Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd

Mathy Vanhoef and Eyal Ronen

ANRW. Montreal, Canada, 22 July 2019.
Background: Dragonfly in WPA3

= Password Authenticated Key Exchange (PAKE)

- Provide mutual authentication
- Negotiate session key
- Forward secrecy & prevent offline dictionary attacks
- Protect against server compromise
Convert password to elliptic curve point $P$  

Commit phase

Convert password to elliptic curve point $P$

Confirm phase
With MODP groups: hash-to-group

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

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

    if P > 1: return P
With MODP groups: hash-to-group

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

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

if P > 1 return P

In practice always true
```
With MODP groups: hash-to-group

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

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

if value >= p: continue

P = \frac{value(p-1)}{q}

if P > 1
    return P

In practice always true
With MODP groups: hash-to-group

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value \((p^{-1})/q\)
    if P > 1: return P
With MODP groups: hash-to-group

for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^{(p-1)/q}
    if P > 1: return P
With MODP groups: hash-to-group

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

No timing leak countermeasures despite warnings by IETF & CFRG!
With MODP groups: hash-to-group

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

WPA3: spoof client address to obtain different executions

No timing leak countermeasures despite warnings by IETF & CFRG!
Raspberry Pi 1 B+: differences are measurable

![Graph showing response time distribution for different iterations](image)
Raspberry Pi 1 B+: differences are measurable

Hostap (WPA3): $\sim75$ measurements / address

iwd (EAP-pwd): $\sim30$ measurements / address
# 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>
### 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>
</tbody>
</table>

forms a signature of the password

Need \(~17\) addresses to test \(~10^7\) passwords
What about elliptic curves?

Hash-to-group with elliptic curves also affected?
› By default Dragonfly uses NIST curves
› Timing leaks for NIST curves are mitigated

Dragonfly also supports Brainpool curves
› After our initial disclosure, the Wi-Fi Alliace private created guidelines that mention these are secure to use
› Bad news: Brainpool curves in Dragonfly are insecure
Hash-to-curve

\[
\text{for (counter = 1; counter < k or not x; counter++)}
\]
\[
\begin{align*}
\text{value} &= \text{hash}(\text{pw}, \text{counter}, \text{addr1}, \text{addr2}) \\
\text{if value} &\geq \text{p: continue} \\
\text{y}_\text{sqr} &= \text{value}^3 + a \times \text{value} + b \\
\text{if is_quadratic_residue(y}_\text{sqr}) &\text{and not x:}
\end{align*}
\]
\[
\begin{align*}
\text{x} &= \text{value} \\
\text{pw} &= \text{random}() \\
\text{y} &= \text{sqrt}(\text{x}^3 + a \times \text{x} + b) \\
\text{return } (\text{x}, \text{y})
\end{align*}
\]
Hash-to-curve

```python
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
    pw = random()

y = sqrt(x^3 + a * x + b)
return (x, y)
```

Problem: no solution for y
Hash-to-curve

for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
    y = sqrt(x^3 + a * x + b)
return (x, y)
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
    pw = random()

y = sqrt(x^3 + a * x + b)
return (x, y)
Hash-to-curve

for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
    y = sqrt(x^3 + a * x + b)
return (x, y)

Problem: different passwords have different execution time
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
Hash-to-curve

for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue

    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()

    y = sqrt(x^3 + a * x + b)
return (x, y)

In case quadratic test is not constant time
for (counter = 1; counter < k or not x; counter++)
  value = hash(pw, counter, addr1, addr2)
  if value >= p: continue
  y_sqr = value^3 + a * value + b
  if is_quadratic_residue(y_sqr) and not x:
    x = value
    pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
    y = sqrt(x^3 + a * x + b)
return (x, y)

May be true for Brainpool curves!
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
    y = sqrt(x^3 + a * x + b)
return (x, y)
Hash-to-curve

for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
    pw = random()

A random #(extra iterations) have a too big hash output

May be true for Brainpool curves!

Quadratic test may be skipped

return (x, y)
Influence of extra iterations

Execution 1
## Influence of extra iterations

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Execution 2</th>
<th>Execution 3</th>
<th>Execution 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

33
Influence of extra iterations

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Execution 2</th>
<th>Execution 3</th>
<th>Execution 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Influence of extra iterations

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Execution 2</th>
<th>Execution 3</th>
<th>Execution 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Variance ~ when password element was found
Influence of extra iterations

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Execution 2</th>
<th>Execution 3</th>
<th>Execution 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Variance** ~ when password element was found
- **Average** ~ when found and #iterations with big hash
Influence of extra iterations

<table>
<thead>
<tr>
<th>Execution 1</th>
<th>Execution 2</th>
<th>Execution 3</th>
<th>Execution 4</th>
</tr>
</thead>
</table>

- Variance ~ when password element was found
- Average ~ when found and #iterations with big hash

→ Again forms a signature of the password
Raspberry Pi 1 B+
Raspberry Pi 1 B+

Hostap (WPA3):
~300 measurements / address
Cache Attacks
Hash-to-curve: Quadratic Residue

for (counter = 1; counter < k or not x; counter++)
value = hash(pw, counter, addr1, addr2)
if value >= p: continue
y_sqr = value^3 + a * value + b
if is_quadratic_residue(y_sqr) and not x:
    x = value

NIST curves: use Flush+Reload to detect if code is executed in 1\textsuperscript{st} iteration
Hash-to-curve: Quadratic

for (counter = 1; counter < k or not x; counter++)
value = $\text{hash}(pw, \text{counter}, \text{addr1}, \text{addr2})$
if value >= p: continue
$y_{\text{sqr}} = \text{value}^3 + a \times \text{value} + b$
if is_quadratic_residue($y_{\text{sqr}}$) and not x:
    $x = \text{value}$

NIST curves: use Flush+Reload to detect if code is executed in 1st iteration

$y = \sqrt{x^3 + a \times x + b}$
return $(x, y)$
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
    pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
There’s a lot more!

Implementation-specific vulnerabilities
› Invalid curve attacks, reflection attacks, **bad randomness**

Wi-Fi specific attacks
› Downgrades to WPA2 & denial-of-service

Practical impact
› Brute-force attacks on GPUs: $1 for RockYou database
› 802.11 being updated to use **Shallue-Woestijne-Ulas**
Thank you! Questions?

Lessons learned:
› Must be constant-time and efficient
› Allow offline computation of P
› Discuss impact of bad randomness
› Limit number of parameters (e.g. curves)
› Dragonfly is hard to implement securely

https://wpa3.mathyvanhoef.com