From Dragondoom to Dragonstar: Side-channel Attacks and Formally Verified Implementation of WPA3 Dragonfly Handshake

Daniel De Almeida Braga¹, Natalia Kulatova²,³, Mohamed Sabt¹, Pierre-Alain Fouque¹, Karthikeyan Bhargavan³

EuroS&P - July 5ᵗʰ, 2023

University of Rennes, CNRS, IRISA¹
Mozilla²
INRIA, Paris³
Toward Secure Wi-Fi Protocols...

- **WEP** (1999)
- **WPA** (2003, 2004)
- **WPA2** (2018)

Client → Access Point
Toward Secure Wi-Fi Protocols...

- Client
- Access Point

- WEP
  - 1999

- WPA
  - 2003

- WPA2
  - 2018

- Offline dictionary attack
Toward Secure Wi-Fi Protocols...

- WEP: 1999
- WPA: 2003
- WPA2: 2004
- WPA3: 2018

- Offline dictionary attack
- KRACK attack
Toward Secure Wi-Fi Protocols...

+ More secure
+ Based on a PAKE (Dragonfly\textsuperscript{1})

\textsuperscript{1} D. Harkins. Dragonfly Key Exchange. RFC 7664. 2015
$r_A, m_A = \text{rand}(2, \ldots, q-1)$

$s_A = r_A + m_A \mod q$

$id_A$

$id_B$

$r_B, m_B = \text{rand}(2, \ldots, q-1)$

$s_B = r_B + m_B \mod q$
$r_A, m_A = \text{rand}(2, \ldots, q-1)$

$s_A = r_A + m_A \mod q$

$P = \text{pwd} \_\text{conv}(\text{pwd}, id_A, id_B)$

$Q_A = -m_A P$

$\text{Commit}(s_A, Q_A)$

$\text{Commit}(s_B, Q_B)$
Dragonfly / SAE - A Balanced PAKE

\[ r_A, m_A = \text{rand}(2, \ldots, q-1) \]
\[ s_A = r_A + m_A \mod q \]

\[ P = \text{pwd_conv}(\text{pwd}, id_A, id_B) \]
\[ Q_A = -m_A P \]

\[ kck || mk = \text{compute_keys}(r_A, P, s_B, Q_B) \]

\[ r_B, m_B = \text{rand}(2, \ldots, q-1) \]
\[ s_B = r_B + m_B \mod q \]

\[ P = \text{pwd_conv}(\text{pwd}, id_A, id_B) \]
\[ Q_B = -m_B P \]

\[ kck || mk = \text{compute_keys}(r_B, P, s_A, Q_A) \]
Alice

\[ r_A, m_A = \text{rand}(2, \ldots, q-1) \]
\[ s_A = r_A + m_A \mod q \]

\[ P = \text{pwd_conv}(\text{pwd}, \text{id}_A, \text{id}_B) \]
\[ Q_A = -m_A P \]

Bob

\[ r_B, m_B = \text{rand}(2, \ldots, q-1) \]
\[ s_B = r_B + m_B \mod q \]

\[ P = \text{pwd_conv}(\text{pwd}, \text{id}_A, \text{id}_B) \]
\[ Q_B = -m_B P \]

kck || mk = compute_keys(r_A, P, s_B, Q_B)

kck || mk = compute_keys(r_B, P, s_A, Q_A)

Verify \text{c}_B

Verify \text{c}_A
... But Still not Bulletproof

2018 2019 2020 today

Dragonblood\textsuperscript{1}

attacks

- Weird choice of password conversion method
  - Probabilistic
  - Difficult to implement securely
- Concerned were raised... and confirmed

\textsuperscript{1} M. Vanhoef and E. Ronen. Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd. In IEEE S&P'20
... But Still not Bulletproof

Dragonblood is Still Leaking

Weird choice of password conversion method
- Probabilistic
- Difficult to implement securely
- Concerned were raised... and confirmed

2 D. De Almeida Braga et al Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild. In ACSAC ’20
... But Still not Bulletproof

Dragonblood is Still Leaking\(^2\)

- Better password conversion (SSWU)
  - Deterministic
  - Straightforward constant-time implementation
- △ Not backward compatible

---

2. D. De Almeida Braga et al *Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild*. In ACSAC ’20
Attack Workflow

Access Point

WPA3 (SAE)

Victim
Attack Workflow

- Rogue AP
- WPA3 (SAE)
- Victim
- Spy process
- Offline dictionary
- Attack
- Remaining passwords
- 5/12
Spawning/Data Acquisition

- Implementation specific
- Usually noisy measurement

**Comparison metric:** Signal to Noise ratio
Attack Workflow

Leaked information

Trace parsing
Offline Dictionary Attack

Remaining passwords
Offline Dictionary Attack

\[ H(\text{secret}) = 10 \ldots \]
## Offline Dictionary Attack

<table>
<thead>
<tr>
<th>$x$</th>
<th>$H(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>secret</code></td>
<td>10..</td>
</tr>
<tr>
<td>$pwd_1$</td>
<td></td>
</tr>
<tr>
<td>$pwd_2$</td>
<td></td>
</tr>
<tr>
<td>$pwd_3$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$pwd_n$</td>
<td></td>
</tr>
</tbody>
</table>

Remaining passwords
### Offline Dictionary Attack

<table>
<thead>
<tr>
<th>x</th>
<th>H(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>10..</td>
</tr>
<tr>
<td>pwd₁</td>
<td>01..</td>
</tr>
<tr>
<td>pwd₂</td>
<td>10..</td>
</tr>
<tr>
<td>pwd₃</td>
<td>11..</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>pwdₙ</td>
<td>10..</td>
</tr>
</tbody>
</table>

Remaining passwords
## Offline Dictionary Attack

<table>
<thead>
<tr>
<th>x</th>
<th>H(x)</th>
</tr>
</thead>
<tbody>
<tr>
<td>secret</td>
<td>10..</td>
</tr>
<tr>
<td>pwd₁</td>
<td>01..</td>
</tr>
<tr>
<td>pwd₂</td>
<td>10..</td>
</tr>
<tr>
<td>pwd₃</td>
<td>11..</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>pwdₙ</td>
<td>10..</td>
</tr>
</tbody>
</table>

Remaining passwords:
## Offline Dictionary Attack

|    | $H(x || pub_1)$ | $H(x || pub_2)$ |
|----|----------------|----------------|
| secret | 10.. | 00.. |
| $pwd_1$ | 01.. | X |
| $pwd_2$ | 10.. | 00.. |
| $pwd_3$ | 11.. | X |
| ... | ... | ... |
| $pwd_n$ | 10.. | 11.. |

Remaining passwords
## Offline Dictionary Attack

| X   | $H(x || pub_1)$ | $H(x || pub_2)$ |
|-----|-----------------|-----------------|
| secret | 10..            | 00..            |
| pwd_1   | 01..            | X               |
| pwd_2   | 10..            | 00..            |
| pwd_3   | 11..            | X               |
| ...     | ...             | ...             |
| pwd_n   | 10..            | 11..            |

Remaining passwords
Attack Workflow

1. Rogue AP
2. WPA3 (SAE)
3. Victim
4. Spy process
5. Leak information
6. Offline dictionary attack
7. Remaining passwords
8. Trace parsing
9. Passwords
We mostly analyzed Wi-Fi daemons...

... what about their dependencies, like crypto libraries?
def set_compressed_point(x, fmt, ec):
    • Branching on the compression format
    • Affects SAE (legacy version)
    • 1-bit leakage
    • Narrow scope outside of Dragonfly
def set_compressed_point(x, fmt, ec)

• Branching on the compression format
• Affects SAE (legacy version)
• 1-bit leakage
• Narrow scope outside of Dragonfly

def bin2bn(buf, buf_length)

• Skipping leading 0 bytes
• Affects both SAE and SAE-PT
• 8-bit leakage with proba 1/256
• Wide scope (targets utility function)
def set_compressed_point(x, fmt, ec)
    • Branching on the compression format
    • Affects SAE (legacy version)
    • 1-bit leakage
    • Narrow scope outside of Dragonfly

def bin2bn(buf, buf_length)
    • Skipping leading 0 bytes
    • Affects both SAE and SAE-PT
    • 8-bit leakage with proba $1/256$
    • Wide scope (targets utility function)

Affected projects:
    • hostap/wpa_supplicant with OpenSSL/WolfSSL
    • iwd with ell
    • FreeRadius with OpenSSL
”Obviously” Vulnerable, yet Difficult to Exploit

- Very few conditional instructions (one cache line or less)
- Many false positives with ”vanilla” Flush+Reload
- Using existing attack to create a new distinguisher

Abuse prefetching behaviors to create a new distinguisher!
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P

Very accurate distinguisher, with a better spatial resolution!
Sustainable patch for hostap

- Cryptographic libraries refused to patch
- Many other potential vulnerabilities (≈ 400)

Shall we replace them?

---

1 J-K. Zinzindohoué et al. HACL*: A Verified Modern Cryptographic Library. In CCS’17
Sustainable patch for hostap

- Cryptographic libraries refused to patch
- Many other potential vulnerabilities ($\approx 400$)

**Shall we replace them?**

HaCl*: A Formally Verified Cryptographic Library

- Memory-safety
- Functional correctness
- Secret independence

---

$^1$ J-K. Zinzindohouë et al. *HaCL*: A Verified Modern Cryptographic Library. In CCS'17
Thank you Alexandre Sanchez for helping with the patch integration
Fixing hostap

Thank you Alexandre Sanchez for helping with the patch integration
Fixing hostap

Thank you Alexandre Sanchez for helping with the patch integration

crypto/
...
crypto.h
crypto_mbedtls.c
crypto_openssl.c
crypto_wolfssl.c
...

1 Thank you Alexandre Sanchez for helping with the patch integration
Thank you Alexandre Sanchez for helping with the patch integration
A New Attack

- Dictionary attack (SAE/SAE-PT)
  - Improved signal-to-noise ratio!
  - First side-channel in SAE-PT (supposed to be ct by design)
- New generic gadget
  - Potential impact on many low-level arithmetic functions

A Better Defense

- 3 security patches (hostap, iwd, FreeRadius)
- Formally verified crypto implementation (HaCl*)
- Benefit from HaCl*'s team support

Material available at
- https://gitlab.inria.fr/ddealmei/artifact_dragondoom
- https://gitlab.inria.fr/ddealmei/artifact_dragonstar
Impact

A New Attack

• Dictionary attack (SAE/SAE-PT)
  • Improved signal-to-noise ratio!
  • First side-channel in SAE-PT (supposed to be ct by design)
• New generic gadget
  • Potential impact on many low-level arithmetic functions

A Better Defense

• 3 Security patches (hostap, iwd, FreeRadius)
• Formally verified crypto implementation (HaCl*)
• Benefit from HaCl*’s team support

Material available at
• https://gitlab.inria.fr/ddealmei/artifact_dragondoom
• https://gitlab.inria.fr/ddealmei/artifact_dragonstar
Impact

A New Attack

• Dictionary attack (SAE/SAE-PT)
  • Improved signal-to-noise ratio!
  • First side-channel in SAE-PT (supposed to be ct by design)

• New generic gadget
  • Potential impact on many low-level arithmetic functions

A Better Defense

• 3 Security patches (hostap, iwd, FreeRadius)
• Formally verified crypto implementation (HaCl*)
• Benefit from HaCl*’s team support

Material available at

• https://gitlab.inria.fr/ddealmei/artifact_dragondoom
• https://gitlab.inria.fr/ddealmei/artifact_dragonstar
Appendix
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y == fmt % 2:
        y = ec.p - y
    P = init_point(x, y, ec)
    [...]
    return P

CPU cache

nb hits: 0

cond (A)
probe (B)

Victim

flush (PDA)

cond (A)
probe (B)

Attacker

prefetch
prefetch

cond (A)
probe (B)
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y = fmt mod 2:
        y = ec.p - y
    P = init_point(x, y, ec)
    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y = fmt mod 2:
        y = ec.p - y
    P = init_point(x, y, ec)
[...]
    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y = fmt mod 2:
        y = ec.p - y
    P = init_point(x, y, ec)
    return P

nb hits: 0
flush (PDA)
flush (F+R)
reload (hit)
reload (miss)
prefetch
prefetch
CPU cache
cond (A)
probe (B)
Victim
Attacker

nb hits: 0
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P

nb hits: 1
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y = fmt mod 2:
        y = ec.p - y
    P = init_point(x, y, ec)
    return P

nb hits: 2
flush (PDA)
reload (hit)
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)
    if y = fmt mod 2:
        y = ec.p - y
    P = init_point(x, y, ec)
    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y = fmt mod 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

    if y == fmt % 2:
        y = ec.p - y

    P = init_point(x, y, ec)

    return P

Very accurate distinguisher, with a better spatial resolution!