K-Hunt: Pinpointing Insecure Cryptographic Keys from Execution Traces

Juanru Li¹, Zhiqiang Lin², Juan Caballero³, Yuanyuan Zhang¹, Dawu Gu¹

¹ G.O.S.S.I.P, Shanghai Jiao Tong University, China
² The Ohio State University, USA
³ The IMDEA Software Institute, Spain

CCS’18, Toronto, Canada
October 16, 2018
Crypto Attacks and Defenses

Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2 @ CCS 2017
Existing Researches

- Crypto misuse on Mobile platforms
  - CryptoLint (Android) @ CCS 2013
  - iCryptoTracer (iOS) @ NSS 2014
  - NativeSpeaker (Android) @ Inscrypt 2017

- Crypto algorithm identification
  - Aligot @ CCS 2012
  - CipherXRay @ TDSC 2012
  - CryptoHunt @ Oakland 2017

- Parameter extraction
  - ReFormat @ ESORICS 2009
  - Dispatcher @ CCS 2009
  - MovieStealer @ Usenix Security 2013
**Crypto Keys: the Utmost Secrets**

- **Kerckhoffs's principle**
  - A cryptosystem should be secure even if everything about the system, *except the key*, is public knowledge.

- **Attacks against crypto keys**

  - *Lest we remember @ 2009*
  - *Heartbleed @ 2014*
  - *Foreshadow @ 2018*
How do we find insecure keys?

```c
1 uint8_t Key[16];
2 uint8_t Data[256] = {0};
3
4 void keygen(uint8_t * key, size_t len)
5 {
6       uint8_t seed[4];
7       for ( size_t i = 0; i < 4; ++i )
8           seed[i] = rand() & 0xff;
9       for ( size_t i = 0; i < len; ++i )
10           key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15       for ( size_t i = 0; i < len; ++i )
16           buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21       keygen(Key, 16);
22       encrypt(Data, 256);
23 }
```
How do we find insecure keys?

```c
1 uint8_t Key[16];
2 uint8_t Data[256] = {0};
3
4 void keygen(uint8_t * key, size_t len)
5 {
6     uint8_t seed[4];
7     for ( size_t i = 0; i < 4; ++i )
8         seed[i] = rand() & 0xff;
9     for ( size_t i = 0; i < len; ++i )
10         key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15     for ( size_t i = 0; i < len; ++i )
16         buf[i] ^= Key[i % 16];
17 }
18
19 int main()
20 {
21     keygen(Key, 16);
22     encrypt(Data, 256);
23 }
```

Key with inadequate randomness
How do we find insecure keys?

```
1 uint8_t Key[16];
2 uint8_t Data[256] = {0};
3 
4 void keygen(uint8_t * key, size_t len)
5 {
  6     uint8_t seed[4];
  7     for ( size_t i = 0; i < 4; ++i )
  8         seed[i] = rand() & 0xff;
  9     for ( size_t i = 0; i < len; ++i )
 10         key[i] = seed[i % 4];
11 }
12 
13 void encrypt( uint8_t * buf, size_t len )
14 {
  15     for ( size_t i = 0; i < len; ++i )
  16         buf[i] ^= Key[i % 16];
17 }
18 
19 int main()
20 {
  21     keygen(Key, 16);
  22     encrypt(Data, 256);
23 }
```

Forget to clean the used key buffer
Cases of insecurely used crypto keys

- Deterministically generated keys (DGK)

The entire lifetime of a crypto key
Cases of insecurely used crypto keys

- Deterministically generated keys (DGK)
- Insecurely Negotiated Keys (INK)

The entire lifetime of a crypto key
Cases of insecurely used crypto keys

- Deterministically generated keys (DGK)
- Insecurely Negotiated Keys (INK)
- Recoverable Keys (RK)

The entire lifetime of a crypto key:

1. Key generation
2. Key derivation
3. Key operating
4. Key cleaning
Crypto Program Analysis

1. Locating the used ciphers

```c
1 uint8_t Key[16];
2 uint8_t Data[256] = {0};
3
4 void keygen(uint8_t * key, size_t len)
5 {
6     uint8_t seed[4];
7     for (size_t i = 0; i < 4; ++i )
8         seed[i] = rand() & 0xff;
9     for (size_t i = 0; i < len; ++i )
10        key[i] = seed[i % 4];
11 }
12
13 void encrypt( uint8_t * buf, size_t len )
14 {
15     for (size_t i = 0; i < len; ++i )
16         buf[i] ^= key[i % 16];
17 }
18
19 int main()
20 {
21     keygen(Key, 16);
22     encrypt(Data, 256);
23 }
```
Crypto Program Analysis

1. Locating the used ciphers

```
uint8_t Key[16];
uint8_t Data[256] = {0};

void keygen(uint8_t * key, size_t len)
{
    uint8_t seed[4];
    for ( size_t i = 0; i < 4; ++i )
        seed[i] = rand() & 0xff;
    for ( size_t i = 0; i < len; ++i )
        key[i] = seed[i % 4];
}

void encrypt( uint8_t * buf, size_t len )
{
    for ( size_t i = 0; i < len; ++i )
        buf[i] ^= Key[i % 16];
}

int main()
{
    keygen(Key, 16);
    encrypt(Data, 256);
}
```

2. Understanding semantics of memory buffers
Crypto Program Analysis

1. Locating the used ciphers

2. Understanding semantics of memory buffers

3. Analyzing key derivation

```c
uint8_t Key[16];
uint8_t Data[256] = {0};

void keygen(uint8_t * key, size_t len)
{
    uint8_t seed[4];
    for (size_t i = 0; i < 4; ++i )
        seed[i] = rand() & 0xff;
    for (size_t i = 0; i < len; ++i )
        key[i] = seed[i % 4];
}

void encrypt( uint8_t * buf, size_t len )
{
    for (size_t i = 0; i < len; ++i )
        buf[i] ^= Key[i % 16];
}

int main()
{
    keygen(Key, 16);
    encrypt(Data, 256);
}
```
Crypto Program Analysis

1. Locating the used ciphers

```c
uint8_t Key[16];
uint8_t Data[256] = {0};

void keygen(uint8_t * key, size_t len)
{
    uint8_t seed[4];
    for ( size_t i = 0; i < 4; ++i )
        seed[i] = rand() & 0xff;
    for ( size_t i = 0; i < len; ++i )
        key[i] = seed[i % 4];
}

void encrypt( uint8_t * buf, size_t len )
{
    for ( size_t i = 0; i < len; ++i )
        buf[i] ^= Key[i % 16];
}

int main()
{
    keygen(Key, 16);
    encrypt(Data, 256);
}
```

2. Understanding semantics of memory buffers

3. Analyzing key derivation

4. Checking whether the used key is cleaned
Challenges

- Code and algorithm diversity
  - Proprietary ciphers
  - Customized implementations
Challenges

- **Code and algorithm diversity**
  - Proprietary ciphers
  - Customized implementations

- **Code complexity**
  - Large code base
  - Boundary identification of crypto functions
Challenges

- **Code and algorithm diversity**
  - Proprietary ciphers
  - Customized implementations

- **Code complexity**
  - Large code base
  - Boundary identification of crypto functions

- **Semantic recovering**
  - Deciding which memory buffers are crypto keys
Our insights

- Instead of identifying crypto algorithms (e.g., RSA)
  - We pinpoint basic blocks related to crypto transformations directly
Our insights

- Instead of identifying crypto algorithms (e.g., RSA)
  - We pinpoint basic blocks related to crypto transformations directly

- Instead of analyzing program binaries
  - We analyze execution traces to pinpoint crypto buffers
Our insights

- Instead of identifying crypto algorithms (e.g., RSA)
  - We pinpoint basic blocks related to crypto transformations directly

- Instead of analyzing program binaries
  - We analyze execution traces to pinpoint crypto buffers

- Instead of statically finding specific misuses
  - We **dynamically detect insecure key**
K-Hunt

- Binary code instrumentation based on Intel’s PIN framework
- Support x86/64 binary executables on Windows, Linux, and MacOS
- Comprises of two phases: **key pinpointing** and **insecure key detecting**
**Key Pinpointing**

---

**Step-I: Crypto Basic Block Identification**

- Arithmetic instructions as features
- Using multiple inputs to find data sensitive instructions
- Randomness test
### Key Pinpointing

- **Step-I: Crypto Basic Block Identification**
  - Arithmetic instructions as features
  - Using multiple inputs to find data sensitive instructions
  - Randomness test

- **Step-II: Crypto Key Buffer Identification**
  - Buffer size analysis
  - Execution context analysis
Insecure Key Detecting

- Taint-analysis based detection
Insecure Key Detecting

- Taint-analysis based detection

1. whether adequate entropy has been collected
Insecure Key Detecting

- Taint-analysis based detection

1. whether adequate entropy has been collected

2. whether both a remote input and a local input are involved
Insecure Key Detecting

ضرورة تحليل التراينت

1. هل تم جمع كمية كافية من الكتلوجين؟
2. هل يتأثراً كلاً من ورقة خارجية وورقة داخلية في العملية؟
3. هل تم نظافه كسر الكتلة بعد العملية الامادة؟

الخريطة: 
ورقة A ورقة B
الكلالج الكتلة وورقة البيانات
الخالج الناتج
Experiments

- Crypto Libraries
  - 10 libraries, three ciphers (AES, RSA, ECDSA)

- Crypto programs
  - 15 programs with variously implemented ciphers (Including proprietary ciphers)
## Key Identification Results

<table>
<thead>
<tr>
<th>Target</th>
<th>Algorithm</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>N</th>
<th>S</th>
<th>IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Botan</td>
<td>AES-256</td>
<td>53</td>
<td>13</td>
<td>7</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>1180</td>
<td>569</td>
<td>162</td>
<td>6</td>
<td>1024</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>958</td>
<td>921</td>
<td>300</td>
<td>2</td>
<td>224</td>
<td>128</td>
</tr>
<tr>
<td></td>
<td>Crypto++</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>1281</td>
<td>26</td>
<td>5</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>1949</td>
<td>924</td>
<td>214</td>
<td>6</td>
<td>896</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>1916</td>
<td>1425</td>
<td>305</td>
<td>8</td>
<td>288</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>Libgcrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>126</td>
<td>25</td>
<td>3</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>565</td>
<td>463</td>
<td>153</td>
<td>6</td>
<td>896</td>
<td>896</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>340</td>
<td>322</td>
<td>49</td>
<td>10</td>
<td>320</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td>LibSodium</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES NI-256</td>
<td>7</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>Ed25519</td>
<td>690</td>
<td>686</td>
<td>171</td>
<td>8</td>
<td>288</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>LibTomcrypt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>60</td>
<td>43</td>
<td>4</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>404</td>
<td>385</td>
<td>69</td>
<td>7</td>
<td>1152</td>
<td>1152</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>330</td>
<td>274</td>
<td>72</td>
<td>4</td>
<td>128</td>
<td>97</td>
</tr>
<tr>
<td></td>
<td>Nettle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>38</td>
<td>13</td>
<td>3</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>411</td>
<td>87</td>
<td>61</td>
<td>6</td>
<td>1152</td>
<td>896</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>186</td>
<td>92</td>
<td>39</td>
<td>8</td>
<td>288</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>mbedTLS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>44</td>
<td>40</td>
<td>13</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>154</td>
<td>138</td>
<td>39</td>
<td>12</td>
<td>1664</td>
<td>256</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>255</td>
<td>245</td>
<td>47</td>
<td>9</td>
<td>384</td>
<td>64</td>
</tr>
<tr>
<td></td>
<td>OpenSSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>58</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>210</td>
<td>175</td>
<td>41</td>
<td>10</td>
<td>1552</td>
<td>640</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>188</td>
<td>143</td>
<td>17</td>
<td>6</td>
<td>192</td>
<td>50</td>
</tr>
<tr>
<td></td>
<td>WolfSSL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>AES-256</td>
<td>50</td>
<td>36</td>
<td>4</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>RSA-2048</td>
<td>295</td>
<td>235</td>
<td>36</td>
<td>7</td>
<td>1152</td>
<td>1152</td>
</tr>
<tr>
<td></td>
<td>ECDSA</td>
<td>277</td>
<td>202</td>
<td>27</td>
<td>5</td>
<td>160</td>
<td>32</td>
</tr>
</tbody>
</table>

- **B1**: candidate basic blocks that contain a high arithmetic instruction ratio;
- **B2**: subset of B1 candidate basic blocks with a linear relation with the input size;
- **B3**: identified crypto basic blocks
- **N**: identified key buffers
- **S**: total size of the identified key buffers
- **IL**: input length of the identified key buffers.
Key Identification Results

- For 10 crypto libraries and 15 crypto programs, we successfully detected frequently used ciphers and their key buffers

- Proprietary ciphers and customized implementations of standard ciphers were detected

- Key buffers with different layouts are all pinpointed

<table>
<thead>
<tr>
<th>Target</th>
<th>Algorithm</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>N</th>
<th>S</th>
<th>IL</th>
</tr>
</thead>
<tbody>
<tr>
<td>7-zip</td>
<td>AES NI-256</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>Ccrypt</td>
<td>AES-256</td>
<td>44</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>Cryptcat</td>
<td>Twofish</td>
<td>54</td>
<td>14</td>
<td>7</td>
<td>1</td>
<td>160</td>
<td>varied</td>
</tr>
<tr>
<td>Cryptochief</td>
<td>Proprietary *</td>
<td>23</td>
<td>12</td>
<td>1</td>
<td>1</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Enpass</td>
<td>AES NI-256</td>
<td>8</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>Imagine</td>
<td>DSA-1024 *</td>
<td>241</td>
<td>72</td>
<td>12</td>
<td>5</td>
<td>464</td>
<td>928</td>
</tr>
<tr>
<td>IpMsg</td>
<td>AES-256</td>
<td>168</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>Keepass</td>
<td>AES-256</td>
<td>481</td>
<td>118</td>
<td>19</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>MuPDF</td>
<td>AES-128</td>
<td>262</td>
<td>46</td>
<td>4</td>
<td>1</td>
<td>176</td>
<td>16</td>
</tr>
<tr>
<td>PSCP</td>
<td>AES-256</td>
<td>195</td>
<td>9</td>
<td>5</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>Sage</td>
<td>ChaCha20 *</td>
<td>31</td>
<td>17</td>
<td>2</td>
<td>1</td>
<td>256</td>
<td>32</td>
</tr>
<tr>
<td>UltraSurf</td>
<td>RC4 *</td>
<td>191</td>
<td>79</td>
<td>6</td>
<td>1</td>
<td>1024</td>
<td>16</td>
</tr>
<tr>
<td>WannaCry</td>
<td>AES-128 *</td>
<td>26</td>
<td>12</td>
<td>3</td>
<td>1</td>
<td>352</td>
<td>16</td>
</tr>
<tr>
<td>Wget</td>
<td>AES-256</td>
<td>268</td>
<td>22</td>
<td>3</td>
<td>1</td>
<td>240</td>
<td>32</td>
</tr>
<tr>
<td>WinRAR</td>
<td>AES-128 *</td>
<td>181</td>
<td>58</td>
<td>3</td>
<td>1</td>
<td>176</td>
<td>32</td>
</tr>
<tr>
<td>WinRAR</td>
<td>AES-256 *</td>
<td>214</td>
<td>51</td>
<td>3</td>
<td>1</td>
<td>240</td>
<td>48</td>
</tr>
</tbody>
</table>
Performance Overhead

Runtime overhead (times) of three pintools of K-Hunt compared to null PIN
Detected Insecurely used keys

- 22/25 tested samples are found to use insecure keys!
- Even well-developed crypto libraries ignore the key cleaning
- DGK in proprietary encryption and verification schemes
- INK in certificate-less network communication

<table>
<thead>
<tr>
<th>Target</th>
<th>DGK</th>
<th>INK</th>
<th>NMZ</th>
<th>MMZ</th>
<th>RKPS</th>
<th>RKPH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Botan</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Crypto++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Libgcrypt</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LibSodium</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>LibTomcrypt</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Nettle</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>GnuTLS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>mbedTLS</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>OpenSSL</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>WolfSSL</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7-zip</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Crypt</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Cryptcat</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Cryptochief</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Enpass</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Imagine</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>IpMsg</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Keepass</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>MuPDF</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>PSCP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Sage</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>UltraSurf</td>
<td>-</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>WannaCry</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Wget</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>WinRAR</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
</tbody>
</table>

- **NMZ**: null memory zeroing
- **MMZ**: manual memory zeroing
- **RKPS**: recoverable key in program stack
- **RKPH**: recoverable key in program heap
Case Study: DGK in Imagine

Imagine (an image and animation viewer) uses DSA as its registration algorithm.

**DSA Signing**

\[ r = g^k \mod p \mod q \] (1)

\[ s = k^{-1}(H(m) + x \cdot r) \mod q \] (2)

**DSA Verifying**

\[ w = s^{-1} \mod q \] (3)

\[ u_1 = H(m) \cdot w \mod q \] (4)

\[ u_2 = r \cdot w \mod q \] (5)

\[ v = (g^{u_1} \cdot y^{u_2} \mod p) \mod q \] (6)

“a hard-coded \( k \) leads an attacker to compute the private key \( x \) with a legal pair of signature \((r, s)\), and thus to forge the signature”

\[ x = r^{-1}(k \cdot s - H(m)) \mod q \]
Case Study: RK in Libsodium

Libsodium’s patch against insecurely used AES round keys:
https://github.com/jedisct1/libsodium/commit/28cac20a7bedd2ff35379874e63a33f6168ba31a

We have made responsible disclosure to the vulnerable software vendors and some of them quickly addressed the issue.

Unfortunately, some software vendors did not even response...
Conclusion

- **K-Hunt**, a dynamic analysis system to detect insecurely used keys in binary code, is developed.

- Three types of insecurely used crypto keys (DGK, INK, RK) are detected using **K-Hunt**.

- Insecurely used keys are found in both crypto libraries (e.g., *libsodium*) and crypto programs (e.g., *Keepass*).
Fortune cookie

A challenge related to the DSA case study
• placed in K-Hunt’s Github repository
• https://github.com/gossip-sjtu/k-hunt

First 10 people to solve the challenge would receive a gift 🎁
• Get the gift at the Ant financial desk outside

Email the answer to loccs@sjtu.edu.cn
Thank you & Questions?

We also build new crypto libraries:

• **YogCrypt** — Chinese standard ciphers (SM2, 3, 4) in Rust
  
  • [https://yogcrypt.org](https://yogcrypt.org)

• **YogSM** — Chinese standard ciphers (SM2, 3, 4) with Intel’s new hardware instructions
  
  • [https://yogsm.org](https://yogsm.org)