Cryptanalysis of the Dragonfly Key Exchange Protocol

Dylan Clarke, Feng Hao*
School of Computing Science
Newcastle University
{dylan.clarke, feng.hao}@ncl.ac.uk

February 5, 2013

Abstract

Dragonfly is a password authenticated key exchange protocol that has been submitted to the Internet Engineering Task Force as a candidate standard for general internet use. We analyzed the security of this protocol and devised an attack that is capable of extracting both the session key and password from an honest party. This attack was then implemented and experiments were performed to determine the time-scale required to successfully complete the attack.

1 Introduction

Dragonfly is a password authenticated key exchange protocol specified by Dan Harkins for exchanging session keys with mutual authentication within mesh networks [1]. Recently, Harkins submitted a variant of the protocol to the Internet Engineering Task Force (IETF) as a candidate standard for general Internet use[1]. We observe that both variants are essentially the same protocol, though some implementation details are different.

It is claimed that the Dragonfly protocol is resistant to active attacks, passive attacks, and off-line dictionary attacks [1, 2]. However, as acknowledged by the author [1], no security proofs are given to support the claim. The lack of security proofs has raised some concerns among members on the IETF mailing list[2]. However, to our best knowledge, no one has presented concrete attacks.

In this paper, we examine the security properties of the Dragonfly protocol. Contrary to the author's claims, we show that both variants are subject to an off-line dictionary attack. In this paper, we will base our analysis upon the

*Supported by EPSRC First Grant RES/0550/7300
[1] `https://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/history/`
[2] `http://comments.gmane.org/gmane.ietf.irtf.cfrg/1786`

original protocol specification as defined in a peer-reviewed paper [1]. However, the attack we will present is trivially applicable to the variant specified in [2]. (According to the Dragonfly author, the current Internet draft, which expires on April 15, 2013 [2], will be changed soon in light of our reported attack.)

2 The Dragonfly Protocol

Dragonfly is based on discrete logarithm cryptography. This means that an implementation of Dragonfly can either use operations on a finite field or an elliptic curve. No assumptions are made about the underlying group, other than that the computation of discrete logarithms is sufficiently computationally difficult for the level of security required. In each case, there are two operations that can be performed: an element operation that takes an input of two elements and outputs a third element, and a scalar operation that takes an input of an element and a scalar and outputs an element.

We take the finite field as an example. Let us define p a large prime. We denote a finite cyclic group Q, which is a subgroup of Z_p^* of prime order q. Hence, $q \,|\, p-1$. We denote the element operation $A.B$ for elements A and B, and the scalar operation A^b for element A and scalar b. These notations are in line with those commonly used when working over a finite field.

The Dragonfly protocol works as follows (also see [1]):

- Alice, Bob have a shared password from which each can deterministically generate a password element $P \epsilon Q$. The algorithms to map an arbitrary password to an element in Q are specified in [1] and [2]. However, the details are not relevant to our attack, so they are omitted here.
- Alice randomly chooses two scalars r_A, m_A from 1 to q, calculates the scalar $s_A = r_A + m_A \mod q$ and the element $E_A = P^{-m_A} \mod p$ and sends s_A, E_A to Bob.
- Bob randomly chooses two scalars r_B, m_B from 1 to q, calculates the scalar $s_B = r_B + m_B \mod q$ and $E_B = P^{-m_B} \mod p$ and sends s_B, E_B to Alice.
- Alice calculates the shared secret $ss = (P^{s_B} E_B)^{r_A} = P^{r_A r_B} \mod p$
- Bob calculates the shared secret $ss = (P^{s_A} E_A)^{r_B} = P^{r_A r_B} \mod p$
- Alice sends $A = H(ss|E_A|s_A|E_B|s_B)$ to Bob where H is a predefined cryptographic hash function
- Bob sends $B = H(ss|E_B|s_B|E_A|s_A)$ to Alice
- Alice and Bob check that the hashes are correct and if they are then they create a shared key $K = H(ss|E_A \cdot E_B|(s_A + s_B)\, mod\, q)$

This is illustrated in Figure 1.

Figure 1: The Dragonfly Protocol

	Alice		Bob
	$P \epsilon Q$		$P \epsilon Q$
1.	$r_A, m_A \epsilon \{1, \dots, q\}$		$r_B, m_B \epsilon \{1, \dots, q\}$
2.	$s_A = r_A + m_A$		$s_B = r_B + m_B$
3.	$E_A = P^{-m_A}$	$\xrightarrow{s_A, E_A}$	$E_B = P^{-m_B}$
4.		$\xleftarrow{s_B, E_B}$	
5.	$ss = (P^{s_B} E_B)^{r_A}$	$\xrightarrow{A = H(ss \vert E_A \vert s_A \vert E_B \vert s_B)}$	Verify A
	$= P^{r_B r_A}$		$ss = (P^{s_A} E_A)^{r_B}$
6.	Verify B	$\xleftarrow{B = H(ss \vert E_B \vert s_B \vert E_A \vert s_A)}$	$= P^{r_A r_B}$
7.	Compute the shared key: $K = H(ss \vert E_A \cdot E_B \vert (s_A + s_B) \, mod \, q)$		

3 A Small Subgroup Attack on Dragonfly

3.1 Attack Methodology

It is claimed in [1] that the Dragonfly protocol is resistant to offline dictionary attacks. However, no security proofs are given. Instead, the author provides a heuristic security analysis as follows. It is assumed that an active attacker would select an arbitrary value for m_B and compute $E_B = G^{m_B}$ where G is the group generator for Q. Then, the attacker would receive a hash value for which the only unknown input to the hash function is z where $P = G^z$. Therefore, for an offline dictionary attack to be successful, the attacker would have to be able to compute z for a random element in Q, which contradicts the assumption that discrete logarithms are hard to compute.

We point out that computing $E_B = G^{m_B}$ is not the best option available to an active attacker. Instead, the attacker can use the following method, summarized in Figure 2. First, the attacker computes $E_B = S_n$ where S_n is the generator of a small subgroup of Z_p^* of order n. Then, the shared secret computed by Alice is $ss = (P^{s_B} \cdot S_n)^{r_A} = P^{s_B r_A} \cdot S_n^{r_A}$, and this is the only unknown value on which the hash sent by Alice is dependent.

The attacker then uses Algorithm 1: 1) to obtain the victim's password element P; 2) to forge a valid response B to bypass authentication (so the victim is unaware that the password has been compromised); 3) to compromise the secrecy of communication by deriving the session key K.

This attack will be feasible as S_n generates a small subgroup and the password space is sufficiently small to permit dictionary attacks. In Algorithm 1 (line 5), following $A = A'$, we will have $ss = ss'$ because the hash is assumed to be a random oracle and is collision resistant. Thus, we obtain:

$$P^{s_B r_A} S_n^{r_A} = (P'^{s_A} E_A)^{s_B} \cdot R_x \tag{1}$$

where R_x is a (yet unknown) small subgroup element.

Figure 2: Small Subgroup Attack

	Alice		Bob (Attacker)
	$P \epsilon Q$		
1.	$r_A, m_A \epsilon \{1, \ldots, q\}$		$s_B \epsilon \{1, \ldots, q\}$
2.	$s_A = r_A + m_A$		$E_B = S_n$
3.	$E_A = P^{-m_A}$	$\xrightarrow{s_A, E_A}$	
4.		$\xleftarrow{s_B, E_B}$	
5.	$ss = (P^{s_B} E_B)^{r_A}$	$\xrightarrow{A = H(ss \vert E_A \vert s_A \vert E_B \vert s_B)}$	$(P, B, K) \leftarrow$
	$= P^{s_B r_A} S_n^{r_A}$		OfflineSearch(A, s_B, E_B)
6.	Verify B	$\xleftarrow{B = H(ss \vert E_B \vert s_B \vert E_A \vert s_A)}$	
7.	Compute the shared key $K = H(ss \vert E_A \cdot E_B \vert (s_A + s_B) \, mod \, q)$		

Algorithm 1 OfflineSearch algorithm

Input: A, s_B, E_B
Output: P, B, K

```
for each P' in dictionary do
    for each R_x in the subgroup do
        ss' := (P'^{s_A} E_A)^{s_B} · R_x
        A' := H(ss'|E_A|s_A|E_B|s_B)
        if A = A' then
            P = P'
            B = H(ss'|E_B|s_B|E_A|s_A)
            K = H(ss'|E_A · E_B|(s_A + s_B) mod q)
            Return {P, B, K}
        end if
    end for
end for
```

After re-arranging the terms, we obtain:

$$\frac{P^{s_B r_A}}{(P'^{s_A} E_A)^{s_B}} = \frac{R_x}{S_n^{r_A}} \tag{2}$$

Notice that the term on the left is an element in a subgroup of prime order q while the term on the right is an element in a small subgroup of order n. Since $q \neq n$, the equality holds only when both sides are identity elements in Z_p^*: i.e., 1. Therefore, $(P'^{s_A} E_A)^{s_B} = P^{r_A s_B}$, from which the only possible value for p' is $p' = p$. After successfully obtaining the victim's password, the attacker is able to easily forge a valid response and send it back to Alice, so Alice is unaware that her password has been compromised. Finally, the attacker can derive the shared session key and engage with Alice in the subsequent secret communication based on that session key.

3.2 Attack Implementation

We implemented an attack simulation in Java. The simulation consisted of three components: the password chooser that randomly chose a dictionary of password elements, the honest party who randomly chose one of these elements as a password and performed the Dragonfly protocol in an honest manner, and the dishonest party who performed the dictionary attack against the honest party.

We ran the Dragonfly protocol in a 160-bit subgroup of a 1024-bit finite group. The group parameters are specified in Appendix A. They are originally from the standard NIST cryptographic toolkit[3]. However, the NIST toolkit does not publish the small subgroups. Hence, we began by using a brute force method to determine the prime factors of $p - 1$ (where p is the prime modulus of the 1024 bit group). In the experiment, we only searched for prime factors of size less than 32 bits. We have found the following prime factors: 2, 3, 13, 23 and 463907. Accordingly, we calculated generators for each of the corresponding small subgroups (see Appendix B) and performed a set of experiments to determine the time to complete an offline dictionary attack for each subgroup.

Each set of experiments involved mounting the attack with dictionaries of 1000, 10000 and 100000 random password elements. The different dictionary sizes allowed us to measure how an increase in dictionary size would affect the time taken to complete the attack. In all cases, the time measured was the time to try every possible password, rather than the time until the correct password was discovered. Each experiment was performed 30 times.

3.3 Results

We note that only one possible password was identified in every experiment and this was the password chosen by the honest party. The times taken to check all possible passwords with a subgroup of size 463907 as dictionary size varies

[3] `http://csrc.nist.gov/groups/ST/toolkit/documents/Examples/DSA2_All.pdf`

Table 1: Experiments for a Subgroup of Size 463907

Dictionary Size	Mean Time to Try All Passwords (ms)	Std Dev
1000	16894592	146428
10000	169475627	4527601
100000	1693389098	72654423

Table 2: Experiments with a Dictionary Size of 1000

Subgroup Size	Mean Time to Try All Passwords (ms)	Std Dev
2	5653	52
3	5655	71
13	6319	82
23	7700	188
463907	16894592	146428

are shown in Table 1. This illustrates that there is a fairly linear relationship between dictionary size and the time taken to try all passwords, and also that the attack is still feasible for a relatively large dictionary size. The times taken to check all possible passwords with a dictionary size of 1000 as the subgroup size varies are shown in Table 2. In all cases the experiments were run under Windows 7 on a 2.9GHz PC with 4GB of memory.

We note that some of the times measured are sufficiently large that Alice may terminate the protocol due to the large time taken for the attacker to respond. However, there are also three mitigating factors to consider: 1) We have measured the mean time to try all passwords, in practice we would expect the attacker to find the correct password without having to try all possibilities; 2) An attacker is likely to have the resources to distribute the calculations over several high performance machines, reducing the calculation time significantly; 3) Even if the protocol is terminated, the attacker will have discovered the password and may be able to make use of it in another run of the protocol.

4 Discussion

4.1 Preventing Small Subgroup Attacks on the Dragonfly Protocol

Small subgroup attacks can be prevented by checking that the received element E (more specifically, E_A for Bob and E_B for Alice) is a member of the group being used by the cryptographic scheme. This can be achieved by checking that E is member of the supergroup, that E is not the identity element and that E^q is equal to the identity element. The importance of this check – known as the public key validation – in key exchange protocols has been highlighted by Menezes and Ustagolu [4].

Table 3: Consequence of attack if public key validation is missing

Consequence of small subgroup attack	Dragonfly	SPEKE
Success in guessing the password offline	Yes	No
Success in impersonation	Yes	Yes
Success in eavesdropping secure communication	Yes	Yes

However, to validate a public key will require a full exponentiation over the finite group, which will significantly decrease the protocol efficiency and make it less appealing than its competitors. For this reason, it remains debatable within the cryptographic community if the public key validation is indispensable. Nonetheless, at least for the case of the Dragonfly protocol, we have shown that the omission of public key validation renders the protocol completely insecure.

4.2 Comparison between Dragonfly and SPEKE

We observe that the Dragonfly protocol is very similar to SPEKE [3] with two minor changes. First, it drops the constraint in [3] that p must be a safe prime (i.e., $p = 2 \cdot q + 1$). Thus, it looks much more efficient than SPEKE since it can accommodate a short exponent, say a value of 160 bits instead of 1023 bits. (Given a fixed modulus p, the cost of exponentiation is linear to the bit-length of the exponent.) However, despite being efficient, the protocol is insecure for the attack we have demonstrated. If we add the cost of public key validation, Dragonfly will have no performance advantage over SPEKE.

Second, instead of sending just one single element by each participant as in SPEKE, Dragonfly adds an extra scalar in the flow. This slight change makes the protocol more complex than the original SPEKE. However, the rationale for this change is not explained in [1] or [2]. We observe that this extra complexity not only reduces the communication efficiency as the message size gets bigger, but also degrades security. To see this, let us assume there is no public key validation in both Dragonfly and SPEKE (so we can remove the effect of the first change, and only focus on studying the effect of the second change). Without the public key validation in SPEKE, an active attacker can confine the session key to an element in a small subgroup [3]. By brute force, the attacker can obtain the session key, thus defeating authentication and confidentiality in the secure communication. However, the attacker is unable to obtain the password. By contrast, in the case of Dragonfly, an active attacker is able to additionally obtain the victim's password (see Table 3). This observation serves to help better understand the underlying structural design of Dragonfly.

5 Conclusion

We have shown that the Dragonfly protocol is vulnerable to a small subgroup based offline dictionary attack. This attack can be prevented by adding a public key validation, which will however decrease the protocol efficiency. In the past three decades, many key exchange protocols have omitted public key validation, but are consequently found vulnerable to small subgroup confinement attacks despite the gained efficiency. Dragonfly is yet another example of this attack – but will not be the last.

Acknowledgment

We thank Professor Peter Ryan for first introducing us to analyzing the dragonfly protocol, and Dan Harkins for providing useful feedback.

References

[1] D. Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In *Sensor Technologies and Applications, 2008. SENSORCOMM '08. Second International Conference on*, pages 839 –844, aug. 2008.

[2] Dan Harkins. Dragonfly key exchange - internet research task force internet draft. `http://tools.ietf.org/html/draft-irtf-cfrg-dragonfly-00` accessed Jan 2013, 2012.

[3] David P. Jablon. Strong password-only authenticated key exchange. *ACM Computer Communications Review*, 26:5–26, 1996.

[4] Alfred Menezes and Berkant Ustaoglu. On the importance of public-key validation in the mqv and hmqv key agreement protocols. In *Proceedings of the 7th international conference on Cryptology in India*, INDOCRYPT'06, pages 133–147, Berlin, Heidelberg, 2006. Springer-Verlag.

A Group Parameters

The group parameters are taken from the NIST cryptographic toolkit using a 1024 bit modulus, and are shown in Table 4.

B Subgroup Generators

The subgroup generators shown in Table 5 are for subgroups of the multiplicative group with prime modulus defined in Appendix A. We only list subgroup sizes of up to 32 bits.

Table 4: Group Parameters

Parameter	Value (Base 16)
Prime Modulus	E0A67 598CD 1B763 BC98C 8ABB3 33E5D DA0CD 3AA0E 5E1FB 5BA8A 7B4EA BC10B A338F AE06D D4B90 FDA70 D7CF0 CB0C6 38BE3 341BE C0AF8 A7330 A3307 DED22 99A0E E606D F0351 77A23 9C34A 912C2 02AA5 F83B9 C4A7C F0235 B5316 BFC6E FB9A2 48411 258B3 0B839 AF172 440F3 25630 56CB6 7A861 158DD D90E6 A894C 72A5B BEF9E 286C6 B
Generator	D29D5 121B0 423C2 769AB 21843 E5A32 40FF1 9CACC 79226 4E3BB 6BE4F 78EDD 1B15C 4DFF7 F1D90 5431F 0AB16 790E1 F773B 5CE01 C804E 50906 6A991 9F519 5F4AB C5818 9FD9F F9873 89CB5 BEDF2 1B4DA B4F8B 76A05 5FFE2 77098 8FE2E C2DE1 1AD92 219F0 B3518 69AC2 4DA3D 7BA87 011A7 01CE8 EE7BF E4948 6ED45 27B71 86CA4 610A7 5
Subgroup Order	E9505 11EAB 424B9 A19A2 AEB4E 159B7 844C5 89C4F

Table 5: Subgroup Generators

Subgroup Size	Generator (Base 16)
2	E0A67 598CD 1B763 BC98C 8ABB3 33E5D DA0CD 3AA0E 5E1FB 5BA8A 7B4CA BC10B A338F AE06D D4B90 FDA70 D7CF0 CB0E6 38BC3 341BE C0AF8 A7330 A3307 DED22 99A0E E606D F0351 77A23 9C34A 912C2 02AA5 F83B9 C4A7C F0235 B5316 BFC6E FB9A2 48411 258B3 0B839 AF172 440F3 25630 56CB6 7A861 158DD D90E6 A894C 72A5B BEF9E 286C6 A
3	C644F AEA25 8D199 FA294 8F762 9C61F A38C5 FD02C 0629A AF401 B8F1C 11777 F1596 E8176 9FD81 DD69D E8A7A 58FF3 AF656 1947C 5317F FEC4E 3E396 C7229 978AD B14AA 96FB0 2D014 4A3B0 433BC D1C73 32DC2 5B3DB DAF68 E3622 0F311 5913D DC408 1E601 96196 E7405 53FBD 94083 128F5 34300 FA399 E71E8 B83C4 9590B 21C8E D2F4C 0
13	6F165 E1313 45256 75B6F 6C0FF 1BAAD 32513 77F34 AAB82 EDA7C E4D7C 85B50 10F81 22412 3FDFF F6CFB 8AFE78 3685FC 67D8D E91F0 CC70D B8340 DFE93 98295 D616B 4FE47 39C62 19D12 688A3 12CBE ECB53 F00E9 6B1FF 9B7DD 8308C 20CEA 82B7F 6FB98 B2D7E 9F581 D01B3 C94C1 074E5 8AED3 A1267 1C8EA AF994 C5742 24EC0 6A914 6E19
23	47DEC 28EB6 0A9BE 720D1 AD4E7 016AE DC162 27C88 755A7 E5259 A5B8E D02CF 76CB7 609CD 4869A 65BD7 5640D 36A30 BB1A4 63A34 A5B8D 5EB0E 29D83 2ADEA DF9D5 8ADF0 A0AA4 715F9 C6C62 0321F 47F0C E1C66 D3A65 66E66 818E5 552C6 0D8F9 EEF36 9144E F07E2 AED12 383D6 9D27D 6C898 0C6E2 D7700 7AD90 45A2D 55E54 DA1B9 05FC7 4
463907	16561 8E5D1 ED397 D8C7A 1D7A7 CB5DB 035DC 93586 DD6B6 B2670 D5FAE 4065E 6F7D7 B326C 902C5 EFC20 B3066 E462B 6D02F 46DEE 94DF5 545BA BB12E 63388 183D7 129F6 EE229 C6EDD C6784 B8CC1 6315E 0BF9B 57D57 2EE63 5CE44 63601 48AA8 48BCC 8BFE4 F4C50 1C030 75E36 67AF3 3FD39 540AC 94DF6 F4CEA 7337C A7B60 2C057 9E849