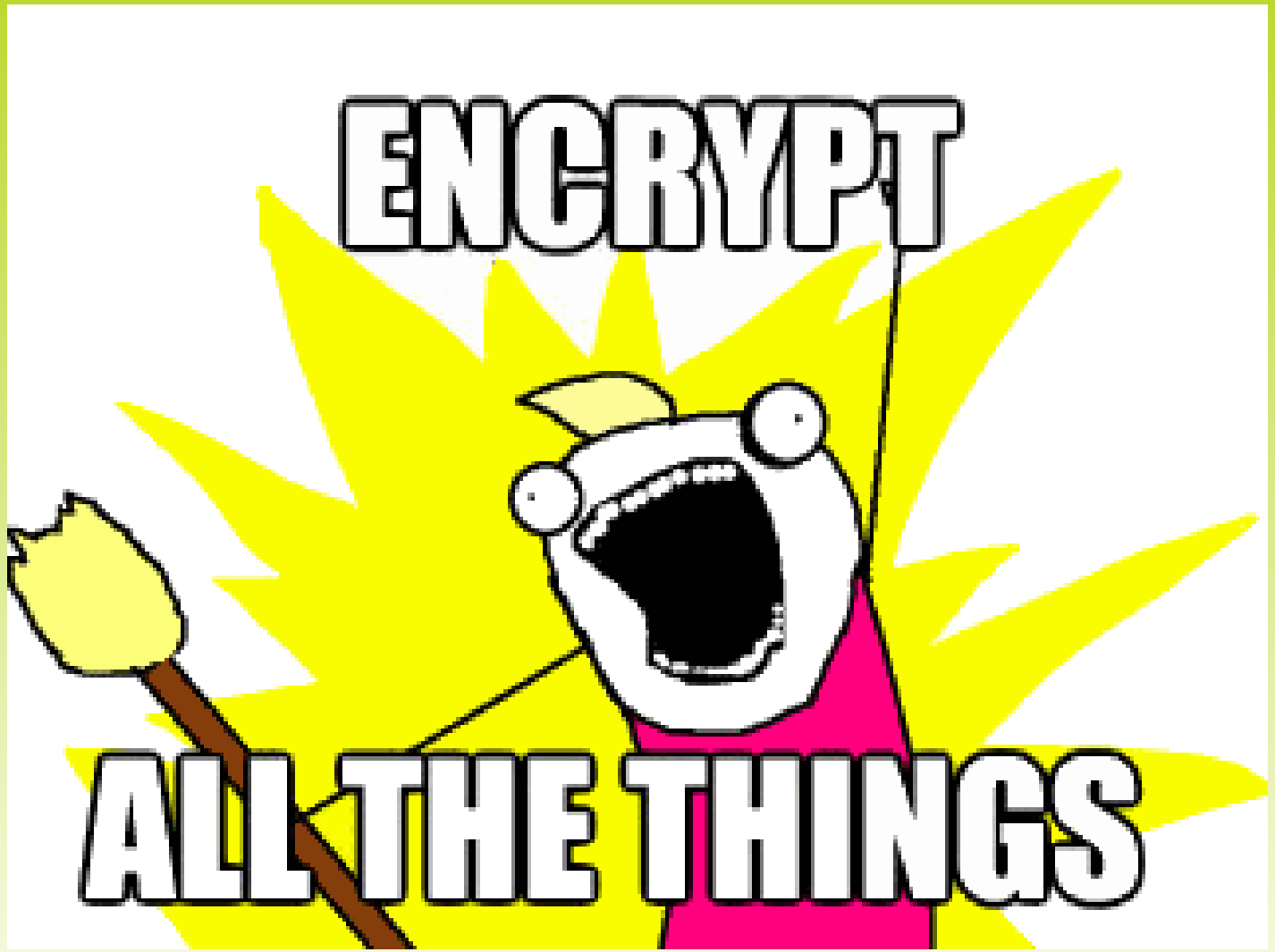
The Security Problems You Aren't Thinking About

Use SSL



Use SSL

- Normally, “It Just Works”
- Custom TrustManager
  - Self-signed certificates
  - Private certificate authority
    - Or other cases when root certificate not recognized by all versions of Android
- DO NOT BLINDLY ACCEPT EVERYTHING
  - FTC suits against Fandango, Credit Karma

Use SSL

- CWAC-Security
  - Helper library
  - Published by a balding guy
- TrustManagerBuilder
  - Fluent interface to create custom TrustManager[] for use with SSL
    - HttpsURLConnection
    - OkHTTP

Use SSL

```
TrustManagerBuilder builder=new TrustManagerBuilder(this)
  .selfSigned(R.raw.selfsigned, "foobar".toCharArray())
  .or()
  .useDefault();

SSLContext ssl=SSLContext.getInstance("TLS");

HttpsURLConnection conn=(HttpsURLConnection)new
URL(url).openConnection();

ssl.init(null, builder.buildArray(), null);
conn.setSSLSocketFactory(ssl.getSocketFactory());

// conn.getInputStream(), etc.
```

Deal With MITM

- MITM = Man in the Middle
  - Or Martian in the Middle, if you prefer
- Typical Implementation
  - You ask for HTTPS connection
  - One is supplied... by MITM proxy, not end server
  - MITM proxy now has access to decrypted communications

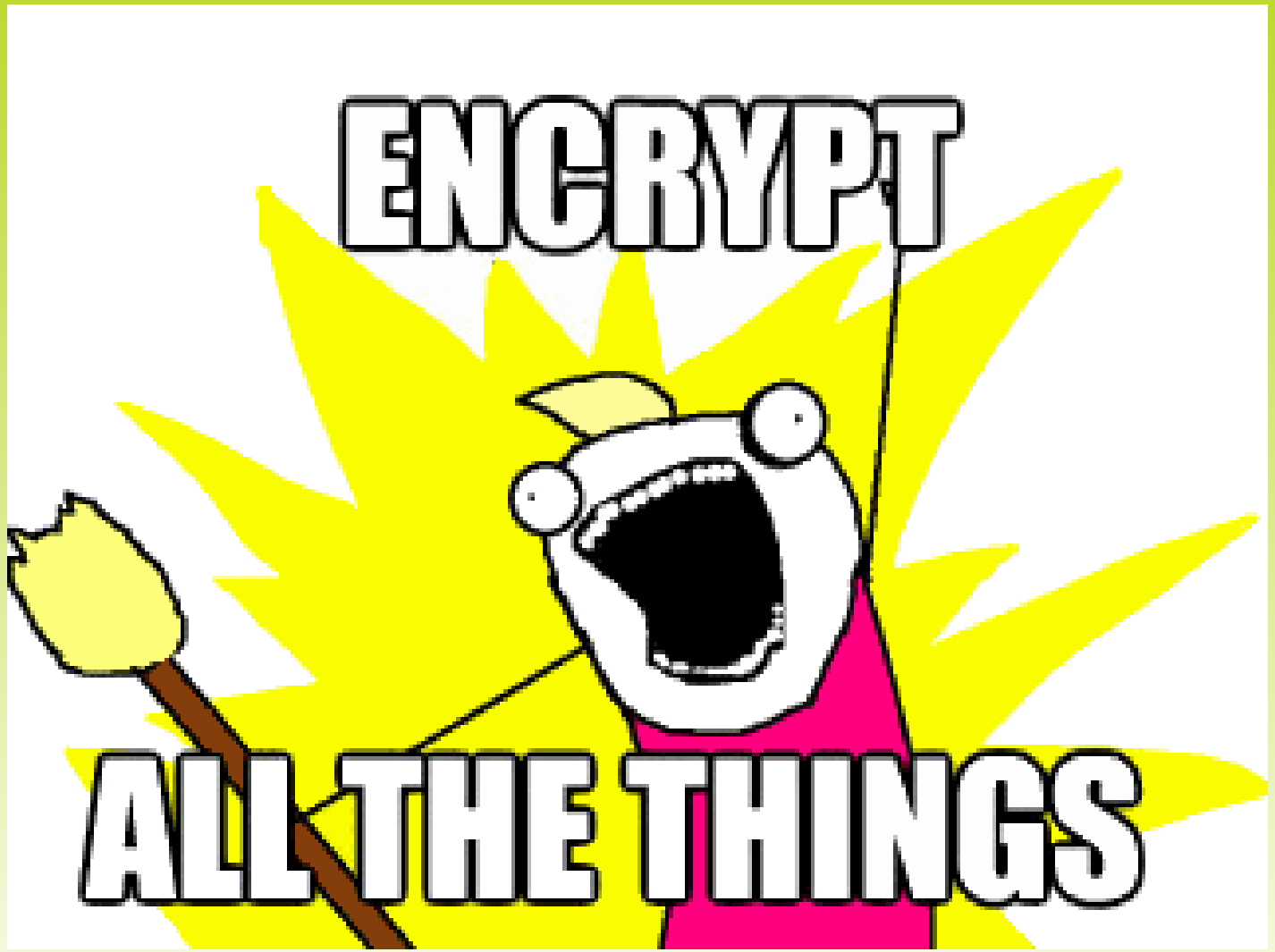
Deal with MITM

- How to Deal: Self-Signed Certificate
  - Certificate authorities for M:N communication
    - M browser vendors
    - N Web sites
  - Most apps have 1:1 communications
  - Self-signed certificate avoids CA and its risks

Deal With MITM

- How to Deal
  - Certificate Pinning
    - IOW, if there is only one possible certificate authority, do not accept anything else
  - Certificate Memorization
    - IOW, if the first one is OK, warn the user if we encounter a replacement certificate
      - Legitimate replacement
      - MITM

Encrypt Data at Rest



Encrypt Data at Rest

- Recommended Options
  - Today: SQLCipher
    - Encrypted database, on internal storage
    - Nearly identical API to native SQLite
    - Use with user-supplied passphrase
  - Tomorrow: IOCipher
    - Virtual filesystem, backed by SQLCipher

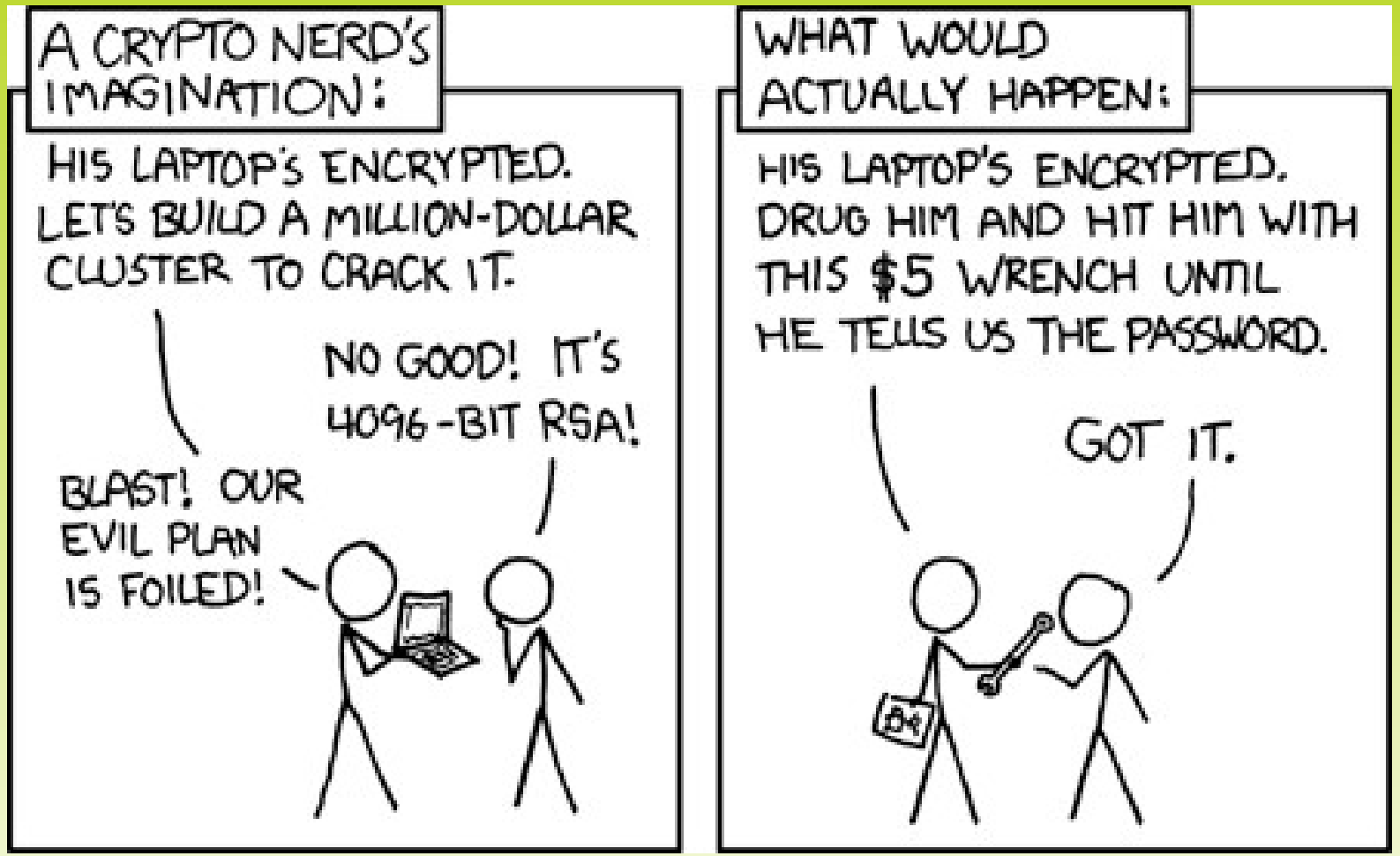
Encrypt Data at Rest

- Related Issues
  - Do not hardcode passwords (see: Whatsapp)
  - Saved passwords offer limited protection for user (see: Facebook Conceal)
  - Do your own backups
    - android:allowBackup=”false”
  - Do not create world-readable files on internal storage
    - Use FileProvider

Require More Security

- Device Administration
  - Force user to have strong device password
  - Strongly encourage user to have full-disk encryption enabled
  - Net: better security through the platform
  - Limitations
    - Not appropriate for all sorts of apps
    - Both vulnerable to brute forcing, $5 wrenches

Require More Security



Eliminate Accidental APIs

- Export What Needs to be Exported
  - Exported = third-party apps can access independently
  - Defaults
    - Activities, services, manifest-registered receivers: exported if there is an `<intent-filter>`
    - Receivers through `registerReceiver()`: always exported
    - ContentProvider through Android 4.1: exported
    - ContentProvider for Android 4.2+: not exported

Eliminate Accidental APIs

- Recommendations
  - Only use `<intent-filter>` if third party apps should be able to use independently
    - Example: launcher activity
  - Do not export ContentProvider
    - ...unless secured by permissions
  - Do not use `registerReceiver()` for intra-process communication
    - Use event bus

Eliminate Accidental APIs

- Beware the Control Channels
  - SMS
  - ServerSocket
  - GCM and other “push” engines
  - WifiDirect
  - Bluetooth
  - Etc.

Beware Spoof Services

- Scenario
  - You publish service with <intent-filter>
  - Somebody else publishes service with same <intent-filter>
  - Silent routing
    - Higher priority gets it
    - Tie/no priority = first one in wins
- Fixes
  - Client: signature validation
  - Service: signature-level permission, periodic scan for spoof services

Validate Your Partner Apps

- Other App May Be Malware!
  - ...and if you use some API it exposes or otherwise hand data to that app, the user may be in trouble
- Examples
  - PDF viewer
  - Payment engine

Validate Your Partner Apps

- Validation by Signature
  - Each APK has signing public key baked in
  - Accessible via PackageManager
  - CWAC-Security
    - SignatureUtils.getSignatureHash() returns SHA-256 hashed version of signature
    - Compare against known value
      - Or compare to original value for “trust on first use” model

```
int msg=R.string.package_invalid;

try {
  String hash=
      SignatureUtils.getSignatureHash(this, "com.android.settings");
  // String hash=
  // SignatureUtils.getOwnSignatureHash(this);

  if (getString(R.string.hash).equals(hash)) {
    msg=R.string.package_valid;
  }
}
catch (Exception e) {
  Log.e(getClass().getSimpleName(), "Exception in validation", e);
  msg=R.string.we_crashed;
}

Toast.makeText(this, msg, Toast.LENGTH_LONG).show();
```

<string name="hash">A9:99:84:D8:09:79:10:BB:96:56:85:73:69:32:15:07:F5:D1:4E:92:CA:32:AE:F9:E1:50:82:53:60:5B:CB:E3</string>

Validate Your Partner Apps

- Validation by Whole APK Hash
  - Avoids “Master Key Vulnerability” and any future similar bugs
  - Path to APK file available via PackageManager
  - Run cryptographic hash on its contents, compare to known value
    - Or compare to original value for “trust on first use” model

Leverage System Permissions

- Do Not Leak User Data
  - Scenario
    - You retrieve contact details, possible because you hold `READ_CONTACTS` permission
    - You expose API to third parties that happens to contain some of these contact details
    - You need to require some permission to access that API
      - `READ_CONTACTS`
      - Custom permission
    - Sans permission, you are leaking private user data

Leverage System Permissions

- Where to Apply
  - Activity, service, receiver: `android:permission`
  - ContentProvider
    - `android:permission`
    - `android:readPermission`
    - `android:writePermission`
  - Java
    - `checkCallingPermission()`
    - PackageManager

Apply Custom Permissions... Carefully

- Add `<permission>` Element
  - `android:name` = unique identifier
    - Do not use `android.permission` prefix, please!
  - `android:label` / `android:description` = what the user sees
    - Speak to users, not developers
    - String resources!
  - `android:protectionLevel`
    - `normal`, `dangerous`, `signature`, `system`

Custom Permission Vulnerability

- Rule: First One In Wins
  - First `<permission>` element for a given `android:name` determines behavior of that permission
  - App can define permission and have `<uses-permission>` and not use the permission for defense
- Net: apps installed before yours could hold your custom permissions
  - ...even signature ones, via downgrade

Custom Permission Vulnerability

- Environment
  - App A: defines and defends with custom permission
  - App B: defines and requests custom permission (attacker)
  - App C: just requests custom permission
- Scenario: App A, then App C
  - User notified about C's request, C gets access

Custom Permission Vulnerability

- Scenario: App C, then App A
  - User not notified, but C does not get permission, since not yet defined when it was installed
  - Problem for apps publishing SDKs
- Scenario: App A, then App B
  - Same as A → C: user informed, app gets permission

Custom Permission Vulnerability

- Scenario: App B, then App A
  - **PROBLEM**: User not notified about B's request
  - **PROBLEM**: B still gets permission
  - Downgrade Variant
    - A defines permission as signature
    - B defines permission as normal
    - B installed first, so permission is normal
    - B gets permission, despite no signature match

Custom Permission Vulnerability

- Not likely for bulk attacks, as normally no guarantee that B would be installed before A
  - If A installed first, user knows about B's request
  - In theory, eventually will be discovered as malware
- Bigger Risk: B Ships with Device
  - Used devices (not wiped or ROM mod)
  - “Presumed good” devices
    - Employer to employees
    - Gifts

```
@Override
public void onCreate(Bundle icicle) {
  super.onCreate(icicle);

  HashMap<PackageInfo, ArrayList<PermissionLint>> evildoers=
      PermissionUtils.checkCustomPermissions(this);

  if (evildoers.size() == 0 || !isFirstRun()) {
    Intent i=
        new Intent(Intent.ACTION_VIEW,
                   Uri.parse(FileProvider.CONTENT_URI + "test.pdf"));

    i.addFlags(Intent.FLAG_GRANT_READ_URI_PERMISSION);
    safelyStartActivity(i);
  }
```

```
else {
  for (Map.Entry<PackageInfo, ArrayList<PermissionLint>> entry : evildoers.entrySet()) {
    Log.e("SecurityDemoA", "This app holds the permission: "
        + entry.getKey().packageName);

    for (PermissionLint lint : entry.getValue()) {
      if (lint.wasUpgraded) {
        Log.e("SecurityDemoA",
              "...and they upgraded the protection level");
      }
      else if (lint.wasDowngraded) {
        Log.e("SecurityDemoA",
              "...and they downgraded the protection level");
      }

      if (lint.signaturesDiffer) {
        Log.e("SecurityDemoA",
              "...and they are signed by a different signing key, despite our signature-level
      }

      if (lint.proseDiffers) {
        Log.e("SecurityDemoA",
              "...and they altered the label or description");
      }
    }
  }

  Toast.makeText(this, R.string.evil, Toast.LENGTH_LONG).show();
}
```

Custom Permissions and "L"

- Good News
  - Two apps cannot have the same <permission> and different signing keys
- Bad News
  - "Deadly embrace" scenarios
  - User confusion
    - E.g., install plugin before host app

Sanitize Your Inputs

- Intent extras
  - See: PreferenceFragment bug
- Intent Uri
  - Value itself
  - Content obtained from ContentResolver or elsewhere
- ContentProvider parameters
  - SQL injection

Sanitize Your Inputs

- Parameters to AIDL-exposed service APIs
- Toolchain
  - Beware the **`gradlew`**!
  - Artifacts and dependencies
    - Build artifacts
    - Compile artifacts

Sanitize Your Outputs

- RemoteViews
  - Particularly for app widgets
- Clipboard
- Intent extras
  - Visible in recent tasks on older Android versions
- Wearables