Developing Effective Test Strategies for Cryptographic Algorithm Implementations

Sydney Pugh$^{1,3}$, M S Raunak$^{1,2}$, D. Richard Kuhn$^2$, Raghu Kacker$^2$

1 Loyola University Maryland, Baltimore, MD USA
2 National Institute of Standards and Technology, Gaithersburg, MD USA
3 University of Pennsylvania, Philadelphia, PA USA

LWC Workshop
Nov 06, 2019
Testing Cryptographic Algorithms is Difficult

Issues

• Lacks test-oracle
  • Developing a test oracle is very costly, often infeasible
• Implementation of cryptographic algorithms are inherently complex
  • Dense with bit manipulations and condition predicates
• Traditional test strategies are generally ineffective
  • Statement and branch coverage

Approach

• Systematically design tests suitable for cryptographic algorithms
Recent Development in Crypto Algorithm Testing

Strong evidence of the application of metamorphic testing to cryptographic algorithm implementations

Appears in IEEE Transactions on Reliability, vol. 67, no. 3, Sept. 2018

Finding Bugs in Cryptographic Hash Function Implementations

Nicky Mouha, Mohammad S Raunak, D. Richard Kuhn, and Raghu Kacker

2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)

Systematic Testing of Post-Quantum Cryptographic Implementations Using Metamorphic Testing

Sydney Pugh, M S Raunak, D. Richard Kuhn, and Raghu Kacker
Previous Testing Success – SHA-3

- BCT: 22%
- BET: 20%
- MUT: 37%
- Any: 48%
Previous Testing Success – PQC

- Bit Contribution: 2.56% Failed or Error
- Bit Exclusion: 46.15% Failed or Error
- Bit Verify: 34.62% Failed or Error
- Encrypt Decrypt Check: 40.96% Failed or Error
Need for LWC: Growth of Small Computing Devices

• Radio Frequency IDentification (RFID) Tags
• Smart Cards
• Microcontrollers
• Embedded Systems
• Sensor Networks
• IoT Devices
Lightweight Cryptography (LWC)

Develop a new standard for Authenticated Encryption with Associated Data (AEAD) and hash functions designed for resource-constrained devices

**Timeline**

- Aug 2018: Formal Call for Proposals
- Feb 2019: Deadline for Submissions (57 received)
- Apr 2019: 56 Submissions Advance to Round 1
- **Sept 2019: Round 2 Begins**

<table>
<thead>
<tr>
<th>ACE</th>
<th>PHOTON-Beetle</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCON</td>
<td>Pyjmask</td>
</tr>
<tr>
<td>Bleep64</td>
<td>Qameleon</td>
</tr>
<tr>
<td>CiliPadi</td>
<td>Quartet</td>
</tr>
<tr>
<td>CLAE</td>
<td>REMUS</td>
</tr>
<tr>
<td>CLX</td>
<td>Romulus</td>
</tr>
<tr>
<td>COMET</td>
<td>SAEAES</td>
</tr>
<tr>
<td>DryGASCON</td>
<td>Saturnin</td>
</tr>
<tr>
<td>Elephant</td>
<td>Shamash &amp; Shamashash</td>
</tr>
<tr>
<td>ESTATE</td>
<td>SiMPLE</td>
</tr>
<tr>
<td>FlexAEAD</td>
<td>SIV-Rijndael256</td>
</tr>
<tr>
<td>ForkAE</td>
<td>SIV-TEM-PHOTON</td>
</tr>
<tr>
<td>Fountain</td>
<td>Skinny</td>
</tr>
<tr>
<td>GAGE-InGAGE</td>
<td>SNEIK</td>
</tr>
<tr>
<td>GIFT-COFB</td>
<td>SPARKLE</td>
</tr>
<tr>
<td>Gimli</td>
<td>SPIX</td>
</tr>
<tr>
<td>Grain-128AEAD</td>
<td>SpoC</td>
</tr>
<tr>
<td>HERN &amp; HERON</td>
<td>Spook</td>
</tr>
<tr>
<td>HYENA</td>
<td>Subterranean 2.0</td>
</tr>
<tr>
<td>ISAP</td>
<td>SUNDAE-GIFT</td>
</tr>
<tr>
<td>KNOT</td>
<td>Sycon</td>
</tr>
<tr>
<td>LAEM</td>
<td>TGIF</td>
</tr>
<tr>
<td>Lilliput-AE</td>
<td>TinyJambu</td>
</tr>
<tr>
<td>Limdolen</td>
<td>Triad</td>
</tr>
<tr>
<td>LOTUS &amp; LOCUS</td>
<td>TRIFLE</td>
</tr>
<tr>
<td>mixFeed</td>
<td>WAGE</td>
</tr>
<tr>
<td>ORANGE</td>
<td>Xoodyak</td>
</tr>
<tr>
<td>Oribatida</td>
<td>Yarara &amp; Coral</td>
</tr>
</tbody>
</table>
Authenticated Encryption with Associated Data

- **AEAD** is a symmetric encryption scheme

![Diagram of AEAD process]

- PT: Plain Text
- CT: CipherText
- AD: Associated Data
- E: Encrypt
- D: Decrypt
- N: Nonce
- K: Key
Cryptographic Hash Functions

Cryptographic Hash functions convert a message into a unique, fixed-length digest

- Collision resistance
- Preimage resistance
- Second-preimage resistance

\[
\text{H(“NIST”) = FCE07FF980244E6D} \\
\text{H(“FIST”) = 70F44C69CA82041B} \\
\text{H(“National Institute of...”) = C034262E461C6474}
\]
Testing Approach

Design Tests Based on Cryptographic Properties

- Implementations should satisfy the algorithmic properties of AEAD and HASH

Tests

- Bit Exclusion
- Bit Contribution (3 variations)
- Buffer Check
- Ciphertext Length Check

Apply Tests to LWC Standardization Process Submissions

- All variants of reference implementations
Bit Exclusion

Motivation
Bits beyond the specified input message length should be ignored

Strategy
• Generate a plaintext message $m$ of length $n$
• Flip one bit of $m$ outside length $n$, call this $m'$
• Check $H(m) = H(m')$?
  • If no, then fail
Bit Exclusion

Bits beyond the specified input length should be ignored

<table>
<thead>
<tr>
<th>PT</th>
<th>1 1 0 1 0 1 ... 1 0 1 ... 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT length</td>
<td>1 1 0 1 0 1 ... 1 1 1 ... 1</td>
</tr>
<tr>
<td></td>
<td>1 1 0 1 0 1 ... 1 0 0 ... 1</td>
</tr>
<tr>
<td></td>
<td>1 1 0 1 0 1 ... 1 0 1 ... 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Hash</th>
<th>8C3C3453F0C5517453DB5C0230D0</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>8C3C3453F0C5517453DB5C0230D0</td>
</tr>
<tr>
<td></td>
<td>2C3C3453F1C5517453DB5C0240D0</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>8C3C3453F0C5517453DB5C0230D0</td>
</tr>
</tbody>
</table>

Passed: 8C3C3453F0C5517453DB5C0230D0
Failed: 2C3C3453F1C5517453DB5C0240D0
Bit Contribution for Plaintext

Motivation

Second-Preimage Resistance: given a message $m$ and hash function $H$, it should be difficult to find a $m' \neq m$ such that $H(m') = H(m)$

Strategy

• Generate a plaintext message $m$ of length $n$
• Flip one bit of $m$, call this $m'$
• Check $H(m) = H(m')$?
  • If yes, then fail
Bit Contribution for Plaintext

Second-Preimage Resistance

Given a message $m$ and hash function $H$, it should be difficult to find a $m' \neq m$ such that $H(m') = H(m)$
Bit Contribution for Nonce

Motivation

LWC requirements states, “AEAD algorithms are expected to maintain security as long as the nonce is unique (not repeated under the same key)”

Strategy

• Generate a random $PT$, $AD$, $N$, and $K$
• Process $PT$, $AD$, $N$, and $K$, yielding $CT$
• Flip one bit of $N$, call this $N'$
• Process $PT$, $AD$, $N'$, and $K$, yielding $CT'$
• XOR $CT$ and $CT'$, and add result to matrix

Really small or really large matrix values imply a failure
Bit Contribution for Key

Motivation

AEAD algorithms are expected to maintain security when the key is unique.

Strategy

• Generate a random $PT$, $AD$, $N$, and $K$
• Process $PT$, $AD$, $N$, and $K$, yielding $CT$
• Flip one bit of $K$, call this $K'$
• Process $PT$, $AD$, $N$, and $K'$, yielding $CT'$
• XOR $CT$ and $CT'$, and add result to matrix

$$
\begin{array}{ccccccc}
& CT_0 & CT_1 & CT_2 & CT_3 & CT_4 & \ldots & CT_c \\
K_0 & 5204 & 5102 & 4802 & 5219 & 4787 & \ldots & 5223 \\
K_1 & 4883 & 5209 & 4778 & 5247 & 4792 & \ldots & 5213 \\
K_2 & 5204 & 5209 & 4778 & 5183 & 5211 & \ldots & 4985 \\
K_3 & 5085 & 5201 & 5179 & 5183 & 5211 & \ldots & 5014 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
K_N & 5226 & 8406 & 4800 & 5214 & 7001 & \ldots & 4985 \\
\end{array}
$$

Really small or really large matrix values imply a failure.
Buffer Check (Decryption Failure Test)

**Motivation**

“Plaintext **should not** be returned by the decryption-verification process if the ciphertext is invalid.”

**Strategy**

- Generate a ciphertext $CT_{PT, AD, N, K}$
- Flip one bit of $CT_{PT, AD, N, K}$
- Invoke *decrypt* function
- Check the buffer where plaintext was to be stored
  - If the buffer has a consecutive 10-byte match to $PT$, then fail
“Plaintext **shall not** be returned by the decryption-verification process if the ciphertext is invalid.”
Ciphertext Length Check

Motivation
Algorithms must make sure that the ciphertext is at most $\text{CRYPTO\_ABYTES}$ longer than the plaintext.

Strategy
• Generate a random $PT$ of length $n$, $AD$, $N$, and $K$
• Process (encrypt) $PT$, $AD$, $N$, and $K$, yielding $CT$
• Make sure $|CT| \geq n$ and $|CT| \leq n + \text{CRYPTO\_ABYTES}$
  • If no, then fail
Experimentation

**AEAD**
56 algorithms, 
157 reference implementations 
(All variants),
- Bit Contribution for Plaintext
- Bit Contribution for Nonce
- Bit Contribution for Key
- Bit Exclusion
- Buffer Check
- Ciphertext Length Check

**HASH**
22 algorithms, 
39 reference implementations, 
(All variants)
- Bit Contribution for Plaintext
- Bit Exclusion
Results – HASH

No failures were discovered for the hash function implementations

• Does not guarantee there are no bugs
Results – AEAD

Bit Exclusion: 0% failed

Bit Contribution for Plaintext: 0% failed

Bit Contribution Nonce: 9.55% failed

Bit Contribution Key: 7.01% failed

Buffer Check (Decryption Failure): 64.97% failed

Ciphertext Length Check: 1.91% failed
AEAD Results—Bit Contribution for Nonce

85.35% passed
5.10% indeterminate
9.55% failed
  • 8/15 failed implementations are definitive failures
    • bleep64
    • lotus
    • orange
    • qameleon128128128v1
    • qameleon12812896v1
    • qameleon12812864v1
    • quartet
    • wage
Implementation *lotus* failed the Bit Contribution for Nonce test

- Bits 64 to 127 do not affect the ciphertext produced
Implementation *lotus* failed the Bit Contribution for Nonce test

- Bits 64 to 127 do not affect the ciphertext produced

**ltwegift64lotus/encrypt.c, corrected**

```c
void init(u8 *nonced_key, u8 *nonced_mask, const u8 *key, const u8 *nonce) {
    u8 twk;
    u8 zero[CRYPTO_BLOCKBYTES] = { 0 };
    u8 enc_zero[CRYPTO_BLOCKBYTES];

    // compute K_N = K + N
    memcpy(nonced_key, key, CRYPTO_KEYBYTES);
    xor_bytes(nonced_key, nonce, CRYPTO_NPUBBYTES);
```

AEAD Results—Buffer Check Test

30.57% passed
4.46% indeterminate
64.97% failed

Some implementations acknowledge that they do not clear the buffer

Possible Solution: Use an additional temporary buffer.
Conclusion

Metamorphic tests based on cryptographic properties is effective

- We have seen many test failures and found several source code bugs

Future Work

- Test the optimized implementations
- Develop a generic testing approach for cryptographic algorithms