The Dragonfly Protocol

COMP5900H Research Summary

William Findlay

October 3, 2020

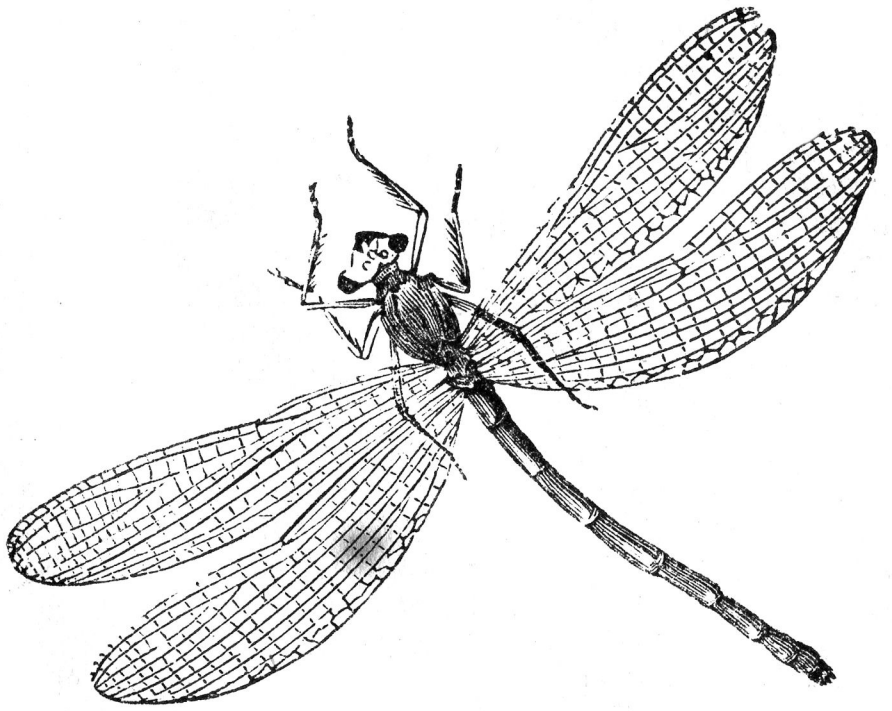

Image of dragonfly provided royalty free by https://thegraphicsfairy.com

1 Introduction

Wireless networks are ubiquitous in the 21^{st} century; from home networks to enterprise infrastructure, wireless communication permeates nearly every aspect of modern life. The nature of wireless communication implies that these networks lack many of the inherent physical security properties present in wired networks such as Ethernet [8]. This motivates the need for secure protocols with strong cryptographic and security guarantees, particularly with respect to mutual authentication and key establishment between parties.

Motivation for Dragonfly in 802.11 WLANs Early attempts at securing wireless local area network (WLAN) communication largely resulted in failure (e.g., WEP) [8] and it was very quickly recognized that better protocols would be needed to establish secure wireless communication between parties. After WEP, WPA was introduced, which provided provisional support for TKIP, an intermediate protocol designed to run entirely in software (and thus support the original hardware that had been designed for WEP) [8]. Soon after, WPA2 was introduced which included support for the more secure AES-CCMP for encryption and authentication. However, WPA2 lacked a sophisticated key exchange and pre-association authentication method, and so SAE and EAP-pwd (both using Dragonfly) were introduced into WPA3 to fill this role [8]. Dragonfly's strong security guarantees (see Section 2.2) made it ideal for use in WPA3.

Outline The rest of this paper proceeds as follows. Section 2 presents the specifications for the Dragonfly protocol as described in [4]. Section 3 summarizes other Dragonfly variants that are in use explains the motivating factors that led to their design. Section 4 concludes.

2 Dragonfly Specifications

In this section, I provide a finite field cryptography explanation of the Dragonfly specification presented in [4], using supplementary information from [3, 6, 9]. Section 2.1 discusses the details of the protocol and Section 2.2 discusses its cryptographic properties and security goals.

2.1 Finite Field Cryptography Dragonfly

The Dragonfly key exchange protocol [4] supports both *finite field cryptography* (FFC) and *elliptic curve cryptography* (ECC) versions. For simplicity, this paper focuses on the finite field version. The elliptic curve version will not be discussed further, except to provide context where necessary.

The security of finite field cryptography relies on a computationally difficult problem known as the *discrete log problem*. Formally, the discrete log problem can be characterized as follows: given some sufficiently large prime number p, a generator G over the group $\mathbb{Z}_p^*$, and some $y = G^x \pmod p$, it is thought to be computationally infeasible to compute the value of x. Here, $\mathbb{Z}_p^*$ is defined as the multiplicative group of the set of integers modulo p and the generator G is an element of this set which can generate any other element in the set by being raised to some power [4, 7, 9].

In addition to finite field cryptography, Dragonfly makes use of a few functions [4] with desirable properties: a cryptographic hash function H, a mapping function F, and a key derivation function KDF. These functions and their relevant properties are summarized in Table 2.1.

The protocol itself consists of a setup phase (Section 2.1.1) and a commit-and-confirm key exchange (Section 2.1.2). Figure 2.1 depicts an overview of the entire key exchange.

2.1.1 The Dragonfly Setup Phase

Before beginning the key exchange phase, both parties are expected to have knowledge of a shared password (distributed out-of-band) to be used in key derivation. Additionally, they are expected to have agreed upon the functions outlined in Table 2.1 as well as a set of *domain parameters* defining the group under which finite field cryptography will occur [3, 4, 6]. Specifically, each party must agree on a large prime p, a generator $G \in \mathbb{Z}_p^*$, and a large prime q such that $G^q \equiv 1 \pmod p$. Here, q is referred to as the *multiplicative order* of G and should be a sufficiently large prime divisor of $\frac{p-1}{2}$ [4].

Table 2.1: Functions used in Dragonfly key exchange along with their relevant properties. Note that $F(x)$ here is the identity function. In ECC Dragonfly, $F(x)$ is used to map points on the elliptic curve to integers. This mapping is not necessary in FFC Dragonfly [4], and so this function is only included to preserve interface compatibility with ECC Dragonfly.

Function	Properties
$H(x)$	• Maps x to a deterministic, pseudorandom output y of fixed size (cryptographic hash) • Computationally infeasible to reverse y back to x
$F(x)$	• The identity function; maps x to x
$KDF_n(k, label)$	• Takes a key, k and some *label* as input • The label should be a string that specifies the current use case • Outputs a new stretched key of length n

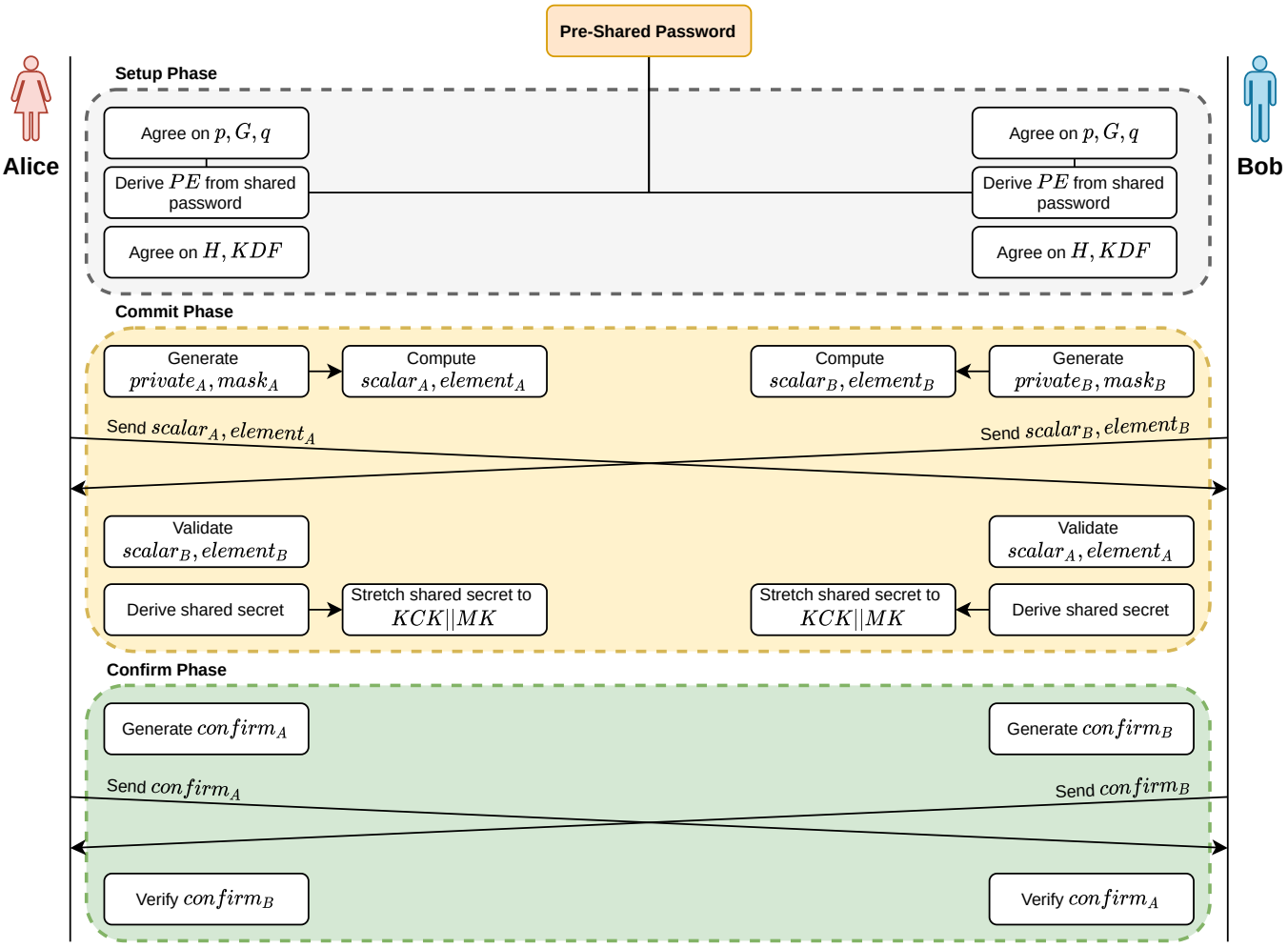

Figure 2.1: An overview of the phases of the Dragonfly key exchange protocol. This figure was created based on information provided in [4]. Note that values computed during the protocol run are considered ephemeral.

Once each party has agreed on the functions and domain parameters to be used, they must individually compute the *password element* (PE), a unique member of the group $\mathbb{Z}_p^*$ mapped from the shared password [3, 4, 6, 9]. To compute PE, both parties employ an iterative strategy until they find a valid group member (i.e., $PE \in \mathbb{Z}_p^*$). This iterative strategy involves the following steps [4]:

(1) Initialize a *counter* to 1.

(2) Compute a *base* by taking the concatenation of both party's identities, the shared password, and the counter initialized in step (1) and passing this new value to the cryptographic hash function $H(x)$. That is:

$$base = H(ID_1 \mid\mid ID_2 \mid\mid password \mid\mid counter)$$

where $ID_1 > ID_2$ (for the purpose of establishing a canonical ordering between the peers) and $\mid\mid$ is the concatenation operator.

(3) Compute a *seed* by taking the *base* computed in step (2) and passing it into the key derivation function KDF_n along with a chosen usage-specific label. This *seed* value is then incremented by 1 (mod $p-1$). Thus, we have:

$$seed = KDF_n(base, label) + 1 \pmod{p-1}$$

(4) Compute a candidate PE as follows:

$$PE_{candidate} = seed^{\frac{p-1}{q}} \pmod{p}$$

(5) If the candidate PE is a value higher than 1, take it as the PE. Otherwise, increment the counter and go back to step (2).

In order to preclude possible timing attacks on the PE generation process, the Dragonfly specification stipulates that implementations should go through a recommended $k \geq 40$ iterations regardless of whether or not they find a valid PE and that it should be highly improbable that the search for a valid PE will exceed k iterations [4]. Once the PE has been fixed by both parties, they are ready to move on to the actual key exchange phase.

2.1.2 The Dragonfly Key Exchange Phase (Commit and Confirm)

Dragonfly's key exchange phase consists of two sub-phases: the commit phase, and the confirm phase. Here, I discuss each phase in turn. At the end of both phases, the parties will have successfully created two shared session keys—a *key confirmation key* and *master key*—as well as having authenticated by demonstrating knowledge of the *key confirmation key* [4].

The Commit Phase The commit phase in Dragonfly is used to demonstrate mutual knowledge of the shared password, with each party committing to a single guess [4]. Each party computes two values, *scalar* and *element*, based on freshly generated random numbers and exchanges them with the other party. These random numbers are *private* and *mask*, which are used to generate *scalar* and *element* as follows [4]:

$$scalar = private + mask \pmod{q}$$
$$element = inv(PE^{mask}) \pmod{p}$$

where $inv(x)$ is the modular inverse operation modulo p. That is, $x \times inv(x) \equiv 1 \pmod{p}$. Values of $scalar < 2$ are considered invalid and the Dragonfly protocol mandates that, if this occurs, both values must be re-computed with a different *private* and *mask* [4]. Once these values have been exchanged between

parties, a validation step ensures that received values are not identical to the recipient's own values and that the received *element* is a valid member of a group $\mathbb{Z}_p^*$.

If validation succeeds, the peer-received values are then taken together with *private* to generate a shared secret as follows [4]:

$$secret = F\left(\left(element_{peer} \times PE^{scalar_{peer}}\right)^{private}\right) \pmod p$$

where $element_{peer}$ and $scalar_{peer}$ are the values received from the peer (not the recipient's own values) and *private* is the recipient's own random value. Note that all operations here are done modulo p. Recall that $F(x)$ is simply the identity function, so in effect it can be ignored.

After the shared secret has been generated, it is stretched into a new secret of a bit-length double that of p's bit-length. This stretching occurs with the help of the key derivation function, KDF, as follows [4]:

$$KDF_n(secret, label)$$

where label is an identifying label, different from the label used in the setup phase and n is the required bit-length. This stretched shared secret is then split into two ephemeral, shared keys of equal length: the *key confirmation key* (KCK) used for authentication and the *master key* (MK) used to derive session keys [4].

The Confirm Phase In the confirmation phase, both parties demonstrate that they have knowledge of the shared secret generated in the previous step and use this knowledge to authenticate with each other [4, 9]. To accomplish this, each party combines several values using the cryptographic hash function function, H, as follows [4]:

$$confirmation_{self} = H(KCK \mid\mid scalar_{self} \mid\mid scalar_{peer} \mid\mid element_{self} \mid\mid element_{peer} \mid\mid ID_{self})$$

where KCK is the key confirmation key and $\mid\mid$ is the concatenation operator. Each party transmits their confirmation value to the other, after which the other party performs the same operation to compute the other's confirmation and compares these values to make sure they match. If all values match, authentication has succeeded [4].

2.2 Cryptographic Properties and Security Goals

The Dragonfly specification [3, 4, 6] was designed with several cryptographic properties and security goals in mind, namely forward secrecy, known-key security, resistance to both active and passive attacks, and resistance to dictionary attacks. I discuss them here in turn.

Forward Secrecy The property of *forward secrecy* [6, 7] refers to a security guarantee on previous session keys in the presence of long-term key exposure. In particular, the exposure of the long term key to an unauthorized party should not permit the unauthorized party to gain any extra information about prior session keys or prior communications secured using these session keys. This property necessitates that the session key must be derived with the help of some additional keying material beyond the long-term key and that both the session key itself and the keying material used to derive the session key must be ephemeral [7]. Dragonfly provides forward secrecy by requiring that both parties contribute a random value to the shared secret before it is used to generate the session key. This value is only known to the contributing party [3]. Naturally, this forward secrecy is based on the security of the underlying cryptographic hash function used to compute the contributions for each party and the bit-length of its output [6].

Known-Key Security In order for a key establishment protocol to have *known-key security* [7], a compromised session key or keys should not compromise the security of future sessions. That is, if an unauthorized party gains access to some session key k_i, future session keys k_j where $j > i$ should not be compromised. Known-key security in Dragonfly is based on the cryptographic properties of the one-way random function, H (discussed in detail in Section 2.1), and the random contributions made to the shared secret by both parties [3, 6].

Resistance to Active Attacks An *active attack* is defined as an attack in which an adversary actively interferes with or modifies the messages sent between two legitimate parties or forges messages and sends them to a legitimate party. Dragonfly resists such attacks by relying on the computational infeasibility of forging the output of the cryptographic hash function H, the randomness of per-party contributions to the shared secret, and the difficulty of finding the value q given PE (i.e., the discrete log problem) [3, 6].

Resistance to Passive Attacks In contrast to an active attack, a passive attack occurs when an adversary passively observes communications between two legitimate parties without directly interfering with their communication in any way. Success here means inhibiting an adversary's ability to learn the shared secret or pre-shared password through passive observation. Here, Dragonfly relies on the security of the *private* random value (since it is masked modulo q before being sent) and the inability of the adversary to verify a guessed password without knowing KCK [3, 6].

Resistance to Offline Dictionary Attacks An offline dictionary attack on a password-based protocol such as Dragonfly involves the attacker iterating through a list of known-likely passwords and using this list as a means to generate reasonable guesses for the password. These guesses would then be checked offline against captured messages. Dragonfly resists such attacks by providing no way to verify the validity of a guess without directly communicating with the victim [4]. It should be noted, however, that side-channel attacks presented in [9] do allow some brute-forcing attacks that make use of an offline dictionary.

3 Dragonfly Variants

The official Dragonfly specification forms the basis for SAE [3], a password-based method for key exchange, originally designed for peer-to-peer mesh networks and eventually incorporated into the 802.11-2012 standard [8]. In WPA3, SAE is used to negotiate a Dragonfly master key and this key is then used in a four-message handshake to establish a WPA session key [8, 9]. Dragonfly was also incorporated as a password-based EAP authentication method for enterprise networks, EAP-pwd [6]. Rather than specifying a handshake between peers, EAP-pwd instead defines the protocol in terms of communication with an authentication server over 802.1x frames [6, 8, 9].

A variant of EAP-pwd that modifies the Dragonfly exchange to the use of salted password hashes was introduced in [5]. The motivation here was primarily integration with enterprise databases that incorporate salting into their password hashing scheme. These salted password hashes would provided added protection against dictionary attacks, especially when re-used with other protocols that may not necessarily be resistant to such attacks [5].

Clark and Hao [2] published a cryptanalysis of the original Dragonfly circa 2013 that revealed an offline dictionary attack on the protocol despite its security goals to the contrary. The proposed solution introduced a validation step which was incorporated into later versions of the protocol (c.f. Section 2.1). Alharbi et al. [1] proposed a Dragonfly variant using two pre-shared passwords that defeats such offline dictionary attacks while significantly improving performance.

A detailed security analysis of Dragonfly was presented by Vanhoef and Ronen [9], including several side channel attacks on the protocol. Vanhoef and Ronen suggest a few simple modifications to mitigate these attacks that have culminated in new drafts of the affected protocols [9].

4 Conclusion

This paper has presented the design of the Dragonfly protocol [4] used in SAE [3] and EAP-pwd [5, 6] as part of WPA3. Despite the careful security considerations incorporated into Dragonfly's original design, security researchers have revealed flaws in the original design that have motivated the development of alternative and updated versions of the protocol. While many of these updates have been incorporated into the canonical versions of Dragonfly running in SAE and EAP-pwd, recent work [9] has revealed that one must never become complacent when designing such protocols in practice.

References

[1] E. Alharbi, N. Alsulami, and O. Batarfi, "An Enhanced Dragonfly Key Exchange Protocol against Offline Dictionary Attack," in *Journal of Information Security*, vol. 6, 2015, pp. 69–81. DOI: 10.4236/jis.2015.62008.

[2] D. Clarke and F. Hao, "Cryptanalysis of the Dragonfly key exchange protocol," in *IET Information Security*, vol. 8, 2015, pp. 283–289. DOI: 10.1049/iet-ifs.2013.0081.

[3] D. Harkins, "Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks," in *2008 Second International Conference on Sensor Technologies and Applications (sensorcomm 2008)*, IEEE, Cap Esterel, France, 2008, pp. 839–844. DOI: 10.1109/SENSORCOMM.2008.131.

[4] D. Harkins, "Dragonfly Key Exchange," RFC 7664, Nov. 2015, Informational RFC. [Online]. Available: https://tools.ietf.org/html/rfc7664.

[5] D. Harkins, "Adding Support for Salted Password Databases to EAP-pwd," RFC 8146, Apr. 2017, Informational RFC. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8146.html.

[6] D. Harkins and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password," RFC 5931, Aug. 2010, Informational RFC. [Online]. Available: https://tools.ietf.org/html/rfc5931.

[7] P. C. van Oorschot, "Authentication Protocols and Key Establishment," in *Computer Security and the Internet: Tools and Jewels*. Springer, 2019, ch. 4, pp. 92–124, Author's archived copy, for personal use.

[8] P. C. van Oorschot, "Wireless LAN Security: 802.11 and Wi-Fi," in *Computer Security and the Internet: Tools and Jewels*. Springer, 2020, ch. 12, pp. 340–370, Pre-print distributed for COMP5900H.

[9] M. Vanhoef and E. Ronen, "Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy-S&P 2020)*, IEEE, 2020. [Online]. Available: http://papers.mathyvanhoef.com/dragonblood.pdf.