Crypto tidbits: misuse, side channels
A paper from 2013 that looked at how Android apps use crypto, as a function of 6 “rules” that reflect the bare minimum a secure programmer should know:
A paper from 2013 that looked at how Android apps use crypto, as a function of 6 “rules” that reflect the bare minimum a secure programmer should know:

1. Do not use **ECB** mode for encryption. Period.
2. Do not use a **non-random IV** for CBC encryption.
3. Do not use **constant encryption keys**.
4. (see paper)
5. (see paper)
6. Do not use **static seeds** to seed `SecureRandom()`.
Crypto misuse in Android apps

15,134 apps from Google play used crypto;
Analyzed **11,748** of them

<table>
<thead>
<tr>
<th># apps</th>
<th>violated rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>5,656</td>
<td>Uses ECB (BouncyCastle default) (R1)</td>
</tr>
<tr>
<td>3,644</td>
<td>Uses constant symmetric key (R3)</td>
</tr>
<tr>
<td>2,000</td>
<td>Uses ECB (Explicit use) (R1)</td>
</tr>
<tr>
<td>1,932</td>
<td>Uses constant IV (R2)</td>
</tr>
<tr>
<td>1,636</td>
<td>Used iteration count &lt; 1,000 for PBE (R5)</td>
</tr>
<tr>
<td>1,629</td>
<td>Seeds SecureRandom with static (R6)</td>
</tr>
<tr>
<td>1,574</td>
<td>Uses static salt for PBE (R4)</td>
</tr>
<tr>
<td>1,421</td>
<td>No violation</td>
</tr>
</tbody>
</table>
BouncyCastle defaults

• BouncyCastle is a library that conforms to Java’s **Cipher** interface:

```java
Cipher c = Cipher.getInstance("AES/CBC/PKCS5Padding");
// Ultimately end up wrapping a ByteArrayOutputStream
// in a CipherOutputStream
```

• Java documentation specifies:

```
If no mode or padding is specified, provider-specific default values for the mode and padding scheme are used. For example, the SunJCE provider uses ECB as the default mode, and PKCS5Padding as the default padding scheme for DES, DES-EDE and Blowfish ciphers
```
<table>
<thead>
<tr>
<th>#Occurences</th>
<th>Symmetric encryption scheme</th>
</tr>
</thead>
<tbody>
<tr>
<td>5878</td>
<td>AES/CBC/PKCS5Padding</td>
</tr>
<tr>
<td>4803</td>
<td>AES *</td>
</tr>
<tr>
<td>1151</td>
<td>DES/ECB/NoPadding</td>
</tr>
<tr>
<td>741</td>
<td>DES *</td>
</tr>
<tr>
<td>501</td>
<td>DESede *</td>
</tr>
<tr>
<td>473</td>
<td>DESede/ECB/PKCS5Padding</td>
</tr>
<tr>
<td>468</td>
<td>AES/CBC/NoPadding</td>
</tr>
<tr>
<td>443</td>
<td>AES/ECB/PKCS5Padding</td>
</tr>
<tr>
<td>235</td>
<td>AES/CBC/PKCS7Padding</td>
</tr>
<tr>
<td>221</td>
<td>DES/ECB/PKCS5Padding</td>
</tr>
<tr>
<td>220</td>
<td>AES/ECB/NoPadding</td>
</tr>
<tr>
<td>205</td>
<td>DES/CBC/PKCS5Padding</td>
</tr>
<tr>
<td>155</td>
<td>AES/ECB/PKCS7Padding</td>
</tr>
<tr>
<td>104</td>
<td>AES/CFB8/NoPadding</td>
</tr>
</tbody>
</table>

Table 4: Distribution of frequently used symmetric encryption schemes. Schemes marked with * are used in ECB mode by default.
Crypto misuse in Android apps

15,134 apps from Google play used crypto;
Analyzied 11,748 of them

A failure of the programmers to know the tools they use
A failure of library writers to provide safe defaults
Side-channel attacks

- Cryptography concerns the *theoretical* difficulty in breaking a cipher
Side-channel attacks

• Cryptography concerns the *theoretical* difficulty in breaking a cipher

• But what about the information that a particular *implementation* could leak?
  • Attacks based on these are “**side-channel attacks**”
Side-channel attacks

- Cryptography concerns the *theoretical* difficulty in breaking a cipher

- But what about the information that a particular *implementation* could leak?
  - Attacks based on these are “*side-channel attacks*”
Simple Power Analysis (SPA)

- Interpret *power traces* taken during a cryptographic operation

- Simple power analysis can reveal the sequence of instructions executed
SPA on DES

Overall operation clearly visible:
Can identify the **16 rounds of DES**

**Figure 1:** SPA trace showing an entire DES operation.
SPA on DES

Overall operation clearly visible: Can identify the **16 rounds of DES**

**Figure 1:** SPA trace showing an entire DES operation.
SPASPA on DES on DES

Specific instructions are also discernible...
SPA on DES

**Figure 3:** SPA trace showing individual clock cycles.

Specific **instructions** are also discernible.
High-level idea

HypotheticalEncrypt(msg, key) {
    for(int i=0; i < key.len(); i++) {
        if(key[i] == 0)
            // branch 0
        else
            // branch 1
    }
}
HypotheticalEncrypt(msg, key) { 
    for(int i=0; i < key.len(); i++) {
        if(key[i] == 0) { // branch 0
            // branch 0
        } else { // branch 1
            // branch 1
        }
    }
}
High-level idea

HypotheticalEncrypt(msg, key) {
    for(int i=0; i < key.len(); i++) {
        if(key[i] == 0)
            // branch 0
        else
            // branch 1
    }
}

Implementation issue: If the execution path depends on the inputs (key/data), then SPA can reveal keys
High-level idea

HypotheticalEncrypt(msg, key) {
    for(int i=0; i < key.len(); i++) {
        if(key[i] == 0) // branch 0
            // branch 0
        else // branch 1
            // branch 1
    }
}

Implementation issue: If the execution path depends on the inputs (key/data), then SPA can reveal keys
HypotheticalEncrypt(msg, key) {
    for (int i=0; i < key.len(); i++) {
        if (key[i] == 0) {
            // branch 0
        } else {
            // branch 1
        }
    }
}

High-level idea

What if branch 0 had, e.g., a jmp that branch 1 didn’t?

Implementation issue: If the execution path depends on the inputs (key/data), then SPA can reveal keys.

What if branch 0 took longer? (timing attacks)
HypotheticalEncrypt(msg, key) {
    for(int i=0; i < key.len(); i++) {
        if(key[i] == 0) // branch 0
            // branch 0
        else // branch 1
            // branch 1
    }
}

High-level idea

What if branch 0 had, e.g., a jmp that branch 1 didn’t?
What if branch 0
- took longer? (timing attacks)
- gave off more heat?

Implementation issue: If the execution path depends on the inputs (key/data), then SPA can reveal keys
HypotheticalEncrypt(msg, key) {
    for (int i=0; i < key.len(); i++) {
        if(key[i] == 0)
            // branch 0
        else
            // branch 1
    }
}

High-level idea

What if branch 0 had, e.g., a jmp that branch 1 didn’t?

What if branch 0
- took longer? (timing attacks)
- gave off more heat?
- made more noise?
- ...

Implementation issue: If the execution path depends on the inputs (key/data), then SPA can reveal keys
Differential Power Analysis (DPA)

- SPA just visually inspects a single run
- DPA runs iteratively and reactively
  - Get multiple samples
  - Based on these, construct new plaintext messages as inputs, and repeat
Mitigating such attacks

• Hide information by making the execution paths depend on the inputs as little as possible
  • Have to *give up some optimizations* that depend on particular bit values in keys
    - Some Chinese Remainder Theorem (CRT) optimizations permitted remote timing attacks on SSL servers

• The crypto community should seek to design cryptosystems under the assumption that some information is going to leak
Other side-channel attacks

• Typical threat model: attacker doesn’t have root access to a particular machine
  • So we safely store keys in memory

• But what if the attacker had physical access to the machine?
Attack

• Attacker’s goal: reboot the machine into an OS that he or she controls to look at memory contents

• Challenge: memory loses state without power

5 sec  30 sec  60 sec  5 min
Cold boot attack

Memory loses its state slower at really cold temperatures

<table>
<thead>
<tr>
<th></th>
<th>Seconds w/o power</th>
<th>Error % at operating temp.</th>
<th>Error % at −50°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>41</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td>0.000095</td>
</tr>
<tr>
<td>B</td>
<td>360</td>
<td>50</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td>0.000036</td>
</tr>
<tr>
<td>C</td>
<td>120</td>
<td>41</td>
<td>0.00105</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>50</td>
<td>0.00144</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>50</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>50</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 2: Effect of cooling on error rates
Cold boot attack

Memory loses its state slower at really cold temperatures

<table>
<thead>
<tr>
<th></th>
<th>Seconds w/o power</th>
<th>Error % at operating temp.</th>
<th>Error % at -50°C</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>41</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td>0.000095</td>
</tr>
<tr>
<td>B</td>
<td>360</td>
<td>50</td>
<td>(no errors)</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td>0.000036</td>
</tr>
<tr>
<td>C</td>
<td>120</td>
<td>41</td>
<td>0.00105</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>42</td>
<td>0.00144</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>50</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>50</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 2: Effect of cooling on error rates
Cold boot attack

Memory loses its state slower at really cold temperatures

<table>
<thead>
<tr>
<th></th>
<th>Seconds w/o power</th>
<th>Error % at operating temp.</th>
<th>Error % at $-50^\circ C$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>60</td>
<td>41</td>
<td>(no errors) 0.000095</td>
</tr>
<tr>
<td></td>
<td>300</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>360</td>
<td>50</td>
<td>(no errors) 0.000036</td>
</tr>
<tr>
<td></td>
<td>600</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>120</td>
<td>41</td>
<td>0.00105</td>
</tr>
<tr>
<td></td>
<td>360</td>
<td>42</td>
<td>0.00144</td>
</tr>
<tr>
<td>D</td>
<td>40</td>
<td>50</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>80</td>
<td>50</td>
<td>0.18</td>
</tr>
</tbody>
</table>

Table 2: Effect of cooling on error rates
Cold boot attack

• **Launching** the attack:
  • Cool down the memory & then power off/take it out
  • Boot into your own OS
  • Scan the memory image for keys (non-trivial but doable, especially if the keys have a format that’s easy to detect)

• Some **defenses** against the attack:
  • Encrypt all of memory (increased CPU support for this)
  • Use trusted hardware (Xbox does this)
    - TPM (Trusted Platform Module) stores keys in hardware that is very difficult to inspect (some self-destruct)
  • Limit the amount of time keys live in memory
    - E.g., remove keys from memory when you enter Sleep mode