Cryptanalysis of the Dragonfly Key Exchange Protocol

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

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

Password authenticated key exchange protocols are used in a variety of situations to allow two parties to bootstrap a secure symmetric key (known as the session key) from a shared low-entropy secret over insecure public channels. This serves two purposes; firstly, the two parties are mutually authenticated based on whether they have the same passwords, and secondly, if the passwords are the same, a strong session key will be derived and then used to protect the subsequent communication between the two parties from eavesdropping and alteration.

It follows that password authenticated key exchange protocols must be resistant to two main threats; an adversary computing the session key that two honest parties agree to, and an adversary computing the low entropy secret that two honest parties share, either by eavesdropping on communications or injecting special values into the protocol execution. The realization of the second threat will then allow an adversary to impersonate either party to the other. This not only compromises authentication; it also compromises the privacy and integrity of subsequent communication, as the adversary can launch a man-in-the-middle attack.

*This work was supported by EPSRC First Grant EP/J011541/1

Dragonfly is a password authenticated key exchange protocol specified by Dan Harkins for exchanging session keys with mutual authentication within mesh networks [13]. 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" [11,13]. However, as acknowledged by the author [13], 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 the best of our 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 vulnerable to an attack that allows an adversary to compute the shared low-entropy secret by injecting special values into the protocol execution and then performing off-line computations. We have implemented this attack and shown it to be a practical threat.

In this paper, we base our analysis upon the original protocol specification as defined in a peer-reviewed paper [13]. However, the attack we present is trivially applicable to the variant specified in [11]. (The IETF draft has since been changed to add a public key validation step to prevent the attack after we notified Harkins of our discovery [12].)

The rest of the paper is organized as follows. In Section 2, we explain background about password authenticated key exchange protocols and focus on explaining the SPEKE protocol since it is similar to Dragonfly. Then, we present our attack, and experimentally evaluate its performance against implementations using finite fields and elliptic curve groups (Section 3). Next we discuss methods for preventing this attack, and the effects this may have on the protocol's performance (Section 4). Finally, Section 5 concludes the paper.

2 Background

We begin by introducing the concept of key exchange protocols (Section 2.1). Next we discuss password authenticated key exchange protocols (Section 2.2). Finally, we present the Dragonfly protocol (Section 2.3).

2.1 Key Exchange Protocols

Key exchange protocols were first introduced by Diffie and Hellman [8], as a method by which two parties could agree on a secret key while communicating over insecure channels. A secret key could be generated that was indistinguishable from a random key to an eavesdropper, without requiring any previous

[1] `https://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/history/`

[2] `http://comments.gmane.org/gmane.ietf.irtf.cfrg/1786`

out-of-band communication between the two parties. The initial Diffie-Hellman key exchange protocol was however vulnerable to man-in-the-middle attacks; an attacker could perform a key exchange with each party and then forward their messages as though they were communicating directly with each other.

The danger of man-in-the-middle attacks led to the need for authentication of the parties taking part in the key exchange. This authentication could be achieved either by using public key certificates [17, 19], or by using human-memorable passwords [3, 14].

2.2 PAKE Protocols

Password Authenticated Key Exchange (PAKE) schemes provide mutual authentication to the key exchange parties based on their shared password, which is generally expected to have low entropy [3]. Since the entropy is low, it would be possible to launch dictionary attacks in which every possible password is tried until the correct one is found.

There are two types of dictionary attacks: on-line and off-line attacks. In an on-line dictionary attack, the attacker directly engages with the victim and tries random guesses of password for each run of the protocol. By nature, no PAKE protocol can prevent this kind of attack. However, a secure PAKE protocol should limit such an on-line attacker to try only one password in each run of the protocol. Consecutively failed attempts can be easily detected and any further attempts can be stopped accordingly.

Off-line dictionary attacks are a far more serious threat. In an off-line dictionary attack, the attacker obtains one or more of the messages sent during an execution of the protocol, and then uses these messages to eliminate values from a dictionary of possible passwords. The messages can either be obtained by eavesdropping on an execution of the protocol between two honest parties (in which case the attacker is described as a passive attacker) or from the execution of the protocol between the attacker and an honest party (in which case the attacker is described as an active attacker and may inject special values into the exchange to aid in the attack). A secure PAKE protocol must not leak any information that would enable an attacker to eliminate likely passwords by performing off-line exhaustive comparison against a dictionary.

There are additional security requirements for a secure PAKE protocol: for example, session key indistinguishability and forward secrecy. Since they are less relevant to the particular attack presented in this paper, we refer interested readers to [10,21] for more details. The requirements for secure PAKE protocols have also been formalized in models using the paradigms of game-based security [1, 2], simulation-based security [5, 22] and universal composability [6, 7].

A number of password-based key exchange protocols have been suggested such as EKE [3,4], SRP [23], SPEKE [14], the Katz-Vaikuntanathan scheme [16] and J-PAKE [10]. Among these protocols, we will mainly focus on SPEKE because it bears a similarity to Dragonfly. The SPEKE protocol is essentially a Diffie-Hellman key exchange where the generators are replaced by values derived from the shared password. In the description of a fully constrained SPEKE [14],

Figure 1: The SPEKE Protocol

	Alice		Bob
	p is a safe prime: $p = 2 \cdot q + 1$		
1.	$x \in [1, q-1]$		$y \in Q$
2.	$X = (s^2)^x$	$\xrightarrow{X}$	
3.		$\xleftarrow{Y}$	$Y = (s^2)^y$
4.	$K = Y^x = s^{2xy}$		$K = X^y = s^{2xy}$

the protocol defines a safe prime $p = 2 \cdot q + 1$ where q is also a prime. Alice sends to Bob $(s^2)^x$ where s is a shared password and x is a random value from $[1, q-1]$. Similarly, Bob sends to Alice $(s^2)^y$ where y is a random value from $[1, q-1]$. Both Alice and Bob can then compute a common key $K = s^{2xy}$. The SPEKE protocol is summarized in Figure 1.

Zhang demonstrates that the SPEKE protocol is not fully resistant to online dictionary attacks [24]. An adversary may test more than one password in one run based on dividing passwords into groups of exponential equivalence. To mitigate this attack, Jablon revised the SPEKE specification in the IEEE P1363.2 standard draft[3] by taking the squaring operation on the one-way hash of the password rather than the password itself. The squaring operation is to ensure the protocol works in a subgroup of prime order q.

Another limitation with SPEKE is related to the use of a safe prime. Given a safe prime p of 1024 bits, the operating subgroup has a size of 1023 bits. So the exponentiation takes a long exponent of about 1023 bits. In terms of computation, this is relatively expensive since the cost of exponentiation is roughly linear to the bit length of the exponent.

We note that EKE and SRP also have issues that may encourage the choice of Dragonfly over them. The EKE protocol has been shown, over many common groups, to allow an attacker to eliminate some passwords due to the structure of observed messages [15]. SRP is not implementable over elliptic curve groups [25]. Variants of SRP over elliptic curve groups have been proposed, but have not been analyzed to the extent that SRP has [25].

2.3 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.

[3] `http://grouper.ieee.org/groups/1363/passwdPK/`

Figure 2: The Dragonfly Protocol

	Alice		Bob
	$P \in Q$		$P \in Q$
1.	$r_A, m_A \in \{1, \ldots, q\}$		$r_B, m_B \in \{1, \ldots, q\}$
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.	$ss = (P^{s_B} E_B)^{r_A}$	$\xrightarrow{A = H(ss\|E_A\|s_A\|E_B\|s_B)}$	Verify A
	$= P^{r_B r_A}$		$ss = (P^{s_A} E_A)^{r_B}$
6.	Verify B	$\xleftarrow{B = H(ss\|E_B\|s_B\|E_A\|s_A)}$	$= P^{r_A r_B}$
7.	Compute the shared key: $K = H(ss\|E_A \cdot E_B\|(s_A + s_B)\, mod\, q)$		

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 $A \cdot B$ 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 (see Figure 2 and also [13]):

- Alice and Bob have a shared password from which each can deterministically generate a password element $P \in Q$. The algorithms to map an arbitrary password to an element in Q are specified in [13] and [11]. 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 $K = H(ss|E_A \cdot E_B|(s_A + s_B)\, mod\, q)$

Similar to SPEKE, Dragonfly uses a generator that is derived from a password. However, by design Dragonfly permits the use of short exponents. So for a given modulus p of 1024 bits, the operating subgroup is only 160 bits. Hence, the length of the exponent is relatively short: only 160 bits. Overall, Dragonfly is a lot more efficient than SPEKE in terms of computation. One factor that contributes to the superior efficiency of Dragonfly over SPEKE is the omission of public key validation. At a first glance, this omission may appear harmless. However, in the following sections, we will explain that such omission renders the Dragonfly protocol vulnerable to off-line dictionary attacks, in which an attacker is able to eliminate all passwords except the correct one in one run of the Dragonfly protocol.

3 A Small Subgroup Attack on Dragonfly

We begin by presenting the algorithm we use to attack Dragonfly in Section 3.1, along with an explanation of why this attack is feasible. Next we describe experiments that measure the success and efficiency of this attack against Dragonfly using finite field cryptography (Section 3.2) and elliptic curve cryptography (Section 3.3). Finally, we present the results of these experiments in Section 3.4.

3.1 Attack Methodology

It is claimed in [13] that the Dragonfly protocol is resistant to off-line dictionary attacks. However, no security proofs are given. Instead, the author provides a heuristic security analysis of the protocol's resistance to passive and active attackers. Our analysis of the protocol has not identified any weaknesses against passive attackers.

The analysis for active attacks is 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 off-line 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 3. First, the attacker sets $E_B = S_n$ where S_n is the generator of a small subgroup of Z_p^* of order n. This small subgroup generator will have been calculated before the protocol begins. 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 to obtain the victim's password element P. This algorithm requires no further interaction with Alice, so can be completed even if Alice terminates the protocol before it is completed.

If Algorithm 1 can be completed quickly enough, then the attacker can 1) forge a valid response B to bypass authentication (so the victim is unaware

Figure 3: Small Subgroup Attack

	Alice		Attacker
	$P \in Q$		S_n
1.	$r_A, m_A \in \{1, \dots, q\}$		$s_B \in \{1, \dots, q\}$
2.	$s_A = r_A + m_A$		Set $E_B = S_n$
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 $K = H(ss \vert E_A \cdot E_B \vert (s_A + s_B)\, mod\, q)$		

that the password has been compromised); 2) compromise the secrecy of communication by deriving the session key K and using it to impersonate Bob to Alice.

Alternatively, if Algorithm 1 takes long enough to complete that Alice terminates the protocol, then the attacker is simply left with the password element P. This can then be used to falsely authenticate the attacker as Bob (or Alice as they share a password) in a future run of the protocol.

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.

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 then able to complete the protocol and/or future runs of it and impersonate either Alice or Bob to the other.

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**
    $ss' := (P'^{s_A} E_A)^{s_B} \cdot R_x$
    $A' := H(ss'|E_A|s_A|E_B|s_B)$
    if $A' = A$ **then**
      $P = P'$
      $B = H(ss'|E_B|s_B|E_A|s_A)$
      $K = H(ss'|E_A \cdot E_B|(s_A + s_B) \, mod \, q)$
      Return $\{P, B, K\}$
    end if
  end for
end for

3.2 Attack Implementation over Finite Field

We implemented an attack simulation in Java. This simulation consists of three components: the password chooser that randomly chooses a dictionary of password elements, the honest party that randomly chooses one of these elements as a password and performs the Dragonfly protocol in an honest manner, and the dishonest party that performs the dictionary attack against the honest party.

We first ran the Dragonfly protocol in a 160-bit subgroup of a 1024-bit finite field. The group parameters are specified in Appendix A. They are originally from the standard NIST cryptographic toolkit[4]. 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 A) and performed a set of experiments to determine the time to complete an off-line 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 and the average value was taken.

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

3.3 Attack Implementation over Elliptic Curve

Besides the finite field, the Dragonfly protocol can also be initiated over an Elliptic Curve, on which the Elliptic Curve Discrete Logarithm (ECDL) problem is assumed to be intractable [13]. Similar to the case of finite field, Harkins does not mandate any public key validation in the protocol specification over elliptic curve. Hence, the attack methodology explained in Section 3.1 equally applies to the elliptic curve version of the Dragonfly protocol.

To demonstrate the attack, we chose to implement the Dragonfly Protocol over a 163-bit Koblitz Curve. The group parameters are specified in Appendix A and are originally from the NIST's Recommended Elliptic Curves for Federal Government Use[5]. This curve has a co-factor of 2, so we identified a small subgroup of size 2 to use in our attack. As with the previous experiments, we mounted the attack with dictionaries of 1000, 10000 and 100000 random password elements, and performed each experiment 30 times.

3.4 Results

We note that only one possible password was identified in every experiment and this was the password chosen by the honest party. In all cases the experiments were run under Windows 7 on a 2.9GHz PC with 4GB of memory.

We measured the times taken to check all possible passwords as dictionary size varies from 1000 to 100000. This showed 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 mean times taken to check one dictionary element as the subgroup size varies are shown in Table 1.

With the original specification of the Dragonfly protocol in [13], the best strategy for the attacker is to choose the smallest subgroup: say the size 2 for the finite field. Assuming a dictionary size of $n = 10000$, the OfflineSearch algorithm will take on average $0.005 \times n/2 = 25$ seconds to find the correct password. After obtaining the correct password, the attacker can trivially derive the session key and continue to engage with the victim for the subsequent secure communication. As a result, the attack can be undetectable (though the victim may notice some delay in getting the response from the other party). We should stress that whether the attack is quick enough to be undetectable is less important. Since it is an off-line dictionary attack, in any case, the attacker will be able to obtain the victim's password. In that regard, the security has already been breached. Furthermore, the attacker is likely to significantly shorten the time of exhaustive search by distributing the calculation over several high performance machines.

[5] `http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf`

Table 1: Attack Efficiency Experiments

Type	Subgroup Size	Mean Time to Try One Password (ms)
Finite Field	2	5
Finite Field	3	6
Finite Field	13	6
Finite Field	23	8
Finite Field	463907	16935
Elliptic Curve	2	20

4 Discussion

We begin by discussing how small subgroup attacks against the Dragonfly protocol can be prevented (Section 4.1). Then, we compare the efficiency of the Dragonfly protocol to the SPEKE protocol (Section 4.2).

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. In the case of a finite field, this can be achieved by checking that E is a member of the supergroup, that E is not the identity element and that E^q is equal to the identity element [18]. For the elliptic curve, a similar check is needed to ensure the received element is a valid public key over the elliptic curve. The importance of this check – known as the public key validation – in key exchange protocols has been highlighted by Menezes and Ustagolu [20] in 2006 and also by a recent attack reported in 2012 [9].

However, to validate a public key will involve some computational cost. This is especially the case when the initiation of the protocol is over a finite field, since a full exponentiation will be needed. This can decrease the protocol efficiency and make it less appealing than its competitors. Based on the time measurements in Table 1, one might be tempted to only perform a partial validation: ensuring the element does not fall within the subgroups of sizes $\{2, 3, 13, 23\}$, since for larger sizes, the time required to complete the off-line search is much longer (say many days). However, we advise against any such partial validation, because the attacker may have much more powerful computing resource than what we had in the experiment. To adequately prevent the small subgroup attack, a full public key validation should be performed.

4.2 Comparison between Dragonfly and SPEKE

We observe that the Dragonfly protocol is very similar to SPEKE [14] with two minor changes. First, it drops the constraint in [14] that p must be a safe prime

Table 2: Consequence of attack if public key validation is missing

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

(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.) Second, instead of sending just one single element by each participant as in SPEKE, Dragonfly adds an extra scalar in the flow. However, the rationale for these changes is not explained in [13] or [11].

First of all, we note that the second change causes the Dragonfly protocol to suffer a more serious consequence than SPEKE if the public key validation is missing. To explain this, let us assume there is no public key validation in both Dragonfly and SPEKE. Without the public key validation in SPEKE, an active attacker can confine the session key to an element in a small subgroup [14]. 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 2).

Furthermore, we note that although the public key validation involves some computational cost, it does not necessarily mean it will make the protocol inefficient. For example, we can patch the Dragonfly protocol by adding a public key validation step. The patched protocol can still be more efficient than SPEKE. This is largely due to the choice of using the short exponents. We illustrate the differences in efficiency between the protocols in two ways, by considering the number of operations required to compute the exponentiations, and by measuring the time taken to perform the protocol in an experiment. We present both the number of modular multiplications and modular squaring operations needed to perform the exponentiation for each protocol and the results of an experiment to measure the times taken to execute each protocol in Table 3, for a modulus p of 1024 bits for both protocols, a subgroup size of 160 bits for Dragonfly and a subgroup size of 1023 bits for SPEKE. The experiments were run under Windows 7 on a 2.9GHz PC with 4GB of memory. Each experiment was performed 30 times. As shown in Table 3, although the public key validation step adds one extra modular exponentiation to the existing three modular exponentiation, the patched Dragonfly still outperforms SPEKE.

Table 3: Computational cost for each participant given a modulus p of 1024 bits, 1023 bit subgroup for SPEKE and 160 bit subgroup for Dragonfly

Protocol	No. of Mod Mul	No. of Mod Square	Time (ms)
SPEKE	1023	2046	250.6
Dragonfly (original)	240	480	49.4
Dragonfly (patched)	320	640	64.0

5 Conclusion

We have shown that the original Dragonfly protocol is vulnerable to a small subgroup based off-line dictionary attack. The protocol can be patched by adding a public key validation step in the specification. In the past three decades, public key validation has been frequently omitted in key exchange protocols (even in standard specifications). Sometimes that omission was due to a negligent mistake, but more often, that was intentional: because the public key validation is commonly seen as an expensive operation. Through the example of the Dragonfly protocol, we show the importance of the public key validation and also demonstrate that the impact of adding public key validation on the overall protocol efficiency is not as significant as some people may think.

Acknowledgment

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

References

[1] Michel Abdalla, Pierre-Alain Fouque, and David Pointcheval. Password-based authenticated key exchange in the three-party setting. In *8th International Workshop on Practice and Theory in Public-Key Cryptography (PKC)*, pages 65–84, 2005.

[2] Mihir Bellare, David Pointcheval, and Phillip Rogaway. Authenticated key exchange secure against dictionary attacks. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 139–155, 2000.

[3] Steven M. Bellovin and Michael Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.

[4] Steven M. Bellovin and Michael Merritt. Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and

password file compromise. In *1st ACM Conference on Computer and Communications Security*, pages 244–250, 1993.

[5] Victor Boyko, Philip D. MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 156–171, 2000.

[6] Ran Canetti, Shai Halevi, Jonathan Katz, Yehuda Lindell, and Philip D. MacKenzie. Universally composable password-based key exchange. In *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*, 2005.

[7] Ran Canetti and Hugo Krawczyk. Universally composable notions of key exchange and secure channels. In *International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, pages 337–351, 2002.

[8] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, pages 644–654, 1976.

[9] Feng Hao and Dylan Clarke. Security analysis of a multi-factor authenticated key exchange protocol. In *International Confernce on Applied Cryptography and Network Security (ACNS)*, pages 1–11, 2012.

[10] Feng Hao and Peter Ryan. J-PAKE: Authenticated key exchange without PKI. *Transactions on Computational Science*, pages 192–206, 2010.

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

[12] Dan Harkins. Dragonfly key exchange - internet research task force internet draft version 01. `http://tools.ietf.org/html/draft-irtf-cfrg-dragonfly-01` accessed Apr 2013.

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

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

[15] Barry Jaspan. Dual-workfactor encrypted key exchange: Efficiently preventing password chaining and dictionary attacks. In *Sixth USENIX Security Symposium*, 1996.

[16] Jonathan Katz and Vinod Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In *15th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 636–652, 2009.

[17] Laurie Law, Alfred Menezes, Minghua Qu, Jerry Solinas, and Scott Vanstone. An efficient protocol for authenticated key agreement. *Designs, Codes and Cryptography*, pages 119–134, 2003.

[18] Arjen K. Lenstra and Eric R. Verheul. Fast irreducibility and subgroup membership testing in XTR. In *4th International Workshop on Practice and Theory in Public Key Cryptography (PKC)*, pages 73–86, 2001.

[19] A. Menezes, M. Qu, and S. Vanstone. Some new key agreement protocols providing mutual implicit authentication. In *Selected Areas in Cryptography*, 1995.

[20] Alfred Menezes and Berkant Ustaoglu. On the importance of public-key validation in the MQV and HMQV key agreement protocols. In *7th International Conference on Cryptology in India (INDOCRYPT)*, pages 133–147, 2006.

[21] David Pointcheval. Password-based authenticated key exchange. In *15th IACR International Conference on Practice and Theory of Public-Key Cryptography (PKC)*, pages 390–397, 2012.

[22] Victor Shoup. On formal models for secure key exchange. http://www.shoup.net/papers/skey.pdf, 1999.

[23] Thomas Wu. The secure remote password protocol. In *Internet Society Network and Distributed System Security Symposium*, pages 97–111, 1998.

[24] Muxiang Zhang. Analysis of the speke password-authenticated key exchange protocol. *Communications Letters, IEEE*, pages 63–65, 2004.

[25] Zhu Zhao, Zhongqi Dong, and Yongge Wang. Security analysis of a password-based authentication protocol proposed to IEEE 1363. *Theoretical Computer Science*, pages 280–287, 2006.

A Group Parameters

A.1 Finite Field Subgroup

The group parameters are taken from the NIST cryptographic toolkit using a 1024 bit modulus, and are shown in Table 4. The subgroups of different sizes and their respective generators are shown in Table 5. However, only subgroups of sizes of up to 32 bits are listed.

Table 4: Group Parameters (Finite Field Subgroup)

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

A.2 Elliptic Curve

The group parameters are for a 163 bit Koblitz Curve, taken from the NIST's Recommended Elliptic Curves for Federal Government Use, and are shown in Table 6. The chosen curve has a small subgroup of size 2 with the point (0,1) being the generator.

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

Table 5: Subgroup Generators (Finite Field)

Table 6: Group Parameters (Elliptic Curve)

Parameter	Value
Binary Field Polynomial	$t^{163} + t^7 + t^6 + t^3 + 1$
Curve Equation	$y^2 + xy = x^3 + x^2 + 1$
Order (base 16)	4 00000000 00000000 00020108 A2E0CC0D 99F8A5EF
Generator x co-ordinate (base 16)	2 FE13C053 7BBC11AC AA07D793 DE4E6D5E 5C94EEE8
Generator y co-ordinate (base 16)	2 89070FB0 5D38FF58 321F2E80 0536D538 CCDAA3D9