Dragonblood: Attacking the Dragonfly Handshake of WPA3[1]

Mathy Vanhoef
New York University Abu Dhabi
mathy.vanhoef@nyu.edu

Eyal Ronen
Tel Aviv University and KU Leuven
eyal.ronen@cs.tau.ac.il

We systematically analyze WPA3 and EAP-pwd, find denial-of-service and downgrade attacks, present severe vulnerabilities in all implementations, reveal side-channels that enable offline dictionary attacks, and propose design fixes which are being officially adopted.

The WPA3 certification aims to secure home networks, while EAP-pwd is used by certain enterprise Wi-Fi networks to authenticate users. Both use the Dragonfly handshake to provide forward secrecy and resistance to dictionary attacks. In this paper, we systematically evaluate Dragonfly's security. First, we audit implementations, and present timing leaks and authentication bypasses in EAP-pwd and WPA3 daemons. We then study Dragonfly's design and discuss downgrade and denial-of-service attacks. Our next and main results are side-channel attacks against Dragonfly's password encoding method (e.g. hash-to-curve). We believe that these side-channel leaks are inherent to Dragonfly. For example, after our initial disclosure, patched software was still affected by a novel side-channel leak. The leaked information can be used to brute-force the password. For instance, brute-forcing a dictionary of size 10^{10} *requires less than $1 in Amazon EC2 instances. These results are also of general interest due to ongoing standardization efforts on Dragonfly as a TLS handshake, Password-Authenticated Key Exchanges (PAKEs), and hash-to-curve. Finally, we discuss backwards-compatible defenses, and propose protocol fixes that prevent attacks. Our work resulted in a new draft of the protocols incorporating our proposed design changes.*

1 Introduction

After the disclosure of key reinstallation attacks (KRACKs) in WPA2, the Wi-Fi Alliance released WPA3 as the successor of WPA2 [2, 3, 4]. It is important to remark that WPA3 does not define new protocols. Instead, it is a certification that defines which existing protocols a device must support. Simplified, WPA3 mandates support of the Dragonfly handshake, and its only new feature is a transition mode where WPA2 and WPA3 are simultaneously supported for backwards-compatibility. Unfortunately, the security guarantees of WPA3 and its Dragonfly handshake are unclear. For example, a close variant of

[1]This white paper for Black Hat is a shortened version of our full paper [1].

Dragonfly received significant criticism while being standardized [5, 6, 7], while a different variant of Dragonfly has a formal security proof [8]. These contradictory claims raise the question of whether Dragonfly is secure in practice.

We systematically evaluate the security of Dragonfly and its usage in WPA3 and EAP-pwd. The EAP-pwd protocol is used by some enterprise Wi-Fi networks to authenticate users, while WPA3 is used in personal Wi-Fi networks. Both protocols rely on Dragonfly to provide forward secrecy and protection against offline dictionary attacks. We systematically analyze the security of Dragonfly, and confirm all We confirmed all results against proprietary and open source implementations of WPA3 and EAP-pwd. Additionally, we released open source tools so users can check if implementations are vulnerable[2].

To evaluate WPA3 and WPA-pwd implementations, we wrote a test harness to see if edge cases in the Dragonfly handshake are properly handled. This revealed authentication bypasses in all EAP-pwd implementations and in one SAE client. We also audited and reverse engineered implementations, revealing additional vulnerabilities such as known and novel side-channel leaks.

We then study the usage of Dragonfly in WPA3. Here we bypass WPA3-SAE's denial-of-service defense, and overload the CPU of a high-end Access Point (AP). Surprisingly, the cause of SAE's high overhead are defenses against known timing side-channels. Second, we demonstrate a downgrade and dictionary attack against WPA3 when it is operating in transition mode, and we discover a downgrade attack against WPA3's Dragonfly handshake itself. Additionally, we present implementation-specific downgrade and dictionary attacks that work even when the victim uses WPA3-only networks.

Our main results are timing side-channels and new micro-architectural cache side-channels against Dragonfly. These attacks apply to WPA3 and EAP-pwd, leak information about the password, and work even against implementations that have defenses against known side-channel leaks. We refer to our full paper [1] on how this enables offline brute-force attacks. For instance, searching through a dictionary of size 10^{10}, which is larger than all available password dumps, can be done for less than $1 in Amazon EC2 instances.

We believe more openness while creating WPA3 and Dragonfly could have prevented most attacks. For example, excluding the MAC addresses from Dragonfly's password encoding method, or using constant-time algorithms, would have mitigated most attacks.

In collaboration with the Wi-Fi Alliance and CERT/CC we notified affected vendors, and helped write patches to prevent most attacks. Affected vendors and allocated Common

[2]These can be found at `https://wpa3.mathyvanhoef.com/#tools`

Vulnerabilities and Exposures (CVE) IDs can be found at [9]. During this coordinate disclosure, the Wi-Fi Alliance privately created recommendations to securely implement WPA3, in which they claim Brainpool curves are safe to use [10]. However, using Brainpool curves requires extra defenses, and we found that patched WPA3 implementations were still vulnerable when using Brainpool curves. This resulted in a second disclosure round, and highlights the difficulty of implementing Dragonfly without side-channels leaks. Fortunately, our proposed protocol changes that do assure these properties, and thereby prevent most attacks, are being incorporated into WPA3 and EAP-pwd [11, 12].

2 Background

In this section we introduce Dragonfly as used in EAP-pwd and WPA3 [13, 4] and cover parts of the 802.11 standard [14]. Note that the Dragonfly variant used in WPA3 is also known as Simultaneous Authentication of Equals (SAE).

2.1 The Dragonfly Handshake

Dragonfly prevents offline dictionary attacks and provides forward secrecy [15]. It is a Password Authenticated Key Exchange (PAKE), meaning it turns a password into a high-entropy key, It is used in practice by both WPA3 and EAP-pwd [4, 13], and variants are also used in TLS-PWD and IKE-PSK [16, 17, 18].

Dragonfly supports Elliptic Curve Cryptography (ECC) with elliptic curves over a prime field (ECP groups), and Finite Field Cryptography (FFC) with multiplicative groups modulo a prime (MODP groups). We use G for the generator of a group and q for the order of G. Lowercase letters denote scalars, and uppercase letters denote group elements. Elliptic curves are defined over the equation $y^2 = x^3 + ax + b \bmod p$ where p is a prime and a, b, and p depend on the curve being used.

2.1.1 Password Derivation

Before initiating the Dragonfly handshake, the pre-shared password is converted to a group element using a hash-to-element method. The hash-to-element method for MODP groups is called hash-to-group, and the one for elliptic curves is called hash-to-curve. In both algorithms the password element P is generated using a try-and-increment strategy, where in each iteration a hash is first computed over the password, an incremental counter, and the peer's identities (IDs). With EAP-pwd, the input of the hash also includes a random token generated by the server. The hash-to-curve variant uses the hash

Listing 1: Hash-to-curve method in Python-like pseudocode. If token is None the SAE variant is executed [14, §12.4.4.2.2], otherwise it executes the EAP-pwd variant [13].

```
1 def hash_to_curve(password, id1, id2, token=None):
2   found, counter, base = False, 0, password
3   label = "EAP-pwd" if token else "SAE"
4   k = 0 if token else 40
5   while counter < k or not found:
6     counter += 1
7     seed = Hash(token, id1, id2, base, counter)
8     value = KDF(seed, label + " Hunting and Pecking", p)
9     if value >= p: continue
10    if is_quadratic_residue(value^3 + a * value + b, p):
11      if not found:
12        x, save, found = value, seed, True
13        base = random()
14
15  y = sqrt(x^3 + a * x + b) mod p
16  P = (x, y) if LSB(save) == LSB(y) else (x, p - y)
17  return P
```

output as the x coordinate of a point, and it then checks if there is a solution for y over the equation $y^2 = x^3 + ax + b \bmod p$ (see Listing 1). If a solution exists, the password element is the point (x, y). Otherwise, the counter is incremented, and another attempt is made to find a solution for y using the new x value. We discuss the hash-to-group method in Section 5, and unless otherwise noted, we assume the elliptic curve variant is used since it is more widely deployed.

To mitigate timing leaks, WPA3-SAE executes the while loop k times no matter when P is found. However, no value for k is suggested, and EAP-pwd does not even have this defense. Other variants of Dragonfly use $k = 40$ [17, 19], and several SAE implementation also use this value (see Section 3.2). In the extra iterations, operations are based on a random password. Information may also leak when checking if there is a solution for y in line 10. The EAP-pwd standard does not realize this, and directly tries to calculate y, which may take longer when there is a solution. In contrast, WPA3 recommends to first check if there is a solution using the Legendre function before calculating y. Unfortunately, even a Legendre function can leak info if not carefully implemented [20]. To prevent this, an update to 802.11 recommends Quadratic Residue (QR) blinding [21]. With this defense, a random number is first generated, squared, and multiplied to the number being checked. The result is then multiplied with a random quadratic (non-)residue, before executing the Legendre function [14, §12.4.4.2.2].

2.1.2 Commit and Confirm Phase

The Dragonfly protocol itself consists of a commit and confirm phase. Figure 1 illustrates these phases, and the corresponding curve operations. In infrastructure WPA3 networks the client always sends the first commit, while with EAP-pwd the RADIUS authentication server always sends the first commit frame. In this paper we consider the RADIUS server and AP to be the same entity.

In the commit phase, each peer picks two random numbers $r_i, m_i \in [2, q[$ such that $r_i + m_i \in [2, q[$ (see Fig. 1). They then calculate $E_i = -m_i \cdot P$ and send s_i and E_i to each other. On reception of these values, each peer verifies that the received s_i is within the range $[1, q[$, and that E_i is a valid point on the curve [14, §12.4.5.4]. If one of these checks fails, the handshake is aborted. Forward secrecy is provided since deriving m_i given P and E_i is hard, i.e., it relies on the elliptic curve discrete logarithm problem.

In the confirm phase, each peer calculates the secret point K (see Fig. 1). The x-coordinate of this point is processed using a hash function to derive the key κ, and a HMAC is computed over the handshake summary tr with as key κ. The result of this hash, denoted by c_i, is sent to the other peer in a confirm frame. On reception of c_i, the receiver verifies its value. It if differs from the expected value, the confirm frame is ignored. Otherwise the handshake succeeds and the negotiated key is κ.

2.2 Dragonfly in WPA3 and EAP-pwd

In practice, Dragonfly is used in personal WPA3 networks, and in enterprise WPA2 or WPA3 networks that authenticate clients using EAP-pwd. The Dragonfly variant used in personal WPA3 networks is called SAE. In its hash-to-element algorithm, the identities of both peers are their MAC addresses. After executing SAE, the negotiated key is used in a 4-way handshake to derive a fresh session key. Although WPA3 still uses WPA2's 4-way handshake, it is not vulnerable to dictionary attacks because the key generated by SAE has much higher entropy than an ordinary password.

Enterprise Wi-Fi networks can use various EAP-based authentication methods. We focus on EAP-pwd, which is based on Dragonfly [13]. In EAP-pwd, the AP initiates the handshake, and the commit and confirm frames are encapsulated in 802.1X frames.

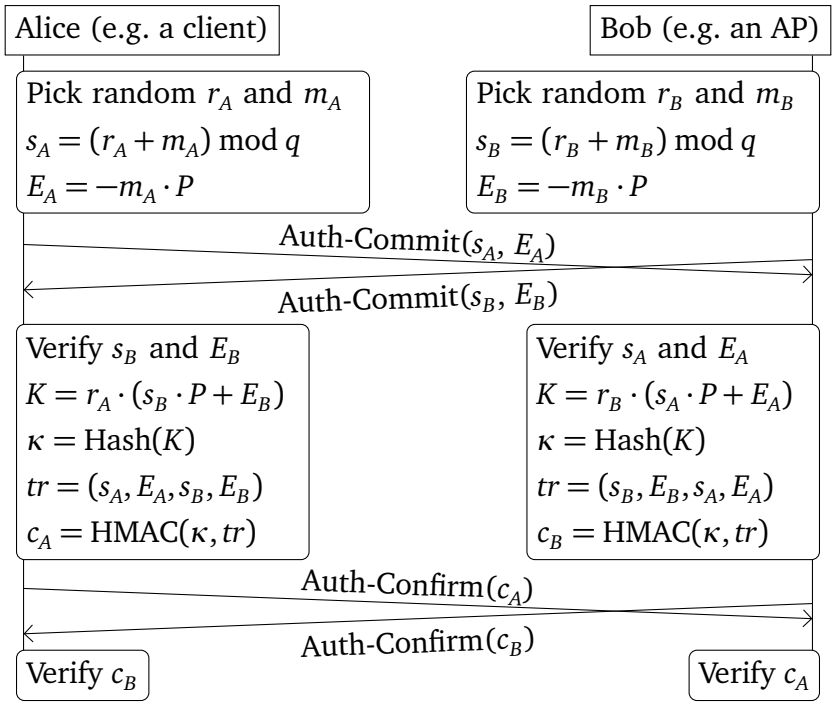


Figure 1: WPA3's SAE handshake. Both stations can simultaneously initiate the handshake, hence the crossed arrows. Here we assume elliptic curves are used, though similar operations are performed when using multiplicative groups.

2.3 WPA3-SAE Transition Mode

To accommodate older devices that do not support WPA3, a network can operate in a transition mode where WPA2 and WPA3 are simultaneously supported using the same password. In this mode the only requirement placed on WPA3 clients is that they must use Management Frame Protection (MFP), even though the network advertises it as optional. The WPA3 certification does not discuss the security of transition mode [4]. Nevertheless, one would expect that if all devices in a network support WPA3, it is as secure as normal WPA3. Unfortunately, in Section 4.1 we show this is not the case.

3 A Systematic Analysis of Dragonfly

In this section we describe our methodology, and we evaluate the security of EAP-pwd and WPA3-SAE implementations.

3.1 Test Harness and Discovered Flaws

We created black-box tests for EAP-pwd and SAE to verify whether the following checks are implemented. First, the receiver of a commit frame must check that s_A is in the range

Table 1: EAP-pwd and SAE tools that accept an invalid scalar/element (2nd column), do not detect reflection attacks (3rd column), or have known timing leaks (k-columns).

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	●	○	○	○

$[2, q[$, and must check that E_A is a member of the group (e.g. that point E_A is on the curve). Additionally, the initiator must detect reflection attacks where the peer reflects the scalar and element. Table 1 lists the tested implementations and discovered attacks.

None of the EAP-pwd implementations validate the received scalar or element. This can be abused in an invalid curve attack, where the adversary sends a point that is on an invalid curve with a very small number of elements, making the key K guessable [22]. For the details of this attack, we refer to our full paper at [1].

All server-side EAP-pwd daemons were vulnerable to reflection attacks. This attack allows the adversary to authenticate as the victim, but does not reveal the session key κ.

For SAE, wpa_supplicant 2.1 to 2.4 are affected by reflection attacks. This can be abused to set up a rogue AP, and complete the SAE handshake, though traffic cannot be intercepted since κ is unknown. We also found that iwd did not verify the received scalar. To exploit this, we send a scalar s_B equal to the order of the elliptic curve such that $s_B \cdot P$ equals the point at infinity $\mathcal{O}$. We then construct a valid point E_B such that $\mathcal{O} + E_B$, when computed by iwd, again equals the point at infinity $\mathcal{O}$, causing κ to be zero. Since iwd can be forced into using this curve, the bug is exploitable, and allows an attacker to act as a rogue AP and intercept all traffic sent by the client.

3.2 Code Audits and Reverse Engineering

The initial specification of EAP-pwd and SAE did not perform extra iterations in the hash-to-curve method [13, 23]. Only SAE got updated to perform extra iterations [24]. Therefore, we evaluate which defenses are deployed in practice.

Version 2.1 up to 2.4 of hostapd and wpa_supplicant use $k = 4$ for SAE, while newer versions use $k = 40$. No extra iterations are performed in EAP-pwd's hash-to-curve method of FreeRADIUS, Radiator, Aruba, iwd 0.16 and lower, and in version 2.0 to 2.6 of hostapd and wpa_supplicant. This can be abused against clients to recover the password.

In FreeRADIUS, the hash-to-curve algorithm of EAP-pwd aborts when 11 or more iterations are needed. This means that one out of every 2048 handshakes fails, which reveals that the password element was not found in the first 10 iterations. We successfully abused this side-channel leak to brute-force the password.

When reversing Aruba's EAP-pwd client for Windows, we found that it generated insecure random numbers. This allows an adversary to predict m_A and recover the point P from E_A.

We also reverse engineered two firmware images that are run on Wi-Fi radios. Interestingly, they execute at minimum only 8 iterations, which is insufficient to prevent information leaks. We conjecture this was done because executing 40 iterations is too costly.

4 Wi-Fi-Centric Attacks

In this section we present downgrade and dictionary attacks against WPA3-SAE. We also compare Dragonfly's high overhead with other hash-to-curve methods, and abuse its high overhead by defeating SAE's denial-of-service (DoS) defense.

4.1 Downgrade & Dictionary Attacks

4.1.1 Attacking WPA3 Transition Mode

In the transition mode of WPA3, an AP accepts connections using WPA3-SAE and WPA2 with the same password. This provides compatibility with older clients, while WPA2's 4-way handshake detects downgrade attacks. As a result, WPA3 provides forward secrecy, even when using the transition mode of WPA3-SAE.

The problem is that, although downgrade attacks are detected by WPA2's 4-way handshake, by that point an adversary has captured enough data to perform a dictionary attack. This is because an adversary only needs to capture a single authenticated 4-way handshake message to carry out a dictionary attack [25]. The adversary simply needs to create a rogue WPA2 AP with the same SSID as the target. This causes the client to connect to the rogue AP using WPA2. Based on the authenticated message 2 sent by the victim, a dictionary attack can be carried out [25].

4.1.2 Attacking SAE's Group Negotiation

SAE can be run using different elliptic curve or multiplicative groups, and the 802.11 standard allows station to prioritize groups in a user-configurable order. Although this provides flexibility, it requires a secure method to negotiate the group to use.

With SAE, the group is negotiated by letting the client include its desired group in the commit frame, along with a valid scalar s_i and element E_i. If the AP does not support this group, it replies using a commit frame with a status field equal to "unsupported group". In turn the client sends a new commit frame using its next preferred group, along with a new scalar and element. This process continues until the client selects a curve that the AP supports. Unfortunately, there is no mechanism to detect if someone interfered with this process. This makes it straightforward to force the client into using a different group by forging a commit frame that indicates the AP does not support the selected group.

We confirmed the attack against wpa_supplicant. To block legitimate commit frames, we modified Atheros firmware to read the header of frames being transmitted, to then jam the remaining content in case it is a commit frame we want to block [26].

4.1.3 Countermeasures

To mitigate our downgrade to dictionary attack, a client should remember if a network supports WPA3-SAE. That is, after successfully connecting using SAE, the client never connect to this network using a weaker handshake. Group downgrade attacks can be mitigated by including a bitmap of the supported groups in the RSNE during the 4-way handshake. This will enable a station to detect if a downgrade attack took place.

4.2 The High Overhead of Dragonfly

When counting the number of operations that Dragonfly's hash-to-curve method requires, we find that it requires much more operations than alternative methods (see Table 2). This high overhead is caused by the try-and-increment loop, where at least 40 iterations are always executed to mitigate timing leaks. The designers realized that an adversary can abuse this overhead by spoofing commit frames. To defend against this an anti-clogging mechanism was added, where client must reflect a secret cookie sent by the AP, before the AP processes the client's commit frame.

Unfortunately, it is trivial to spoof MAC addresses and bypass the above defense. To demonstrate this, we created a tool where the adversary acts as a client, injects commit frames using spoofed MAC addresses, and reflects any secret cookies it receives. In our

Table 2: Operations needed to hash to a curve for various methods [27]. We assume Dragonfly uses a constant time Legendre function instead of quadratic residue blinding.

Method	Hash	$x+y$	$x \cdot y$	x^2	x^y	x^{-1}	$\sqrt{x}$
Dragonfly	80	80	40	40	80	0	1
Icart	1	5	6	3	1	1	0
SWU	2	8	6	5	2	1	1
S-SWU	1	6	4	4	1	1	1

experiments, spoofing 8 commit exchanges per second using curve P-521 causes the AP's CPU usage to reach 100%. Clients that now try to connect using WPA3 either face long delays, or cannot connect at all.

One solution is to make the password element is independent of the peers's identities. The password element can then be calculated offline and reused in subsequent handshakes. Another solution is to use a more efficient hash-to-curve method (e.g. one of Table 2).

Our attack shows that Dragonfly's timing leak defenses are too costly. Moreover, we believe lightweight devices will not fully implement all defenses because of this

5 Timing Attacks

In this section we show that the hash-to-group and hash-to-curve methods are vulnerable to (novel) timing attacks. The obtained info can be used to recover the victim's password.

5.1 Variable Number of Iterations

Apart from elliptic curves, SAE and EAP-pwd also support MODP groups, in which case the hash-to-group method in Listing 2 is used. Line 5 causes extra iterations when the output of the Key Derivation Function (KDF) returns a number bigger or equal to the prime p. The CFRG warned about this, but did not analyze the leak in detail [28]. The number of bits returned by KDF is equal to the number of bits needed to represent p, meaning the probability that *value* is bigger than p depends on the group being used. For most MODP groups this probability is negligible, because p is close to a power of two. However, for the RFC 5114 groups 22, 23, and 24, the probability that the output of KDF is bigger than p is high [29]. For example, for group 22 this probability equals 30.84%, and for group 24 the probability is 47.01%.

Listing 2: Hash-to-group method in Python-like pseudocode. If token is None the SAE variant is executed [14, §12.4.4.3.2], otherwise it executes the EAP-pwd variant [13].

```
1 def hash_to_group(password, id1, id2, token=None):
2   label = "EAP-pwd" if token else "SAE"
3   for counter in range(1, 256):
4     seed = Hash(token, id1, id2, password, counter)
5     value = KDF(seed, label + " Hunting and Pecking", p)
6     if value >= p: continue
7
8     P = value^((p-1)/q) mod p
9     if P > 1: return P
```

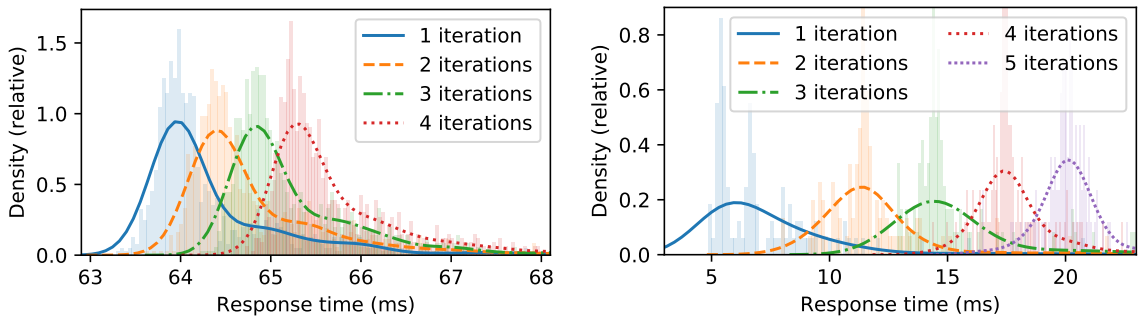
Since the KDF output depends on the password, the number of executed iterations also depends on the password. If someone learns this number, they learn that passwords which need a different number of iterations are not being used. For hash-to-group the number of executed iterations X follows a geometric distribution:

$$\Pr[X = n] = \Pr[value \geq p]^{n-1} \cdot (1 - \Pr[value \geq p]) \tag{1}$$

Hence the average number of iterations needed to derive P for MODP groups equals $E[X] = (1 - \Pr[value \geq p])^{-1}$. For group 22, this equals 1.45, and for group 24 this equals 1.89 iterations. In other words, on average one timing measurement allows the adversary to learn the result of multiple iterations. Moreover, the MAC addresses (i.e. identities) of the peers also influence the output of the KDF, and hence also influence the number of executed iterations. This means we can attack clients and APs by spoofing MAC addresses, and for each address measure the number of executed iterations.

5.2 Timing Attacks against Brainpool Curves

During our initial coordinated disclosure, the Wi-Fi Alliance privately created recommendations to mitigate our attacks [10]. These recommendations state that Brainpool curves are safe to use, and that no extra defenses are needed when using them. However, even though the hash-to-curve method already has timing leak defenses, it still suffers from timing leaks when using Brainpool curves. The problem is that, similar to hash-to-group, the hash-to-curve method also checks if the KDF output is smaller than p (line 8 in Listing 1). For most curves this is not an issue, since their prime is close to a power of two, but with Brainpool curves this check can fail with high probability. We refer to our full paper on how this can be exploited [1].



(a) WPA3 AP with MODP group 22.

(b) EAP-pwd client with curve P-256.

Figure 2: Response time distributions of timing attacks for selected parameters. The victim uses a Raspberry Pi 1 B+. Graph 2a targets hostapd 2.6, and graph 2b iwd 0.14.

5.3 Experiments against WPA3-Enabled APs

Our two experiments target APs. We used a Raspberry Pi 1 B+ for the AP because its 700 MHz CPU matches the typical CPU of home routers and professional APs [30]. The Raspberry Pi used a WNDA3200 Wi-Fi dongle. Hostapd was used as the AP daemon, since it is the most widely used daemon in both professional and home routers. We wrote a tool that spoofs commit frames, and measures response times. After each measurement, a deauthentication frame is sent, causing the target to clear all state related to the spoofed address and enabling us to rapidly perform new measurements.

Two optimizations are important. First, we use virtual interface support of Atheros chips to acknowledge frames sent to spoofed MAC addresses. This stops the AP from retransmitting frames, making the attack more reliable. Second, response times are influenced by background traffic and background tasks on the AP. Both sources of noise are problematic because they are not constant throughout an attack. To handle this, we interleave the time measurements of spoofed MAC addresses, instead of performing all measurements for each address one by one. As a result, temporal noise equally influences the timings of all addresses, instead of only affecting one address.

Using our setup we attacked hostapd 2.6 using MODP group 22. We spoofed 20 addresses and made 1 000 measurements for each address. Figure 2a shows the resulting response time distributions of selected MAC addresses that each result in a different number of iterations. We evaluated several statistical tests to differentiate addresses that result in a different number of iterations, such as simple averages, Student's and Welch's T-test, Mann-Whitney U test, Wilcoxon signed-rank test, one-way ANOVA, and Crosby's box test [31]. With these tests, there is a trade-off between the amount of differences

detected, and the chance of false positives. Our goal is to detect as many differences without any false positives. Even under these conditions, Crosby's box test outperformed all classical tests. When using this test with a low percentile of 5 and high percentile of 35, we need 75 measurements per address to differentiate all addresses that require a different number of iterations with 99.5% confidence. These pairwise comparisons are then used to sort MAC addressed based on the number of executed iterations. From this ranking we can derive bounds on how many iterations each MAC address executed.

5.4 Attacking SAE and EAP-pwd Clients

Our next experiment targets clients. To simulate devices that offload the SAE handshake to their Wi-Fi chip, we tested our attacks against a Raspberry Pi 1 B+ with iwd as a lightweight client. Running iwd on the Raspberry Pi required recompiling Linux to enable recent kernel features.

To attack a WPA3-SAE client, we need to know when it starts executing the hash-to-element method. Since the client initiates the handshake, we cannot do this for the first commit frame it sends. Instead, the rogue AP responds to the client that the offered group is not supported. This causes the client to build a commit frame for another group, which requires executing the hash-to-element from scratch. We can measure how long this takes, and hence perform timing attacks against both WPA3-SAE and EAP-pwd clients.

As an example, we perform a timing attack against an iwd client using EAP-pwd with curve P-256. With EAP-pwd, the number of executed iterations are influenced by the client's username, the identity of the server, and by a token generated by the server (see e.g. line 7 in Listing 1). Because the server always generates a new random token, we cannot attack it. Instead we attack the client by spoofing 20 different tokens. The resulting timing measurements for selected tokens are shown in Figure 2b. Using Crosby's box test with a low percentile of 5 and high percentile of 45, we can recover the number of iterations using 30 timing measurements per token.

5.5 Countermeasures

The ideal defense is to exclude the MAC addresses (i.e. identities) from the hash-to-element methods. Similar to our defense against DoS attacks, this would allow the password element to be calculated offline and then reused. Although timing leaks may still occur, for a given password the execution time would then always be identical, meaning on average only two password bits are leaked. This change also makes it harder to trigger and measure executions of the algorithm. Another option is to use a constant-time hash-to-curve method (e.g. one of Table 2). Unfortunately, both these changes are not backwards-compatible.

6 Cache-Based Attacks on ECC groups

In this section we demonstrate that implementations of the hash-to-curve algorithm of SAE may be vulnerable to cache-based side-channel attacks.

The goal of our attack is to learn if first iteration of the hash-to-curve algorithm succeeded or not. This information can be used in an offline password brute-force attack to recover the target's password [1]. Unlike the hash-to-element method, the implementation of the hash-to-curve algorithm for ECC groups does include mitigations against side-channel attacks. Those mitigations include performing extra dummy iterations on random data [14, §12.4.4.3.2], and blinding of the underlying cryptographic calculation of the quadratic residue test [21]. The resulting code of wpa_supplicant and hostapd implementation we reviewed is pseudo-constant time, i.e., there might be some minor variation in run time, but they are too minute to be measured by an adversary. However, such pseudo-constant time implementations might still be vulnerable to different types of micro-architectural side-channel attacks [32, 33, 34].

6.1 Micro-Architectural Side-Channel Attacks

Modern processors try to optimize their behavior (e.g. memory access, branch prediction) by saving an internal state that depends on the past. Micro-architectural side-channel attacks exploit leaked information about the running of other programs due to sharing of this state (for a survey see [35]). Cache-based side-channel attacks exploit the state of the memory cache (either instructions or data) and have been widely used to break cryptographic primitives [36, 37, 38, 39, 40]. Cache attacks can be seen as a way to partly circumvent process (or virtual machine) isolation. Although an attacker running code in an unprivileged process is not able to read the memory of the target process, he can still learn information about the memory access patterns.

In the FLUSH+RELOAD attack [39] the attacker starts by evicting (or flushing) a memory location from the cache. After waiting for a predetermined interval, he measures the time it takes to reload the flushed location and then flushes it again. If during the interval the victim accesses this memory location, it will be cached, and the reload time for the attacker will be short. Otherwise, the reloading of the flushed memory location will be much slower. In this way, the attacker can trace the victim's memory access patterns.

6.2 Attacking the hostap Implementation

We target the `sae_derive_pwe_ecc` function of the latest hostap code before our initial disclosure (commit `0eb34f8f2`) with the default curve P-256. Our test machine uses a 4-core Intel Core i7-7500 processor, with a 4 MiB cache and 16 GiB memory, running Ubuntu 18.04.1. We monitor the instruction cache accesses of wpa_supplicant with an unprivileged user-mode spy process. This is accomplished using the FLUSH+RELOAD attack of the Mastik toolkit [39, 41].

We want to leak the result of the QR test in the first iteration of the hash-to-curve algorithm. We can try to attack the blinded QR test code, or the code that checks the result of the test. A simple cache attack against the blinded QR test is infeasible as the two possible code paths are compiled into a single cache line.[3]

The two code paths of the branch inside the iteration loop (see Listing 3 line 17) are compiled into two separate cache lines. Therefore we can monitor cache access to nQR case cache line which is the target of the conditional jump (see Listing 4 line 9). To differentiate between the branches taken in the first and subsequent iterations, we created a synchronization "clock" by monitoring another cache line that is accessed once every iteration (similarly to [44]).

On our test platform, monitoring two cache lines repeatedly over time caused a high rate of false positives (i.e. false detection of access to cache lines). This error rate increases considerably when the monitored cache lines are close. Consequently, for our "clock" monitor a cache line far away from the nQR cache line (in our case the function `sha256_prf_bits`).

6.2.1 Cache Template Attack

We want to learn the result of the QR test in the first iteration for each cache trace we measured. However, our measurements are noisy, and the measured cache access patterns to the two monitored cache lines show a high variance between different traces with the same result. This might be due to OS related noise, speculative execution, or due to the influence of the random dummy iterations on the branch predictor. To overcome this, we perform a simplified variant of a cache template attack [45, 46]. That is, we measure a trace of the cache access pattern by monitoring the two addresses (the "clock" and the non-QR case) in fixed intervals of $5 \cdot 10^4$ clock cycles (each iteration takes

[3] More advanced micro architectural attacks targeting the branch predictor [42, 43, 34] will fail due to the extra random iterations.

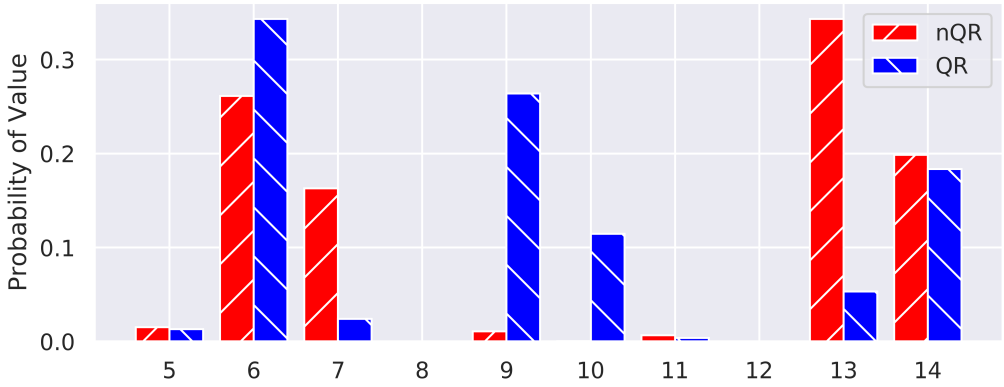


Figure 3: Probability distribution for attack results

roughly $2 \cdot 10^5$ clock cycles on our test machine). We encode each measurement as a bit, with value one if the measured cache line was accessed and zero if it wasn't. Each interval corresponds to two bits. We encode each interval in the trace into two bits that correspond to the two memory locations.

Our attack returns the the first two non zero intervals. This means the return value consists of 4 bits (resulting in 9 possible return values). Figure 3 shows the distribution of these return values when the first iteration of the hash-to-curve algorithm results in a non-QR number (nQR), and when the first iteration results in a QR number (QR).

To overcome the noise and achieve a high success rate, we repeat the attack for 20 times for each MAC address, and use a simple linear classifier to get the result.

We trained our classifier with two training sets of $100 \cdot 20$ traces for each of the non-QR and the QR cases. We then tested our attack and linear classifier on a larger test set of $400 \cdot 20$ traces for each case. We achieved a 100% success rate (400 out of 400) in the non-QR case, and a 99.5% success rate (398 out of 400) in the QR case.

6.3 Cache Attacks against Brainpool Curves

After our initial coordinate disclosure, hostap mitigated the vulnerabilities described in Section 6.2. However, as discussed in Section 5.2, the patched code still has a secret-dependent branch that can be exploited when using Brainpool Curves. This allows for a new cache attack similar to our original one in the same attack scenario. Using the same test setup and technique, targeting the latest patched version of hostap (commit `e0e15fc23`).

The fixed interval of our attack is reduced to $5 \cdot 10^3$ clock cycles (if the resulting hash is larger than the modulus then the iteration is very short). For our "clock" we monitor

a cache line inside the `hmac_sha256_vector` function that is accessed once in each iteration (called by `sha256_prf_bits`). The second monitored cache line is inside the `crypto_bignum_init_set` function that is called only if the resulting hash is smaller than the modulus. This new attack is more robust than the original one, achieving 100% success rate using only 10 traces for each MAC address.

6.4 Discussion and Countermeasures

We believe that all Dragonfly variants are affected by our attack. Similar to our timing attacks, the ideal solution is to use a constant-time hash-to-curve method, and to exclude the peer's identities from the password element computation. As a backwards-compatible defense, a constant-time Legendre function can be used, secret-dependent branches can be replaced with constant-time select utilities, and at least k iterations must always be executed. Additionally, when using Brainpool curves, line 10 in the hash-to-curve algorithm of Listing 1 should always be executed.

7 Related Work

After the introduction of WPA, it was quickly found to be vulnerable to dictionary attacks [25]. Later, He and Mitchell formally analyzed WPA's 4-way handshake, and discovered a DoS vulnerability [47, 48]. This resulted in the standardization of a slightly improved variant [14]. He et al. continued to analyze the 4-way handshake, and proved its correctness [49]. However, implementations of the 4-way handshake were still vulnerable to downgrade attacks [50]. Researcher also found that the older WPA-TKIP protocol is also still commonly supported and vulnerable to side-channel attacks [51, 52]. Recently, Vanhoef and Piessens discovered that WPA2 was vulnerable to key reinstallation attacks [2, 3]. To make practical man-in-the-middle attacks against handshakes more difficult, operating channel verification was recently added to 802.11 [53]. Finally, Kohlios and Hayajneh provide an overview of WPA2 and the differences with WPA3 [54].

Researchers also discovered several DoS attack against Wi-Fi networks. The most well-known is the deauthentication attack [55]. Additionally, Könings et al. found several DoS vulnerabilities in the physical and MAC layer of 802.11 [56], and other researchers constructed jammers using commodity hardware [26, 57]. A detailed survey of DoS attacks at the physical and MAC layer is given by Bicakci and Tavli [58]. Aiello et al. show how susceptibility to denial-of-service attacks can be balanced with the need for perfect forward secrecy [59]. To the best of our knowledge, our clogging attack against WPA3 is the first that overloads the CPU of the victim.

An initial version of Dragonfly was vulnerable to an offline dictionary attack [60]. A modified variant was then specified in 2008 [15]. Several close variants of it have been defined over the years [13, 16, 18, 17]. Trevor Perrin did a review of an improved draft of the handshake [61], and later provided an overview of other people's comments on the handshake [5]. Struik reviewed a draft of the handshake [6]. Clarke and Hao discovered a small subgroup attack against a draft of Dragonfly, which was mitigated in a new draft [62]. Lancrenon and Skrobot provided a security proof of a close variant of Dragonfly [8]. Finally, Alharbi et al. designed a variant of Dragonfly that attempts to keep computational costs low [63].

Other types of PAKEs have also been proposed by researchers over the years [64, 65, 66, 67, 68, 69, 70], some of which have been submitted as RFCs [71, 72, 73, 74, 75].

8 Conclusion

In light of our attacks, we believe that WPA3 does not meet the standards of a modern security protocol. Since EAP-pwd uses a close variant of WPA3's Dragonfly handshake, it is affected by similar flaws. We believe that a more open design process would have avoided these weaknesses.

Notable is that most of our attacks are against the password encoding method of Dragonfly, i.e., against its hash-to-group and hash-to-curve algorithm. This indicates that implementing these methods without side-channel leaks is very tedious. Also notable is that Dragonfly supports a large variety of cryptographic groups, making it hard to fully analyze the handshake. Both points are evidenced by the fact that after our initial disclosure, patched implementations were still vulnerable to a novel side-channel attack.

Interestingly, a minor change to Dragonfly's password encoding algorithm would have prevented most of our attacks. In particular, the peer's MAC addresses (i.e. identities) can be excluded from the password encoding algorithm, and instead included later on in the handshake. For EAP-pwd the server's random token must also be excluded. This allows the password element to be computed offline, meaning an attacker can no longer actively trigger executions of the password encoding method. It also means that for a given password the execution time of the password encoding method is always the same, limiting the amount of info being leaked, which cannot help an attacker to guess the password by much [76]. Surprisingly, when the CFRG was reviewing a variant of Dragonfly, they in fact suggested this type of modification [77, 61, 78, 79, 80]. If this criticism would have incorporated, most of our attacks would have been avoided.

We conjecture that resource-constrained devices will not fully implement all backwards-compatible side-channel defenses, because the resulting overhead is too high. In fact, we already found Wi-Fi radios that only partly mitigate timing attacks. Moreover, correctly implementing all backwards-compatible side-channel countermeasures is non-trivial. This is worrisome, because security protocols should be designed to reduce the change of implementation vulnerabilities. Fortunately, in reaction to our results, both WPA3 and EAP-pwd are now standardizing a constant-time password encoding algorithm that can be executed offline [11, 12].

References

[1] Mathy Vanhoef and Eyal Ronen. Dragonblood: A security analysis of WPA3's SAE handshake. Cryptology ePrint Archive, Report 2019/383, 2019. https://eprint.iacr.org/2019/383.

[2] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *CCS*, 2017.

[3] Mathy Vanhoef and Frank Piessens. Release the kraken: new KRACKs in the 802.11 standard. In *CCS*, 2018.

[4] Wi-Fi Alliance. WPA3 specification version 1.0. Retrieved 6 April 2019 from https://www.wi-fi.org/file/wpa3-specification-v10, April 2018.

[5] Trevor Perrin. [TLS] question regarding CFRG process. Retrieved 29 October 2018 from https://www.ietf.org/mail-archive/web/tls/current/msg10962.html, 2013.

[6] Rene Struik. [Cfrg] review of draft-irtf-dragonfly-02 (triggered by [TLS] working group last call for draft-ietf-tls-pwd). Retrieved 9 November 2018 from https://www.ietf.org/mail-archive/web/cfrg/current/msg03527.html, November 2013.

[7] Joseph Salowey. [TLS] conclusion of WGLC draft-ietf-tls-pwd. Retrieved 7 April from https://mailarchive.ietf.org/arch/msg/tls/Fep2-E7xQX7OQKzfxOoFInVFtm4, December 2013.

[8] Jean Lancrenon and Marjan Škrobot. On the provable security of the Dragonfly protocol. In *Information Security*. Springer International Publishing, 2015.

[9] CERT/CC. Vulnerability note vu#871675: Security issues with WPA3. Last retrieved 29 July from `http://www.kb.cert.org/vuls/id/871675`, 2019.

[10] Wi-Fi Alliance. WPA3 security considerations overview. Retrieved 24 May 2019 from `https://www.wi-fi.org/file/wpa3-security-considerations`, April 2019.

[11] Dan Harkins. Finding PWE in constant time. Retrieved 24 July 2019 from `https://mentor.ieee.org/802.11/dcn/15/11-19-1173-08-000m-pwe-in-constant-time.docx`, July 2019.

[12] Dan Harkins. Improved Extensible Authentication Protocol Using Only a Password. Internet-Draft draft-harkins-eap-pwd-prime-00, Internet Engineering Task Force, July 2019. Work in Progress.

[13] Dan Harkins and G. Zorn. Extensible authentication protocol (EAP) authentication using only a password. RFC 5931, August 2010.

[14] IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*, 2012.

[15] Dan Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In *The Second International Conference on Sensor Technologies and Applications (SENSORCOMM)*, pages 839–844, Aug 2008.

[16] Dan Harkins. Secure pre-shared key (PSK) authentication for the internet key exchange protocol (IKE). RFC 6617, June 2012.

[17] Dan Harkins. Secure Password Ciphersuites for Transport Layer Security (TLS). RFC 8492, 2019.

[18] Dan Harkins. Dragonfly key exchange. RFC 7664, November 2015.

[19] Kevin M. Igoe. Re: [Cfrg] status of DragonFly. Retrieved 9 September 2018 from `https://www.ietf.org/mail-archive/web/cfrg/current/msg03264.html`, December 2012.

[20] Thomas Icart. How to hash into elliptic curves. In *Advances in Cryptology (CRYPTO)*, 2009.

[21] Dan Harkins. Addressing a side-channel attack on SAE. Retrieved 9 September 2018 from `https://mentor.ieee.org/802.11/dcn/14/11-14-0640-00-000m-side-channel-attack.docx`, 2014.

[22] Ingrid Biehl, Bernd Meyer, and Volker Müller. Differential fault attacks on elliptic curve cryptosystems. In *Advances in Cryptology (CRYPTO)*. Springer, 2000.

[23] IEEE Std 802.11s. *Amendment 10: Mesh Networking*, 2011.

[24] IEEE Std 802.11. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Spec*, 2012.

[25] Robert Moskowitz. Weakness in passphrase choice in WPA interface. Retrieved 26 September 2018 from `https://wifinetnews.com/archives/2003/11/weakness_in_passphrase_choice_in_wpa_interface.html`, 2003.

[26] Mathy Vanhoef and Frank Piessens. Advanced Wi-Fi attacks using commodity hardware. In *ACSAC*, 2014.

[27] Sam Scott, Nick Sullivan, and Christopher A. Wood. Hashing to Elliptic Curves. Internet-Draft draft-irtf-cfrg-hash-to-curve-03, Internet Engineering Task Force, March 2019. Work in Progress.

[28] Scott Fluhrer. Re: [Cfrg] status of DragonFly. Retrieved 8 November 2018 from `https://www.ietf.org/mail-archive/web/cfrg/current/msg03265.html`, December 2012.

[29] Matt Lepinski and Stephen Kent. Additional Diffie-Hellman Groups for Use with IETF Standards. RFC 5114, 2008.

[30] WikiDevi. Semantic search: wireless routers. Last retrieved 14 November 2018 form `https://wikidevi.com/`, 2018.

[31] Scott A. Crosby, Dan S. Wallach, and Rudolf H. Riedi. Opportunities and limits of remote timing attacks. *ACM Trans. Inf. Syst. Secur.*, 12(3), 2009.

[32] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Lucky 13 strikes back. In *ASIA CCS*, 2015.

[33] Eyal Ronen, Kenneth G. Paterson, and Adi Shamir. Pseudo constant time implementations of TLS are only pseudo secure. In *CCS*, 2018.

[34] Eyal Ronen, Robert Gillham, Daniel Genkin, Adi Shamir, David Wong, and Yuval Yarom. The 9 lives of bleichenbacher's CAT: new cache attacks on TLS implementations. In *To appear in the IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2019.

[35] Qian Ge, Yuval Yarom, David Cock, and Gernot Heiser. A survey of microarchitectural timing attacks and countermeasures on contemporary hardware. *J. Cryptographic Engineering*, 8(1), 2018.

[36] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: The case of AES. In *CT-RSA*, 2006.

[37] Daniel J. Bernstein. Cache-timing attacks on AES, 2005.

[38] Onur Acıiçmez. Yet another microarchitectural attack: Exploiting I-Cache. In *CSAW*, 2007.

[39] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *USENIX Security*, 2014.

[40] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. Drive-by key-extraction cache attacks from portable code. In *ACNS*, 2018.

[41] Yuval Yarom. Mastik: A micro-architectural side-channel toolkit. `https://cs.adelaide.edu.au/~yval/Mastik/Mastik.pdf`, 2017.

[42] Onur Acıiçmez, Shay Gueron, and Jean-Pierre Seifert. New branch prediction vulnerabilities in OpenSSL and necessary software countermeasures. In *IMA Int. Conf.*, 2007.

[43] Dmitry Evtyushkin, Ryan Riley, Nael B. Abu-Ghazaleh, and Dmitry Ponomarev. BranchScope: A new side-channel attack on directional branch predictor. In *ASPLOS*, 2018.

[44] Yuval Yarom, Daniel Genkin, and Nadia Heninger. Cachebleed: A timing attack on openssl constant time RSA. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 346–367. Springer, 2016.

[45] Billy Bob Brumley and Risto M. Hakala. Cache-timing template attacks. In *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 667–684. Springer, 2009.

[46] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. Cache template attacks: Automating attacks on inclusive last-level caches. In *USENIX Security*, 2015.

[47] Changhua He and John C Mitchell. Analysis of the 802.1 i 4-Way handshake. In *WiSe*. ACM, 2004.

[48] John Mitchell and Changhua He. Security analysis and improvements for IEEE 802.11i. In *NDSS*, 2005.

[49] Changhua He, Mukund Sundararajan, Anupam Datta, Ante Derek, and John C Mitchell. A modular correctness proof of IEEE 802.11i and TLS. In *CCS*, 2005.

[50] Mathy Vanhoef and Frank Piessens. Predicting, decrypting, and abusing WPA2/802.11 group keys. In *USENIX Security*, 2016.

[51] Mathy Vanhoef and Frank Piessens. Practical verification of WPA-TKIP vulnerabilities. In *ASIA CCS*, pages 427–436. ACM, 2013.

[52] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. Practical side-channel attacks against wpa-tkip. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*, 2019.

[53] Mathy Vanhoef, Nehru Bhandaru, Thomas Derham, Ido Ouzieli, and Frank Piessens. Operating channel validation: Preventing multi-channel man-in-the-middle attacks against protected Wi-Fi networks. In *WiSec*, 2018.

[54] Christopher P Kohlios and Thaier Hayajneh. A comprehensive attack flow model and security analysis for Wi-Fi and WPA3. 2018.

[55] John Bellardo and Stefan Savage. 802.11 denial-of-service attacks: real vulnerabilities and practical solutions. In *USENIX Security*, 2003.

[56] Bastian Könings, Florian Schaub, Frank Kargl, and Stefan Dietzel. Channel switch and quiet attack: New DoS attacks exploiting the 802.11 standard. In *LCN*, 2009.

[57] Matthias Schulz, Francesco Gringoli, Daniel Steinmetzer, Michael Koch, and Matthias Hollick. Massive reactive smartphone-based jamming using arbitrary waveforms and adaptive power control. In *WiSec*, 2017.

[58] Kemal Bicakci and Bulent Tavli. Denial-of-service attacks and countermeasures in IEEE 802.11 wireless networks. *Comput. Stand. Interfaces*, 31(5), 2009.

[59] William Aiello, Steven M. Bellovin, Matt Blaze, John Ioannidis, Omer Reingold, Ran Canetti, and Angelos D. Keromytis. Efficient, DoS-resistant, secure key exchange for internet protocols. In *CCS*, 2002.

[60] Scott Fluhrer. Re: [Cfrg] I-D for password-authenticated EAP method. Retrieved 9 November 2018 from `https://www.ietf.org/mail-archive/web/cfrg/current/msg02206.html`, February 2008.

[61] Trevor Perrin. [TLS] review of Dragonfly PAKE. Retrieved 9 September 2018 from `https://www.ietf.org/mail-archive/web/tls/current/msg10922.html`, December 2013.

[62] D. Clarke and F. Hao. Cryptanalysis of the dragonfly key exchange protocol. *IET Information Security*, 8(6):283–289, 2014.

[63] Eman Alharbi, Noha Alsulami, and Omar Batarfi. An enhanced Dragonfly key exchange protocol against offline dictionary attack. *Journal of Information Security*, 6(02):69, 2015.

[64] Steven M Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Computer Society Symposium on Research in Security and Privacy*. IEEE, 1992.

[65] Michael Steiner, Gene Tsudik, and Michael Waidner. Refinement and extension of encrypted key exchange. *ACM SIGOPS Operating Systems Review*, 29(3):22–30, 1995.

[66] David P Jablon. Strong password-only authenticated key exchange. *ACM SIGCOMM Computer Communication Review*, 26(5):5–26, 1996.

[67] Thomas D Wu et al. The secure remote password protocol. In *NDSS*, volume 98, pages 97–111. Citeseer, 1998.

[68] SeongHan Shin, Kazukuni Kobara, and Hideki Imai. Security proof of AugPAKE. *IACR Cryptology ePrint Archive*, 2010:334, 2010.

[69] Stanislaw Jarecki, Hugo Krawczyk, and Jiayu Xu. Opaque: An asymmetric pake protocol secure against pre-computation attacks. Cryptology ePrint Archive, Report 2018/163, 2018. `https://eprint.iacr.org/2018/163`.

[70] José Becerra, Dimiter Ostrev, and Marjan Škrobot. Forward secrecy of SPAKE2. In *International Conference on Provable Security (ProvSec)*. Springer, 2018.

[71] T. Wu. The SRP authentication and key exchange system. RFC 2945, September 2000.

[72] D. Taylor, T. Wu, N. Mavrogiannopoulos, and T. Perrin. Using the secure remote password (SRP) protocol for TLS authentication. RFC 5054, September 2007.

[73] S. Shin and K. Kobara. Efficient augmented password-only authentication and key exchange for IKEv2. RFC 6628, June 2012.

[74] S. Smyshlyaev, E. Alekseev, I. Oshkin, and V. Popov. The security evaluated standardized password-authenticated key exchange (SESPAKE) protocol. RFC 8133, March 2017.

[75] F. Hao. J-PAKE: Password-authenticated key exchange by juggling. RFC 8236, September 2017.

[76] Moni Naor, Benny Pinkas, and Eyal Ronen. How to (not) share a password: Privacy preserving protocols for finding heavy hitters with adversarial behavior. *IACR Cryptology ePrint Archive*, 2018:3, 2018.

[77] Dennis Kügler. Re: [IPsec] PAKE selection: SPSK. Retrieved 23 April 2019 from `https://mailarchive.ietf.org/arch/msg/ipsec/NEicYFDYJYcQuNdknY0etLyfITA`, May 2010.

[78] Kevin M. Igoe. [Cfrg] status of DragonFly. Retrieved 8 November 2018 from `https://www.ietf.org/mail-archive/web/cfrg/current/msg03258.html`, December 2012.

[79] Kevin M. Igoe. [Cfrg] status of DragonFly. Retrieved 8 November 2018 from `https://www.ietf.org/mail-archive/web/cfrg/current/msg03261.html`, December 2012.

[80] Rene Struik. Re: [cfrg] small editorial error in and question on draft-irtf-cfrg-dragonfly-01 (was: Re: CFRG meeting at IETF 87). Retrieved 10 April 2019 from `https://mailarchive.ietf.org/arch/msg/cfrg/Z-nnOKTA4ddmFd17l5KzlRwWm5Y`, July 2013.

Appendix

Listing 3: Hash-to-curve method of SAE.

```
static int sae_derive_pwe_ecc(
    struct sae_data *sae, const u8 *addr1,
    const u8 *addr2, const u8 *password,
    size_t password_len, const char *identifier) {
  ...
  /* Create a random quadratic residue (qr) and quadratic
   * non-residue (qnr) mod p for blinding purposes. */
  if (get_random_qr_qnr(prime, prime_len, sae->tmp->prime,
                        bits, &qr, &qnr) < 0)
      return -1;
  ...
  for (counter = 1; counter <= k || !x; counter++) {
    ...
    res = sae_test_pwd_seed_ecc(sae, pwd_seed, prime
                                qr, qnr, &x_cand);
    if (res < 0) goto fail;
    if (res > 0 && !x) {
      ...
      x = x_cand; /* saves the current x value */
      ...
      /* Use a dummy password for the following
       * rounds, if any. */
      addr[0] = dummy_password;
      len[0] = dummy_password_len;
    } else if (res > 0)
      crypto_bignum_deinit(x_cand, 1);
  }
  ...
```

Listing 4: Assembly output of SAE's hash-to-curve.

```
000000000002efe0 <sae_derive_pwe_ecc>:
...
2f2c8:  e8f3170500   callq  80ac0 <sha256_prf_bits>
...

2f719:  e8f2fa0400   callq  7f210 <crypto_bignum_legendre>
...
2f751:  e81af70400   callq  7ee70 <crypto_bignum_deinit>
2f75d:  0f8559010000  jne
  2f8bc <sae_derive_pwe_ecc+0x8dc>
...
; handle qr case code range
2f7d2:  0f8660faffff  jbe
  2f238 <sae_derive_pwe_ecc+0x258>
...
; start nqr case code
2f8bc:  488b7c2440   mov    0x40(%rsp),%rdi
2f8c1:  be01000000   mov    $0x1,%esi
2f8c6:  e8a5f50400   callq  7ee70 <crypto_bignum_deinit>
2f8cb:  e994faffff   jmpq
  2f364 <sae_derive_pwe_ecc+0x384>
; end nqr case code
...
```