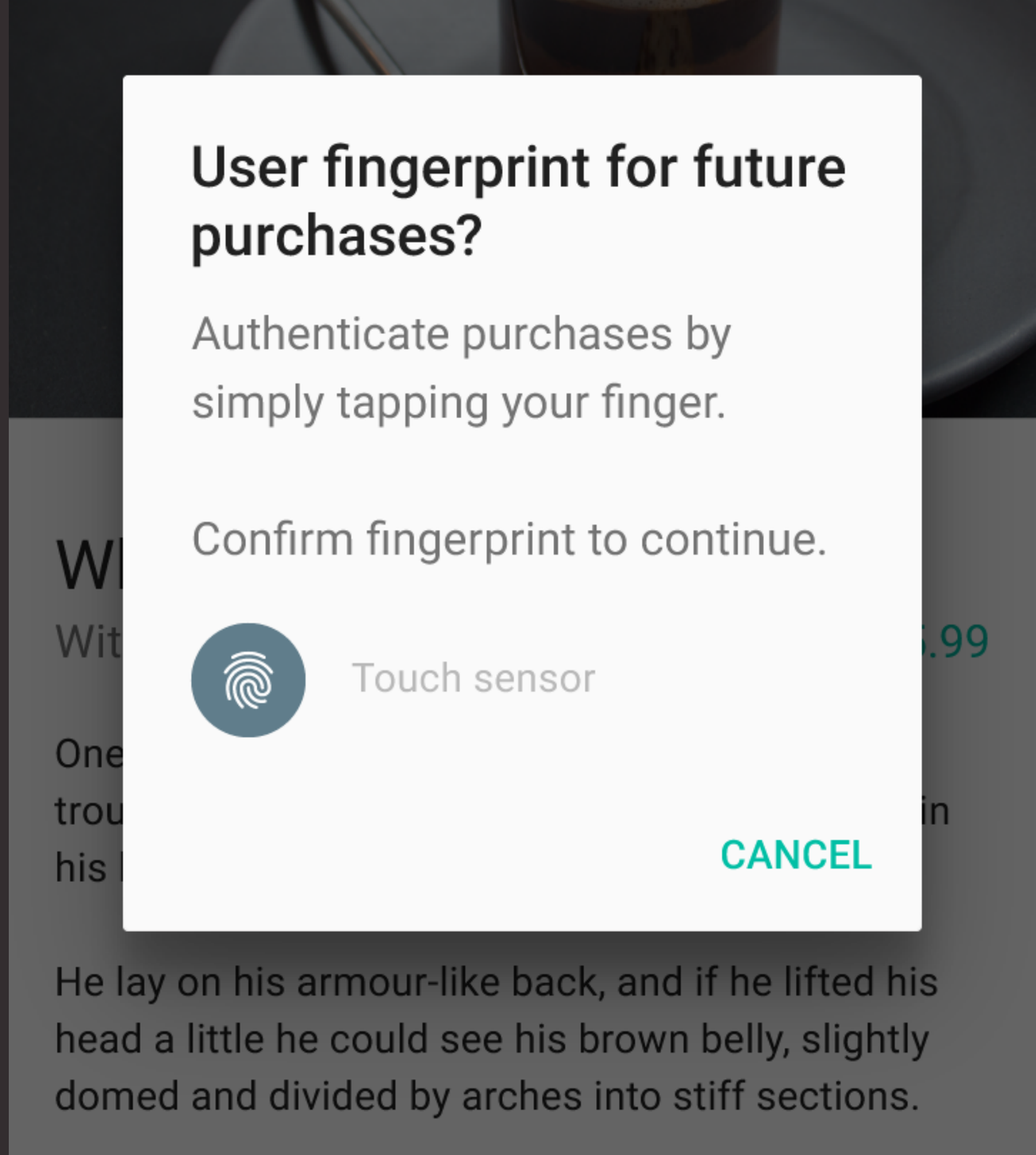
Okay, so we can scan fingerprint and sign things.

How does that help us?

Register the Public Key With Your Backend

Simple example

```
{
    'registerRequest' : {
        'username' : 'jdoe',
        'password' : 'p4ss4w0rd',
        'timestamp' : 1474000105154,
        'publicKey' : 'PUBLIC_KEY_BASE_64'
    },
    'signature' : 'SIGNATURE_BASE_64'
}
```



Register the Public Key

Be sure to sign your registration request, and validate it in the backend, to prove the user has the ownership of the private key and can use it.

Validate these

```
fun isRequestValidlySigned(request: SignedRequest<T>, publicKey: String) : Boolean {
        val pubKeyBytes = Base64.getDecoder().decode(publicKey)
        val signatureBytes = Base64.getDecoder().decode(request.signature)

        val kf = KeyFactory.getInstance("RSA")
        val key = kf.generatePublic(X509EncodedKeySpec(pubKeyBytes))
        val verify = Signature.getInstance("SHA256withRSA")
        verify.initVerify(key)
        verify.update(request.payload.messageForVerification().toByteArray())
        return verify.verify(signatureBytes)
    }
```

Now Make Protected Requests

```
{
    'purchaseRequest' : {
        "item": "Pepperoni Pizza",
        "quantity": 500,
        "deliveryAddress" : "1600 Pennsylvania Avenue",
        'timestamp' : 1474000105154,
    },
    'signature' : 'SIGNATURE_BASE_64'
}
```

If the signature validates against the data provided, then you know the user applied their fingerprint successfully.

Login instead of a purchase?

You're likely already getting a session token from your login.

Just add an API endpoint that takes a signed request for a session token.

Normalizing Data

You may find yourself needing to transform and normalize the data when signing and validating.

For instance:

```
Pepperoni Pizza|500|1600 Pennsylvania Avenue|1474000105154
```

Preventing Replay Attacks

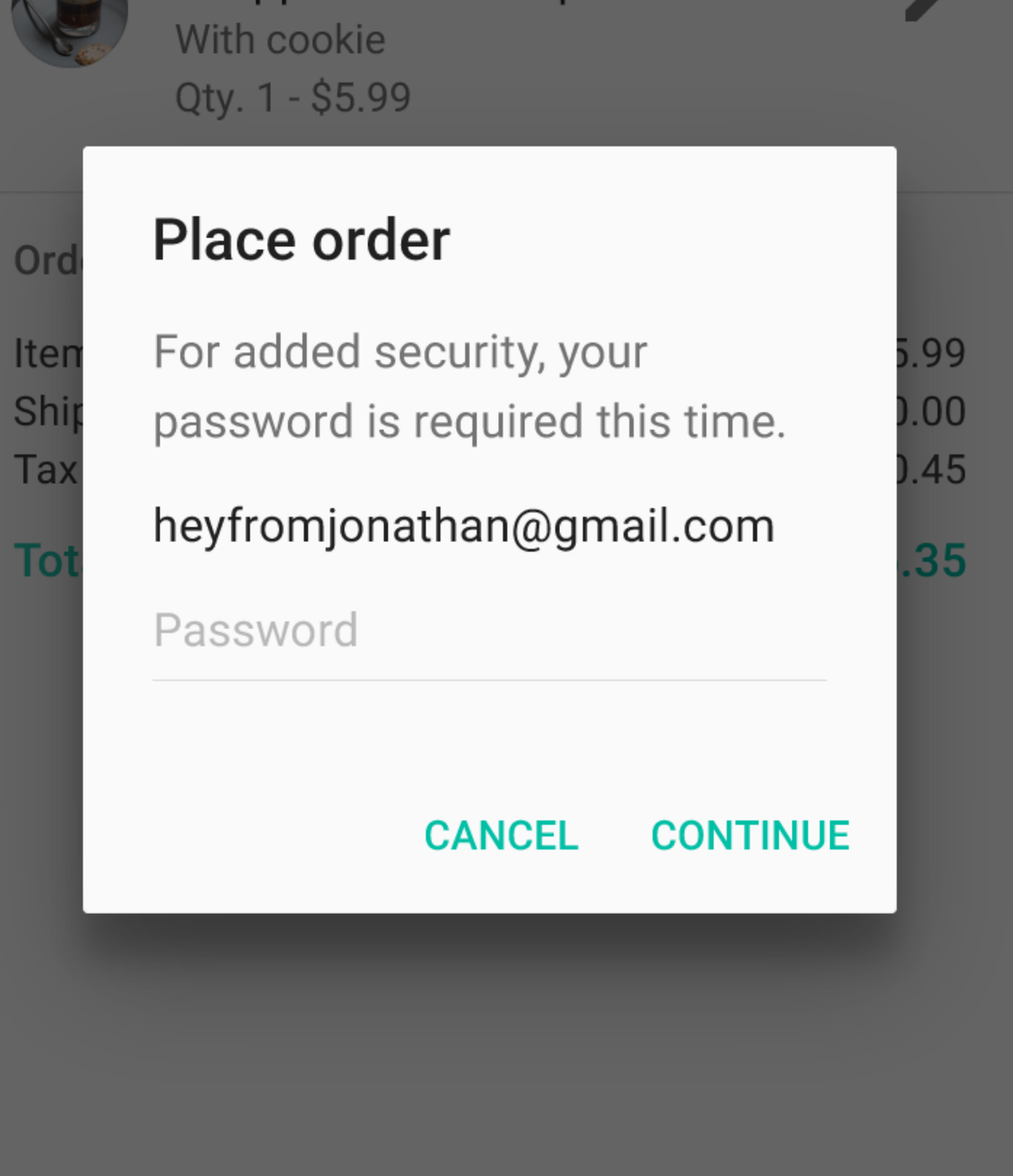
Use a cryptographic "nonce."

The timestamp in the request prevents replaying the same request later. Tampering with the timestamp would invalidate the signature.

You could also use other business data available to you if you can verify that the value hasn't been used multiple times.

Handling Key Invalidation

- If the device has its lockscreen disabled or a new fingerprint is added, all authenticated keys are invalidated.
- Attempting to prepare a `Signature` with such a key will result in an `InvalidKeyException`.
- Don't use the key anymore. You have to re-create a new keypair and enroll it in your backend.



Be Friendly

You should still *allow* the user to use their password in lieu of fingerprint. Put a "Use Password" button in the fingerprint dialog.

Now onto UI

Fingerprint

Android only

Fingerprint detection may be used to unlock a device, sign in to apps, and authenticate purchases using Google Play and some third-party apps.

To authenticate purchases using Fingerprint, display the Fingerprint authentication dialog.

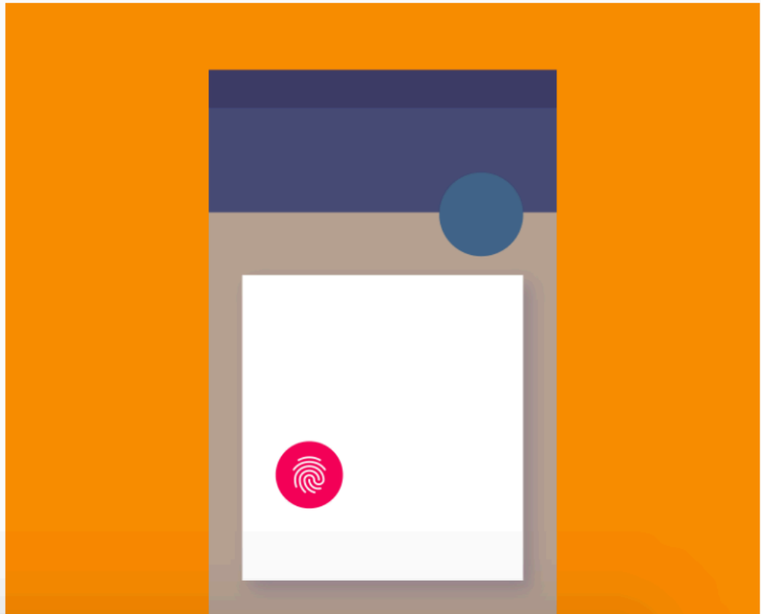
Fingerprint is not as secure as a strong PIN or password. Authentication alternatives include a user's account password, an app PIN, and device credentials.

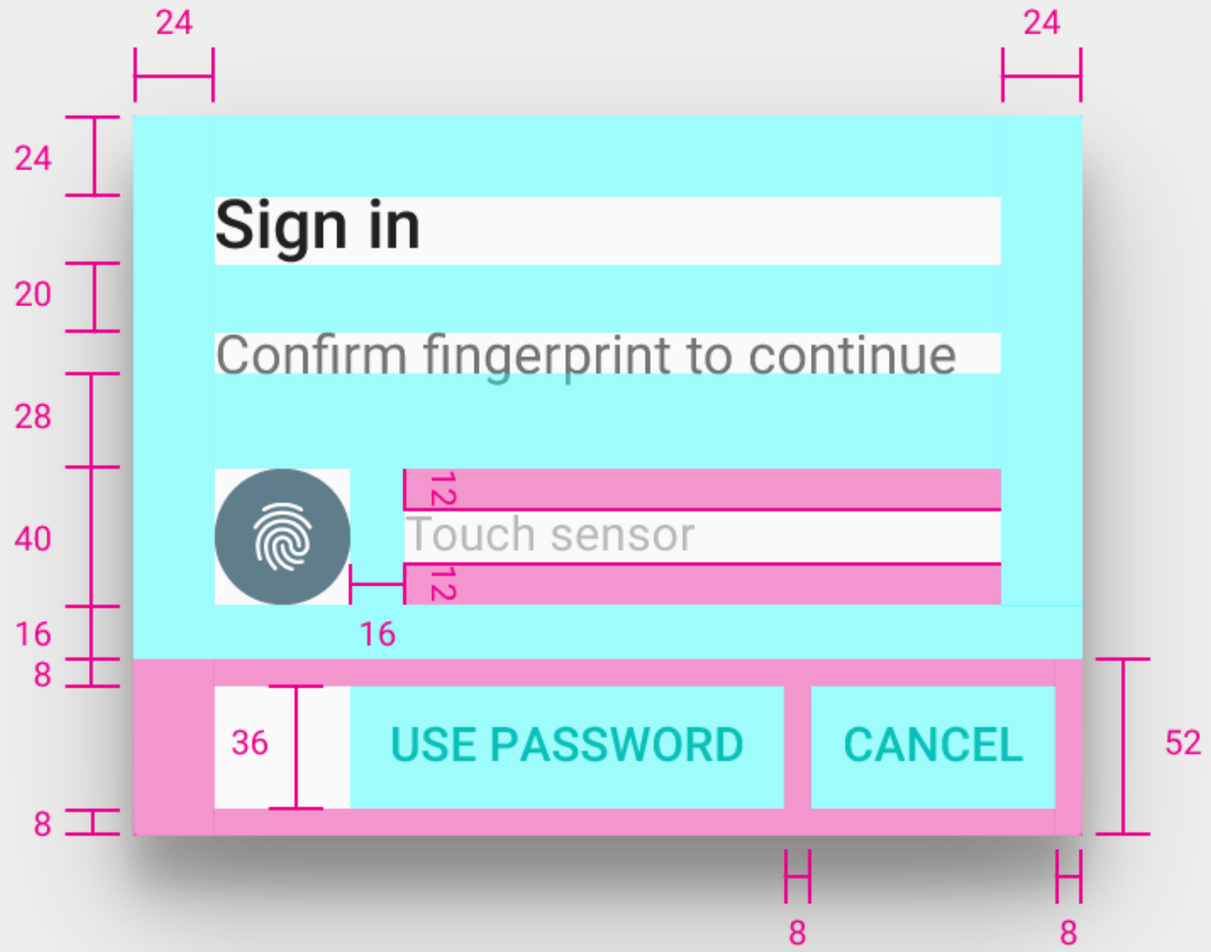
When to use

Upon opening the app
During your app's purchase flow
In your app settings
After enrollment

Icon

Fingerprint icon size: 24dp
Circle surrounding icon: 40dp




24
24
24
20
28
40
16
8
8
Sign in
Confirm fingerprint to continue
12
Touch sensor
12
16
36
USE PASSWORD
CANCEL
52
8
8

Consistency is Important

- Use the phrase "Confirm fingerprint."
- Take the error messages from Android unless you have a good reason not to.
- Users will be told to expect the fingerprint symbol. Display it as a standard icon (40dp circle with 24dp image).

Other Important Takeaways We Learned

- The "blessed" `CryptoObject` from the `FingerprintManager` success callback can be used only once.
- We needed to post that back to the main thread before we oculd use the `CryptoObject`.
- Use `FingerprintManagerCompat`!

Sample Code

http://tinyurl.com/android-fingerprint

Questions?

Thank You!

Image Credits (Creative Commons)

- Nexus 6p https://flic.kr/p/C7keQW
- Notary Embosser https://flic.kr/p/dNMmAi