Dragonblood: A Security Analysis of WPA3’s SAE Handshake

Mathy Vanhoef and Eyal Ronen

WAC Workshop @ CRYPTO, Santa Barbara, 17 August 2019.
Background: Dragonfly in WPA3 and EAP-pwd

= Password Authenticated Key Exchange (PAKE)

- Provide mutual authentication
- Negotiate session key
- Forward secrecy & prevent offline dictionary attacks
- Protect against server compromise
Dragonfly
Dragonfly

Convert password to group element $P$

Convert password to group element $P$
Dragonfly

Convert password to group element $P$

Convert password to group element $P$

Commit phase
Dragonfly

- Convert password to group element $P$
- Convert password to group element $P$
- Commit phase
- Negotiate shared key
Dragonfly

Convert password to group element $P$

Commit phase

Negotiate shared key

Confirm phase

Convert password to group element $P$
Dragonfly

Convert password to group element $P$

Commit phase

Negotiate shared key

Confirm phase

Confirm peer negotiated same key
Dragonfly

Convert password to group element $P$

Convert password to group element $P$

Supports two crypto groups:
1. MODP groups
2. Elliptic curves

Confirm phase
Supports two crypto groups:
1. MODP groups
2. Elliptic curves

Confirm phase
What are MODP groups?

Operations performed on integers $x$ where:

- $x < p$ with $p$ a prime
- $x^q \mod p = 1$ must hold
- $q = \#\text{elements in the group}$

→ All operations are MODulo the Prime (≡ MODP)
Convert password to MODP element

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue

P = value^{(p-1)/q}

return P
Convert password to MODP element

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue

P = value^{(p-1)/q}
```

Convert value to a MODP element
Convert password to MODP element

for (counter = 1; counter < 256; counter++)

    value = hash(pw, counter, addr1, addr2)

    if value >= p: continue

    P = value^{(p-1)/q}

return P

Problem for groups 22-24: high chance that value >= p
Convert password to MODP element

for (counter = 1; counter < 256; counter++)

    value = hash(pw, counter, addr1, addr2)

    if value >= p: ???

    P = value^{(p-1)/q}

return P
Convert password to MODP element

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    \[ P = \frac{value^{(p-1)/q}}{q} \]
return P
Convert password to MODP element

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value ≠ p: continue
P = \frac{value^2 - 1}{q}
return P

#iterations depends on password
Convert password to MODP element

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value < p: continue

P = value \cdot (p-1)/q

return P

#iterations depends on password

No timing leak countermeasures, despite warnings by IETF & CFRG!
IETF mailing list in 2010

“[..] susceptible to side channel (timing) attacks and may leak the shared password. I'd therefore recommend [excluding the MAC addresses].”

“not so sure how important that is [..] doesn't leak the shared password [..] not a trivial attack.”
Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
</tr>
</tbody>
</table>
Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
</tr>
<tr>
<td>Password 3</td>
<td></td>
</tr>
</tbody>
</table>
Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
</tr>
<tr>
<td>Password 3</td>
<td></td>
</tr>
</tbody>
</table>
What information is leaked?

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value != p: continue
    P = value \cdot (p-1)/q

**Spoof client address** to obtain different execution & leak new data
Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
<th>addrB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
<th>addrB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
<th>addrB</th>
<th>addrC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
<th>addrB</th>
<th>addrC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Password 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need ~17 addresses to determine password in RockYou (~$10^7$) dump
Leaked information: #iterations needed

<table>
<thead>
<tr>
<th>Client address</th>
<th>addrA</th>
<th>addrB</th>
<th>addrC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Measured</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Forms a signature of the password

Need ~17 addresses to determine password in RockYou (~$10^7$) dump
Raspberry Pi 1 B+: differences are measurable

Hostap AP: ~75 measurements / address
What about elliptic curves?

Operations performed on points \((x, y)\) where:

\[ x < p \text{ and } y < p \text{ with } p \text{ a prime} \]
\[ y^2 = x^3 + ax + b \mod p \text{ must hold} \]

→ Need to **convert password to point** \((x,y)\) on the curve
Hash-to-curve: EAP-pwd

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, \sqrt{x^3 + ax + b})

EAP-pwd: similar timing leak with elliptic curves
Hash-to-curve: WPA3

```python
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue

if square_root_exists(x) and not P:
    P = (x, \sqrt{x^3 + ax + b})
    pw = rand()

return P
```

**WPA3**: always do 40 loops & return first P
Hash-to-curve: WPA3

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue

    if square_root_exists(x) and not P:
        P = (x, \sqrt{x^3 + ax + b})

    pw = rand()

return P

Extra iterations based on random password
Hash-to-curve: WPA3

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue

    if square_root_exists(x) and not P:
        \[
P = (x, \sqrt{x^3 + ax + b})\]
        pw = rand()

return P

Problem for Bainpool curves: high chance that x >= p
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
if x >= p: continue
if square_root_exists(x) and not P:
    P = (x, \sqrt{x^3 + ax + b})
pw = rand()
return P
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, \sqrt{x^3 + ax + b})
    pw = rand()
return P

#Times skipped depends on password
Hash-to-curve: WPA3

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, \sqrt{x^3 + ax + b})
    pw = rand()
return P

#Times skipped depends on password & random password in extra iterations
Hash-to-curve: WPA3

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)

if x >= p: continue

if square_root_exists(x) and not P:
    P = (x, \sqrt{x^3 + ax + b})

pw = rand()

return P

Variance ~ when password element was found
Hash-to-curve: WPA3

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, \sqrt{x^3 + ax + b})

pw = rand()

return P

Variance ~ when password element was found
Average ~ when found & #iterations code skipped
Raspberry Pi 1 B+

Hostap (WPA3):

~300 measurements / address
Cache Attacks
NIST Elliptic Curves

for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt[3]{x^3 + ax + b}$)
        pw = rand()
return P

NIST curves: use Flush+Reload to detect when code is executed
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, \( \sqrt{x^3 + ax + b} \))
    pw = rand()
return P

NIST curves: use Flush+Reload to detect when code is executed
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, \sqrt{x^3 + ax + b})
        pw = rand()
return P

Monitor using Flush+Reload to know in which iteration we are

Brainpool curves: use Flush+Reload to detect when code is executed
Cache-attacks in practice

Requires powerful adversary:
› Run unprivileged code on victim’s machine
› Act as malicious client/AP within range of victim

Abuse leaked info to recover the password
› Spoof various client addresses similar to timing attack
› Use resulting password signature in dictionary attack
Attack Optimizations

Timing & cache attack result in password signature

› Both use the same brute-force algorithm

Improve performance using GPU code:

› We can brute-force $10^{10}$ passwords for $1$
› MODP / Brainpool: all 8 symbols costs $67$
› NIST curves: all 8 symbols costs $14k$
Detailed Analysis: See Paper

- Estimate required #(spoofed MAC addresses):

\[
\ell = \sum_{i=1}^{\infty} i \cdot \Pr[Z_d = i] = \sum_{i=1}^{\infty} i \cdot (\Pr[Z_d \leq i] - \Pr[Z_d \leq i - 1])
\]

- Offline brute-force cost:

\[
\sum_{n=1}^{k'} n \cdot p_e^{n-1} \cdot (1 - p_e) + p_e^{k'} \cdot \sum_{n=1}^{\infty} (k' + n) \cdot (1 - p_e)^{n-1} \cdot p_e
\]
Implementation Inspection
Invalid Curve Attack

Commit \( x', y' \)

Point isn’t on curve
Invalid Curve Attack

Point isn’t on curve

Commit $x', y'$

Negotiated key is predictable

Commit reply
Invalid Curve Attack

Point isn’t on curve

Commit $x', y'$

Negotiated key is predictable

Commit reply

Guess key and send confirm

Confirm phase
Invalid Curve Attack

Point isn’t on curve

Commit $x’, y’$

Negotiated key is predictable

Confirm phase

Bypasses authentication

- EAP-pwd: all implementations affected
- WPA3: only iwd is vulnerable
Reflection Attack: EAP-pwd example

Commit(x, y)

Reflect frame

Commit(x, y)
Reflection Attack: EAP-pwd example

association

Commit(x, y)

Reflect frame

Commit(x, y)

Confirm

Reflect frame

Confirm
Reflection Attack: EAP-pwd example

Authenticate as victim

- EAP-pwd: all servers are vulnerable
- WPA3: old wpa_supplicants affected

Reflect frame
Other Implementation Vulnerabilities

Bad randomness:
- Can recover password element P
- Aruba’s EAP-pwd client for Windows is affected
- With WPA2 bad randomness has lower impact!

Side-channels:
- FreeRADIUS aborts if >10 iterations are needed
- Aruba’s EAP-pwd aborts if >30 are needed
- Can use leaked info to recover password
Wi-Fi Specific Attacks
Denial-of-Service Attack

Convert password to group element $P$

AP converts password to EC point when client connects

Conversion is computationally expensive (40 iterations)
Forging 8 connections/sec saturates AP’s CPU
Downgrade Against WPA3-Transition

Transition mode: **WPA2/3 use the same password**
  - WPA2’s handshake detects downgrades → forward secrecy
  - Performing partial WPA2 handshake → **dictionary attacks**

Solution is to **remember which networks support WPA3**
  - Similar to trust on first use of SSH & HSTS
  - Implemented by Pixel 3 and Linux’s NetworkManager
Crypto Group Downgrade

Handshake can be performed with multiple curves
› Initiator proposes curve & responder accepts/rejects
› **Spoof reject messages to downgrade** used curve

= **design flaw**, all client & AP implementations vulnerable
Implementation-specific downgrades

› Clone WPA3-only network & advertise it only supports WPA2
› Galaxy S10 & iwd connected using the WPA3-only password
› Results in trivial dictionary attack
Disclosure
Disclosure process

Notified parties early with hope to influence WPA3

› Some initially sceptic, considered it implementation flaws
› Group downgrade: “was known, but forgot to warn about it”

Reaction of the Wi-Fi Alliance

› Privately created backwards-compatible security guidelines
› 2\textsuperscript{nd} disclosure round to address Brainpool side-channels
Fundamental issue still unsolved

› On lightweight devices, doing 40 iterations is too costly
› Even powerful devices are at risk: handshake might be offloaded to the lightweight Wi-Fi chip itself

Wi-Fi standard now being updated

› Prevent crypto group downgrade attack
› Allow offline computation of password element
Additional updates to Wi-Fi standard

MODP crypto groups:
› Restrict usage of weak MODP groups
› Constant-time algo (modulo instead of iterations)

Elliptic curve groups:
› Restrict usage of weak elliptic curves
› Constant-time algo (simplified SWU)
Updates aren’t backwards-compatible

Might lead to WPA3.1?
› Not yet clear how this will be handled
› Risk of downgrade attacks to original WPA3

Will people be able to easily attack WPA3?
› No, WPA3 > WPA2 even with its flaws
› Timing leaks: non-trival to determine if vulnerable
Conclusion

› WPA3 vulnerable to side-channels
› Countermeasures are costly
› **Standard now being updated**
› Issues could have been avoided!

https://wpa3.mathyvanhoef.com
Thank you! Questions?

› WPA3 vulnerable to side-channels
› Countermeasures are costly
› **Standard now being updated**
› Issues could have been avoided!

https://wpa3.mathyvanhoef.com