Dragonblood: A Security Analysis of WPA3's SAE Handshake [1]

Nicolas Dutly

23.03.2020

[1] Based on Vanhoef and Ronen

Outline

IEEE 802.11 history

WEP

- relies on the RC4 cipher
- 104 bit key
- FMS attack
- passive key recovery

WPA

- Temporal Key Integrity Protocol (TKIP)
- Key mixing function
- 64 bit MAC
- still relies on RC4

WPA2

- Mandatory support for AES-CCMP
- Four way handshake
- WPA2-PSK vulnerable to offline bruteforce attacks
- KRACK attacks (Vanhoef and Piessens)

WPA3: The sucessor of WPA2

WPA3 Wi Fi ALLIANCE

Protection against offline bruteforce attacks

WPA2-PSK remplaced by WPA3-SAE (variant of the *Dragonfly* protocol).

- 192 bit security for entrepise networks
- Opportunistic wireless encryption (OWE) for open networks
- Modern primitive support (AES-GCM, EC support)

- Simplified setup for display-less devices

SAE: Simultanous Authentication of Equals

- Password authenticated key exchange (PAKE)
- Supports ECP and MODP groups
- Standart run consists of 3 phases:
  1. Password derivation
  2. Commit phase
  3. Confirm phase

We will now go through the commit and confirm phases.

SAE handshake

Assume both parties used the shared password p_s to derive a point P on an agreed EC.

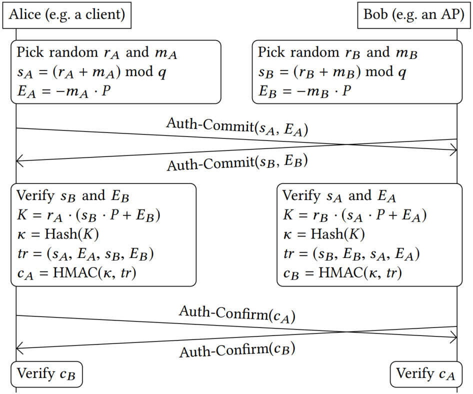


Figure: SAE handshake (taken from Vanhoef and Ronen)

Dragonblood

Downgrade attacks:

- Downgrade of the group parameters used in SAE
- Downgrade attack agains WPA3 transition mode

Weaknesses in the dragonfly handshake:

- Timing-based side channel attack
- Cache-based side channel attack
- Denial of Service attack

Password derivation: Hash to MODP

```
hash_to_group(password, mac1, mac2):
    for counter in range(1,256):
        seed = Hash(mac1,mac2,password,counter)
        value = KDF(seed, label, p)
        if value >= p: continue
        Elem = math.pow(value, (p-1)/q) mod p
        if Elem > 1: return Elem
```

Figure: (simplified) hash2modp function

Password derivation: Hash to MODP

```
hash_to_group(password, mac1, mac2):
    for counter in range(1,256):
        seed = Hash(mac1,mac2,password,counter)
        value = KDF(seed, label, p)
        if value >= p: continue
        Elem = math.pow(value, (p-1)/q) mod p
        if Elem > 1: return Elem
```

Figure: (simplified) hash2modp function

- Number of iteration depends on password !

Password derivation: Hash to MODP

```
hash_to_group(password, mac1, mac2):
    for counter in range(1,256):
        seed = Hash(mac1,mac2,password,counter)
        value = KDF(seed, label, p)
        if value >= p: continue
        Elem = math.pow(value, (p-1)/q) mod p
        if Elem > 1: return Elem
```

Figure: (simplified) hash2modp function

- Number of iteration depends on password !
- Number of iterations also depends on mac values!

More subtle timing leaks

The line

```
if value >= p: continue
```

Also induces a timing leak. Given a MODP group we can compute the probability that the KDF returns an invalid output.

- We will come back to this in a few slides. This leak is more pronounced for certain elliptic curves.

How can we deduce the password?

Imagine we have a list of passwords:

- `pass123` Simulated number of iterations: 4 | Actual iterations: 3
- `password` Simulated number of iterations: 2 | Actual iterations: 5
- `r0ckstar` Simulated number of iterations: 1 | Actual iterations: 1
- `asdfg` Simulated number of iterations: 2 | Actual iterations: 2
- ...

What is the valid password ?

How can we deduce the password?

Imagine we have a list of passwords:

- ~~`pass123`~~ Simulated number of iterations: 4 | Actual iterations: 3
- ~~`password`~~ Simulated number of iterations: 2 | Actual iterations: 5
- `r0ckstar` Simulated number of iterations: 1 | Actual iterations: 1
- `asdfg` Simulated number of iterations: 2 | Actual iterations: 2
- ...

Do we have any other information which might help us exclude further passwords ?

Additional information: Spoofing MAC addresses

```
seed = Hash(mac1,mac2,password,counter)
value = KDF(seed, label)
```

We can spoof MAC addresses

- r0ckstar — sim. iter with MAC M_1: 1 | Actual iterations: 1
- ~~r0ckstar~~ — sim. iter with MAC M_2: 1 | Actual iterations: 2
- asdfg — Simulated number of iterations: 2 | Actual iterations: 2
- ...

What about EC based crypto ?

Hash2Curve

```
hash_to_curve(password, mac1, mac2):
    k = 40, found = False
    while count < k:
        count++
        seed = Hash(mac1,mac2,password,count)
        value = KDF(seed, label, p)
        if value >= p: continue
        if quad_res(value^3 + a * value + b, p):
            if not found:
                x,found = value, True
                password = rand()
    y = sqrt(x^3+a*x+b) mod p
    return (x,y)
```

Figure: (Simpified) Hash2Curve pseudocode

What are the key differences compared to `hash2modp` *?*

Quardratic residues and blinding

We say that $x^3 + ax + b$ is a quadratic residue $\mod p$ if $\exists e$ s.t

$$x^3 + ax + b \equiv e^2 \mod p$$

recall the ECs are defined by the Weierstrass equation

$$y^2 = x^3 + ax + b \mod p$$

To prevent timing leaks by `quad_res` we compute the existence of a quadratic residue of

$$(x^3 + ax + b)r^2 n$$

where r is a random number and n is a random quadratic non-residue.

Timing leaks for Brainpool curves

```
        if value >= p: continue
        if quad_res(value^3 + a * value + b, p):
            ...
```

Curve	len(p)	$\mathbb{P}[value \geq p]$
brainpoolP224r1	224	15.72 %
brainpoolP256r1	256	33.60 %
brainpoolP384r1	384	45.03 %
brainpoolP512r1	512	33.26 %

- Extra iterations depend on random password
- More iterations done on the real password implies lower execution time variance
- Non trivial to exploit

What makes brainpool curves so "bad" ?

Timing leaks for brainpool curves

The KDF returns a random stream of n bits with n being the number of bits needed to represent p.

A simple example:

- Take $p = 17(= 0b10001)$
- The KDF returns 5 bits
- What is the probability that $\mathrm{KDF}(\cdot) \geq p$?

$$2^{-5}\left(\sum_{i=0}^{3}\binom{3}{i}\right) = 0.25$$

- What about $p = 31$?
- 31 is a *Mersenne prime*: $31 = 2^5 - 1$
- We also need 5 bits to represent 31 ($0b11111$).
- But now $\mathbb{P}[\mathrm{KDF}(\cdot) \geq p] \approx 0.031$

- Brainpool curves use *random* primes.
- In contrast, NIST curves use quasi Mersenne primes, for example

$$p_{192} = 2^{192} - 2^{64} - 1$$

or

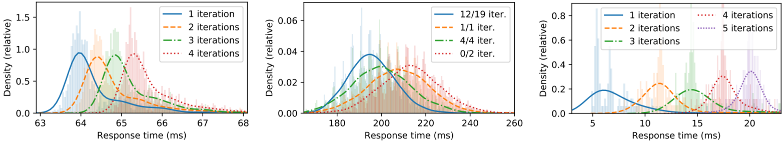
$$p_{521} = 2^{521} - 1$$

Conclusion

This results in very low probabilities for invalid KDF values compared to the case of random brainpool primes.

Exploiting timing leaks in practice

Can we actually deduce the number of iterations ?



(a) WPA3 AP with MODP group 22. (b) WPA3 AP with Brainpool curve 29. (c) EAP-pwd client with curve P-256.

Figure: Response time distributions (Vanhoef and Ronen)

- MODP groups: ~75 measurements / address
- Brainpool: ~2000 measurements / address

Computational Costs in practice

Costs are based on an Amazon EC2 P3 instance with 8 V100 GPUs (74$ / h)

Wordlist / method	Size	Cost for MODP ($)	Cost for P-256 ($)
RockYou	~10^7	$2.1 \cdot 10^{-6}$	$4.4 \cdot 10^{-4}$
HaveIBeenPwned	~10^8	$8.0 \cdot 10^{-5}$	$1.7 \cdot 10^{-2}$
Probable wordlist	~10^9	$1.2 \cdot 10^{-3}$	$2.5 \cdot 10^{-1}$
Bruteforce 8 symbols	~10^{14}	670	$14'000$

Estimated costs (Vanhoef and Ronen)

Denial of service attacks

```
hash_to_curve(password, mac1, mac2):
    k = 40
    while count < k:
        seed = Hash(mac1,mac2,password,count)
        value = KDF(seed, label, p)
        ...
        if quad_res(value^3 + a * value + b, p):
           ...
    y = sqrt(x^3+a*x+b) mod p
    ...
```

- Defenses (extra iterations, QR blinding) are costly
- Tradeoff between DoS and timing leak resistance
- Defense mechanism: Cookies (similar to IKEv2 and wireguard)
- Problem: Wifi is a broadcast medium, we can steal and reflect cookies.
- At 7 faked commits per second the AP is at 80% CPU usage (500bit EC)

Denial of service attacks

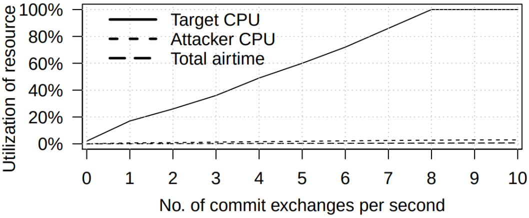


Figure: DoS attack against an AP using a P521 curve (Vanhoef and Ronen)

- Implemented side channel defenses are too costly
- Low end devices might choose not to implement them, favouring performance

Crypto Downgrade attacks

Semantic: Parameters are proposed by client, server responds with Yes/No

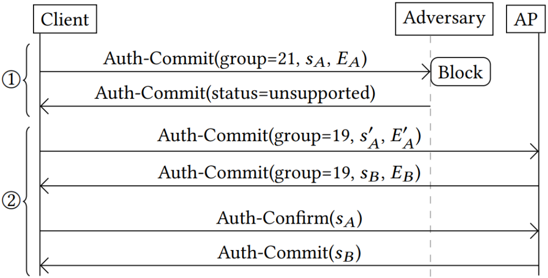


Figure: SAE group negotiation (Vanhoef and Ronen)

Solution

Only support known good groups and curves

WPA3-transition mode

WPA3 transition mode dictates that an AP accepts WPA2 and WPA3 connections using the same password.

- An attacker cannot: Trick a client that an AP in WPA3 transition mode only supports WPA2

Device	Software	Trans	3-Only
MSI GE60	iwd v0.14	●	●
Latitude 7490	Net. Manager 1.17	○	○
Google Pixel 3	qpp1.190205.018.b4	○	○
Galaxy S10	g975usqu1asba	●	●
AP of vendor A	Firmware 10.20.0168	●	○
RaspberryPi 1 b+	OpenWRT r9576	●	○
MSI GE60	wpa_supplicant 2.7	●	○

Figure: List of devices vulnerable to WPA3-trans. / WPA 3 downgrade attacks (Vanhoef and Ronen)

- Setup a rogue WPA2 AP with the same SSID close to the target
- Perform a partial handshake (until client sends authenticated packet)

Solution

Remember which networks support WPA3 (trust on first use)

Invalid Curve attack

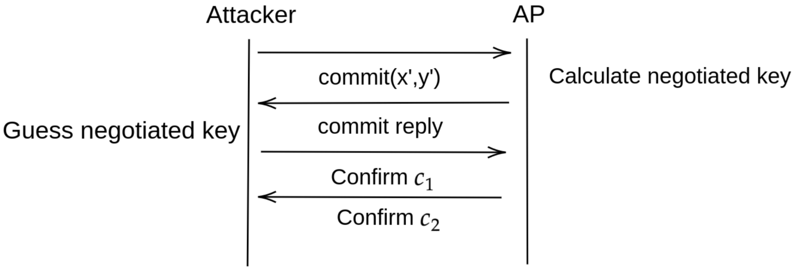


Figure: Forcing the AP to use an invalid point makes the negotiated key predictable

Solution

Implementation should check whether the point is valid (on the curve)

Reflection attack (EAP-PWD)

- On EAP-PWD, the dragonfly handshake is initialized by the AP
- An Attacker can reflect the received frames

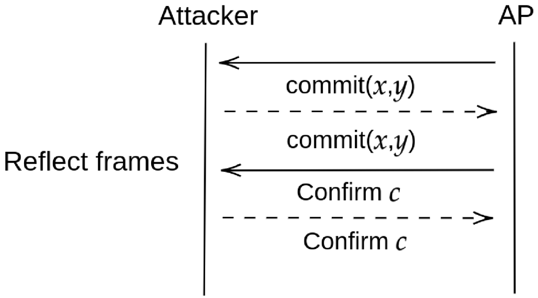


Figure: Reflection attack in a EAP-PWD scenario

Is the attacker authenticated ? Can he send traffic ?

Summary: Implementation vulnerabilities

Software	Invalid	Reflect	$k = 0$	$k \leq 4$
FreeRADIUS	●	●	●	●
Radiator	●	●	●	●
hostapd 2.0-2.7	●	●	2.0-2.6	2.0-2.6
wpa_supplicant 2.0-2.7	●	—	2.0-2.6	2.0-2.6
Aruba client	●	—	●	●
iwd 0.2-0.16	●	—	0.2-0.14	0.2-0.14
hostapd 2.1-2.7	○	—	○	2.1-2.4
wpa_supplicant 2.1-2.7	○	2.1-2.4	○	2.1-2.4
iwd 0.7-0.16	●	○	○	○

Figure: Overview of SAE (bottom) / EAP-PWD (top) implementation vulnerabilities (Vanhoef and Ronen)

Conclusion

Today we've seen:

- Multiple side channel attacks
- The cost of mitigating side channel attacks
- Multiple implementation specific vulnerabilities

You can find the slides / tex online at my GitHub

Should I use WPA3?

Yes. Patches are on the way. Even without them, WPA3 offers better security than WPA2.

References I

Mathy Vanhoef and Frank Piessens. "Key Reinstallation Attacks: Forcing Nonce Reuse in WPA2". In: **Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)**. ACM, 2017.

Mathy Vanhoef and Eyal Ronen. "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd". In: **IEEE Symposium on Security & Privacy (SP)**. IEEE, 2020.