Password Authenticated Key Exchange by Juggling

Feng Hao*[1] and Peter Ryan[2]

[1] Center for Computational Science, University College London

[2] School Of Computing Science, University of Newcastle upon Tyne

Abstract. Password-Authenticated Key Exchange (PAKE) studies how to establish secure communication between two remote parties solely based on their shared password, without requiring a Public Key Infrastructure (PKI). Despite extensive research in the past decade, this problem remains unsolved. Patent has been one of the biggest brakes in deploying PAKE solutions in practice. Besides, even for the patented schemes like EKE and SPEKE, their security is only heuristic; researchers have reported some subtle but worrying security issues.

In this paper, we propose to tackle this problem using an approach different from all past solutions. Our protocol, Password Authenticated Key Exchange by Juggling (J-PAKE), achieves mutual authentication in two steps: first, two parties send ephemeral public keys to each other; second, they encrypt the shared password by juggling the public keys in a verifiable way. The first use of such a juggling technique was seen in solving the Dining Cryptographers problem in 2006. Here, we apply it to solve the PAKE problem, and show that the protocol is zero-knowledge as it reveals nothing except one-bit information: whether the supplied passwords at two sides are the same. With clear advantages in security, our scheme has comparable efficiency to the EKE and SPEKE protocols.

Keywords: Password-Authenticated Key Exchange, EKE, SPEKE, key agreement.

1 Introduction

The username/password paradigm is the most commonly used authentication mechanism in security applications [3]. Alternative authentication factors, including tokens and biometrics, require purchasing additional hardware, which is often considered too expensive for an application.

However, passwords are low-entropy secrets, and subject to dictionary attacks [3]. Hence, they must be protected during transmission. The widely deployed method is to send passwords through SSL/TLS [36]. But, this requires a Public Key Infrastructure (PKI) in place; maintaining a PKI is expensive. In addition, using SSL/TLS is subject to man-in-the-middle attacks [3]. If a user authenticates himself to a phishing website by disclosing his password, the password will be stolen even though the session is fully encrypted.

Since passwords are inherently weak, one logic solution seems to replace them with strong secrets, say, cryptographically secure private keys. This approach was adopted by the UK National Grid Service (NGS) to authenticate

* This work was done while the authors worked on an EPSRC-funded project (EP/D051754/1).

users [4]. In the UK, anyone who applies to access the national grid computing resource must first generate a private/public key pair of his own, and then have the public key certified by NGS. The authentication procedure for the grid computing environments in the USA is similar [18]. However, developments in the past ten years reveal that users – most of them are non-computer specialists – encounter serious difficulties in managing their private keys and certificates [5]. This has greatly hindered the wider acceptance of the grid computing technology.

Hence, weak passwords are just a fact of life that we must face. Researchers have been actively exploring ways to perform password-based authentication without using PKIs or certificates – a research subject called the Password-Authenticated Key Exchange (PAKE) [9]. The first milestone came in 1992 when Bellovin and Merrit introduced the EKE protocol [13]. Despite some reported weaknesses [22, 26, 29, 32], the EKE protocol first demonstrated that the PAKE problem was at least solvable. Since then, a number of protocols have been proposed. Many of them are simply variants of EKE, instantiating the "symmetric cipher" in various ways [9]. The few techniques that claim to resist known attacks have almost all been patented – most notably, EKE was patented by Lucent Technologies [15], and SPEKE by Phoenix Technologies [24]. As a result, the scientific community and the wider security industry cannot readily benefit from the implementations of these techniques [16].

The security with the EKE and SPEKE protocols is only heuristic. Given the way the two techniques were designed, formal security proofs seem unlikely without introducing new assumptions or relaxing requirements; we will explain the details in Section 4. In the following section, we will introduce a different approach to solve the PAKE problem, and show that our solution is free from the security issues reported with the EKE and SPEKE protocols.

2 Protocol

2.1 Model

We assume the key exchange is carried out over an unsecured network. In such a network, there is no secrecy in communication, so transmitting a message is essentially no different from broadcasting it to all. Worse, the broadcast is unauthenticated. An attacker can intercept a message, change it at will, and then relay the modified message to the intended recipient.

It is perhaps surprising that we are still able to establish a private and authenticated channel in such a hostile environment solely based on a shared password – in other words, bootstrapping a *high-entropy* cryptographic key from a *low-entropy* secret. First of all, we formulate the security requirements that a PAKE protocol should fulfill.

1. **Off-line dictionary attack resistance** – It does not leak any password verification information to a passive attacker.
2. **Forward secrecy** – It produces session keys that remain secure even when the password is later disclosed.
3. **Known-key security** – It prevents a disclosed session key from affecting the security of other sessions.
4. **On-line dictionary attack resistance** – It limits an active attacker to test only one password per protocol execution.

In our threat model, we do not consider the Denial of Service (DoS) attack [3], which is rare but powerful. Since almost all PAKE protocols are built upon public key cryptography, they are naturally vulnerable to DoS attacks, in which an attacker's sole purpose is to keep a server performing expensive public key operations. When this becomes a threat in some applications, there are well-established solutions – for example, we can add a client-puzzle protocol [21] before engaging in any key exchange (also see [10]).

There are generally two types of PAKE protocols: balanced and augmented ones [34]. A balance scheme assumes two communicating parties hold symmetric secret information, which could be a password, or a hashed password. It is generic for any two-party communication, including client-client and client-server. On the other hand, an augmented scheme is more customized to the client-server case. It adds the "server compromise resistance" requirement – an attacker should not be able to impersonate users to a server after he has stolen the password verification files stored on that server, but has not performed dictionary attacks to recover the passwords [14, 23]. This is usually realized by storing extra password verification data – such as a (weak) public key derived from the password – together with a hash of the password on the server [14].

By design an augmented scheme is more complex and requires more computing power, but the added benefits are questionable. First, one may ask whether the threat of impersonating users to a *compromised* server is significantly realistic in practice [30]. More importantly, none of the augmented schemes can provide any comforting assurance once the server is indeed compromised – all passwords need to be revoked and updated anyway. Though it may be the case that some passwords are stronger than others and are more resistant to off-line dictionary attacks (as said in [23]), the "strongness" of a password is difficult to quantify, given that people often vary the same password slightly for different applications and use date of birth or pet's name as the password. If the presumption of a "strong" password turns out incorrect, the so-called "server compromise resistance" might just provide a false sense of security. To defend against the "server compromise" threat, a more proper solution is to apply a threshold scheme to distribute the password verification data among a set of servers, as suggested in [17, 31].

In this paper, we only consider the balanced case (there is a general method to convert any balanced scheme to an augmented one if needed [19]). In the following section, we will propose a balanced PAKE protocol that satisfies the outlined requirements.

2.2 Two-step exchange

Let G denote a subgroup of $\mathbb{Z}_p^*$ with prime order q in which the Decision Diffie-Hellman problem (DDH) is intractable [8]. Let g be a generator in G. The two communicating parties, Alice and Bob, both agree on (G, g). Let s be their shared password[3], and $s \neq 0$ for any non-empty password. Since s has low entropy, we assume the value of s falls within $[1, q-1]$.

Alice selects two secret values x_1 and x_2 at random: $x_1 \in_R \mathbb{Z}_q$ and $x_2 \in_R \mathbb{Z}_q^*$. Similarly, Bob selects $x_3 \in_R \mathbb{Z}_q$ and $x_4 \in_R \mathbb{Z}_q^*$. Note that since q is prime, $\mathbb{Z}_q$ only differs $\mathbb{Z}_q^*$ in that the later excludes '0' [36]. Hence, $x_2, x_4 \neq 0$; the reason will be evident in security analysis.

[3] Depending on the application, s could also be a hash of the shared password, so that it preserves some privacy against an honest-but-curious server administrator.

Step 1 *Alice sends out* g^{x_1}, g^{x_2} *and knowledge proofs for* x_1 *and* x_2. *Similarly, Bob sends out* g^{x_3}, g^{x_4} *and knowledge proofs for* x_3 *and* x_4.

The above communication can be completed in one step as neither party depends on the other. When this step finishes, Alice and Bob verify the received knowledge proofs, and also check $g^{x_2}, g^{x_4} \neq 1$.

Step 2 *Alice sends out* $\mathcal{A} = g^{(x_1+x_3+x_4)\cdot x_2 \cdot s}$ *and a knowledge proof for* $x_2 \cdot s$. *Similarly, Bob sends out* $\mathcal{B} = g^{(x_1+x_2+x_3)\cdot x_4 \cdot s}$ *and a knowledge proof for* $x_4 \cdot s$.

When this step finishes, Alice computes $K = (\mathcal{B}/g^{x_2 \cdot x_4 \cdot s})^{x_2} = g^{(x_1+x_3)\cdot x_2 \cdot x_4 \cdot s}$, and Bob computes $K = (\mathcal{A}/g^{x_2 \cdot x_4 \cdot s})^{x_4} = g^{(x_1+x_3)\cdot x_2 \cdot x_4 \cdot s}$. With the same keying material K, a session key can be derived $\kappa = H(K)$, where H is a hash function. Before using the session key, Alice and Bob may perform an additional key confirmation process as follows: Alice sends $H(H(\kappa))$ to Bob, and Bob then replies $H(\kappa)$. This process is the same as in [22]. It gives explicit assurance that the two ends derived the same session key[4].

In the protocol, senders need to produce valid knowledge proofs. The necessity of the knowledge proofs is motivated by Anderson's sixth principle in designing secure protocols [2]: "*Do not assume that a message you receive has a particular form unless you can check this.*" Fortunately, Zero-Knowledge Proof (ZKP) is a well-established primitive in cryptography; it can allow one to prove his knowledge of a discrete logarithm without revealing it [36].

As one example, we could use Schnorr's signature [37], which is non-interactive, and reveals nothing except the one bit information: "whether the signer knows the discrete logarithm". Let H be a secure hash function[5]. To prove the knowledge of the exponent for g^x, one sends $\{g^v, r = v - x_i h\}$ where $v \in_R \mathbb{Z}_q$ and $h = H(g, g^v, g^x, \text{SignerID})$. This signature can be verified by the receiver through checking whether g^v and $g^r g^{xh}$ are equal. Adding the SignerID into the hash function is to prevent replaying the signature. Since Alice and Bob's SignerIDs are unique, Alice cannot replay Bob's signature back to Bob and vice versa.

In the second step of the protocol, Alice sends $\mathcal{A} = g_a^{x_2 \cdot s}$ to Bob, where $g_a = g^{x_1+x_3+x_4}$. Here, g_a serves as a generator. As the group G has prime order, any non-identity element is a generator [36]. So Alice could simply check $g_a \neq 1$ to ensure it is a generator. In fact, as we will explain in Section 3, $(x_1 + x_3 + x_4)$ is random over $\mathbb{Z}_q$ even in the face of active attacks. Hence, the probability for $g_a = 1$ is extremely small – on the order of 2^{-160} for 160-bit q. Symmetrically, the same argument applies to the Bob's case.

2.3 Implementation

Since our protocol involves several zero-knowledge proofs, one might concern about its cost. We now count the number of exponentiations in the protocol and evaluate its computational efficiency. Note that for Schnorr's

[4] Actually, the explicit key confirmation process is not essential [35]. Alice and Bob can also start using the session key straight away to encrypt data, which is called an implicit key confirmation process. In terms of security, the difference between the two approaches is insignificant.

[5] Schnorr's signature is provably secure in the random oracle model, which requires a secure hash function.

Item	Description	No of Exp	Time (ms)
1	Compute $\{g^{x_1}, g^{x_2}\}$ and KPs for $\{x_1, x_2\}$	4	23
2	Verify KPs for $\{x_3, x_4\}$	4	24
3	Compute $\mathcal{A}$ and KP for $\{x_2 \cdot s\}$	2	9
4	Verify KP for $\{x_4 \cdot s\}$	2	10
5	Compute κ	2	9
	Total	**14**	**75**

Table 1. Computational cost for Alice in J-PAKE

signature, it requires one exponentiation in generation and two in verification. Hence, in our protocol, each party would need to perform 14 exponentiations in total – including 8 in the first step, 4 in the second step, and 2 in computing the session key.

To better assess the cost in real terms, we implement the protocol in Java on a 2.33-GHz laptop running Mac OS X. The modulus p is chosen 1024-bit and the subgroup order q 160-bit. The cost for Alice is summarized in Table 1; for Bob, it is the same. The results demonstrate that the protocol – executed only once in a session – runs sufficiently fast. The total computation time is merely 0.075 sec. As compared to the time that the user keys in his password, this latency is negligible at the client. However, the cost at the server may accumulate to be significant if requests are dealt with simultaneously. Therefore, the threat of Denial of Service (DoS) attacks still needs to be properly addressed in practical deployments (e.g., by using [21]).

3 Security analysis

In this section, we analyze the protocol's resistance against both passive and active attacks. First, we consider a passive attacker who eavesdrops on the communication between Alice and Bob. Alice's ciphertext $\mathcal{A}$ contains the term $(x_1 + x_3 + x_4)$ on the exponent. The following two lemmas show the security properties of $(x_1 + x_3 + x_4)$ and $\mathcal{A}$.

Lemma 1 *Under the Discrete Logarithm (DL) assumption, Bob cannot compute* $(x_1 + x_3 + x_4)$.

Proof. To obtain a contradiction, we reveal x_2 to Bob. The knowledge proofs in the protocol show that Bob knows x_3 and x_4. Hence, Bob knows $\{g^{x_1}, x_2, x_3, x_4\}$ (based on which he can compute $\mathcal{A}$, $\mathcal{B}$). Assume Bob is able to compute $(x_1+x_3+x_4)$. Then, he is able to compute x_1. This, however, contradicts the DL assumption [36], which states that one cannot compute x_1 from g, g^{x_1}. Therefore, even with x_2 revealed, Bob is still unable to compute $(x_1 + x_3 + x_4)$. □

Lemma 2 *Under the Decision Diffie-Hellman (DDH) assumption, Bob cannot distinguish Alice's ciphertext* $\mathcal{A} = g^{(x_1+x_3+x_4)\cdot x_2 \cdot s}$ *from a random element in the group.*

Proof. From Lemma 1, $(x_1 + x_3 + x_4)$ is a random value over $\mathbb{Z}_q$, unknown to Bob. Also, $x_2 \cdot s$ is a random value over $\mathbb{Z}_q$, unknown to Bob. Based on the Decision Diffie-Hellman assumption [8], Bob cannot distinguish $\mathcal{A}$ from a random element in the group. □

Based on the protocol symmetry, the above two Lemmas can be easily adapted from Alice's perspective – Alice cannot compute $(x_1 + x_2 + x_3)$, nor distinguish $\mathcal{B}$ from a random element in the group. The following theorem proves that our protocol fulfills the "off-line dictionary attack resistance" requirement (see Section 2.1).

Theorem 3 (Off-line dictionary attack resistance) *Under the DDH assumption, the ciphertexts* $\mathcal{A} = g^{(x_1+x_3+x_4)\cdot x_2 \cdot s}$ *and* $\mathcal{B} = g^{(x_1+x_2+x_3)\cdot x_4 \cdot s}$ *do not leak any information for password verification.*

Proof. Lemma 2 implies that Bob cannot computationally correlate $\mathcal{A}$ to the ciphertext he can compute: $\mathcal{B}$. From Lemma 2, even Bob cannot distinguish $\mathcal{A}$ from a random value in G. Surely, a passive attacker cannot distinguish $\mathcal{A}$ from a random value in G, nor can he computationally correlate $\mathcal{A}$ to $\mathcal{B}$. Based on the protocol symmetry, the passive attacker cannot distinguish $\mathcal{B}$ from a random value in G either. Therefore, to a passive attacker, $\mathcal{A}$ and $\mathcal{B}$ are two random and independent values; they do not leak any useful information for password verification. □

While the above theorem shows that the password is secure against a passive attacker, we now show the session key is secure too. In the following theorem, we consider a stronger attacker, who knows the disclosed s.

Theorem 4 (Forward secrecy) *Under the Square Computational Diffie-Hellman (SCDH) assumption*[6]*, the past session keys derived from the protocol remain secure even when the secret* s *is later disclosed.*

Proof. After knowing s, the passive attacker wants to compute $\kappa = H(K)$ given inputs: $\{g^{x_1}, g^{x_2}, g^{x_3}, g^{x_4}, g^{(x_1+x_3+x_4)\cdot x_2}, g^{(x_1+x_2+x_3)\cdot x_4}\}$.

Assume the attacker is able to compute $K = g^{(x_1+x_3)\cdot x_2 \cdot x_4}$ from those inputs. For simplicity, let $x_5 = x_1 + x_3 \bmod q$. The attacker behaves like an oracle – given the ordered inputs $\{g^{x_2}, g^{x_4}, g^{x_5}, g^{(x_5+x_4)\cdot x_2}, g^{(x_5+x_2)\cdot x_4}\}$, it returns $g^{x_5 \cdot x_2 \cdot x_4}$. This oracle can be used to solve the SCDH problem as follows. For g^x where $x \in_R \mathbb{Z}_q$, we query the oracle by supplying $\{g^{-x+a}, g^{-x+b}, g^x, g^{b\cdot(-x+a)}, g^{a\cdot(-x+b)}\}$, where a, b are arbitrary values chosen from $\mathbb{Z}_q$, and obtain $f(g^x) = g^{(-x+a)\cdot(-x+b)\cdot x} = g^{x^3-(a+b)\cdot x^2+ab\cdot x}$. In this way, we can also obtain $f(g^{x+1}) = g^{(x+1)^3-(a+b)\cdot(x+1)^2+ab\cdot(x+1)} = g^{x^3+(3-a-b)\cdot x^2+(3-2a-2b+ab)\cdot x+1-a-b+ab}$. Now we are able to compute $g^{x^2} = \left(f(g^{x+1}) \cdot f(g^x)^{-1} \cdot g^{(-3+2a+2b)\cdot x-1+a+b-ab}\right)^{1/3}$. This, however, contradicts the SCDH assumption [6], which states that one cannot compute g^{x^2} from g, g^x. □

We now consider the case when a session key is compromised (see the "known-key security" requirement in Section 2.1). Compared with other ephemeral secrets x_i $(i = 1 \ldots 4)$ – which can be immediately erased after the key bootstrap phase – a session key lasts longer throughout the session, which might increase the likelihood of its exposure. An exposed session key should not cause any global effect on the system [36].

In our protocol, a known session key does not affect the security of either the password or other session keys. From an explicit key confirmation process, an attacker learns $H(H(\kappa)) = H(H(H(K)))$ and $H(\kappa) = H(H(K))$. It is obvious that learning $\kappa = H(K)$ does not give the attacker any additional knowledge about K, and therefore, the security of the password encrypted at that session remains intact. Also, the session key $\kappa = H(g^{(x_1+x_3)\cdot x_2 \cdot x_4 \cdot s})$ is

[6] The SCDH assumption is provably equivalent to the Computational Diffie-Hellman (CDH) assumption – solving SCDH implies solving CDH, and vice versa [6]

determined by the (fresh) ephemeral inputs from both parties in the session. Note that $x_2, x_4 \neq 0$ by definition and $(x_1 + x_3)$ is random over $\mathbb{Z}_q$, hence the obtained session key is completely different from keys derived in other sessions. Therefore, compromising a session key has no effect on other session keys.

Finally, we study an active attacker, who directly engages in the protocol execution. Without loss of generality, we assume Alice is honest, and Bob is compromised (i.e., an attacker).

In the protocol, Bob demonstrates that he knows x_4 and the exponent of g_b, where $g_b = g^{x_1+x_2+x_3}$. Therefore, the format of the ciphertext sent by Bob can be described as $\mathcal{B}' = {g_b}^{x_4 \cdot s'}$, where s' is a value that Bob (the attacker) can choose freely.

Theorem 5 (On-line dictionary attack resistance) *Under the SCDH assumption, an active attacker cannot compute the session key if he chose a value $s' \neq s$.*

Proof. After receiving $\mathcal{B}'$, Alice computes

$$K' = (\mathcal{B}'/g^{x_2 \cdot x_4 \cdot s})^{x_2} \tag{1}$$

$$= g^{x_1 \cdot x_2 \cdot x_4 \cdot s'} \cdot g^{x_2 \cdot x_3 \cdot x_4 \cdot s'} \cdot g^{{x_2}^2 \cdot x_4 \cdot (s'-s)} \tag{2}$$

To obtain a contradiction, we reveal x_1 and s, and assume that the attacker is now able to compute K'. The attacker behaves as an oracle: given inputs $\{g^{x_2},\ x_1,\ x_3,\ x_4,\ s,\ s'\}$, it returns K'. Note that the oracle does not need to know x_2, and it is still able to compute $\mathcal{A} = g^{(x_1+x_3+x_4) \cdot x_2 \cdot s}$ and $\mathcal{B}' = g^{(x_1+x_2+x_3) \cdot x_4 \cdot s'}$ internally. Thus, the oracle can be used to solve the Square Computational Diffie-Hellman problem by computing ${g^{x_2}}^2 = (K'/(g^{x_1 \cdot x_2 \cdot x_4 \cdot s'} \cdot g^{x_2 \cdot x_3 \cdot x_4 \cdot s'}))^{{x_4}^{-1}(s'-s)^{-1}}$. Here[7], $x_4 \neq 0$ and $s' - s \neq 0$. This, however, contradicts the SCDH assumption. So, even with x_1 and s revealed, the attacker is still unable to compute K' (and hence cannot perform key confirmation later). □

The above theorem shows that our protocol is zero-knowledge. Because of the knowledge proofs, the attacker is left with the only freedom to choose an arbitrary s'. If $s' \neq s$, he is unable to derive the same session key as Alice. During the later key confirmation process, the attacker will learn one-bit information: whether s' and s are equal. This is the best that any PAKE protocol can possibly achieve, because by nature we cannot stop an imposter from trying a random guess of password. However, consecutively failed guesses can be easily detected, and thwarted accordingly. The security properties of our protocol are summarized in Table 2.

4 Comparison

In this section, we compare our protocol with the two well-known balanced schemes: EKE and SPEKE. These two techniques have several variants, which follow very similar constructs [9]. However, it is beyond the scope of this paper to evaluate them all. Also, we will not compare with augmented schemes (e.g., A-EKE, B-SPEKE, SRP, AMP and OPAKE [34]), as they have different design goals.

[7] This explains why in the protocol definition we need $x_4 \neq 0$, and symmetrically, $x_2 \neq 0$

Modules	Security property	Attacker type	Assumptions
Schnorr signature	leak 1-bit: whether sender knows discrete logarithm	passive/active	DL and random oracle
Password encryption	indistinguishable from random	passive/active	DDH
Session key	incomputable	passive	CDH
	incomputable	passive (know s)	CDH
	incomputable	passive (know other session keys)	CDH
	incomputable	active (if $s' \neq s$)	CDH
Key confirmation	leak nothing	passive	–
	leak 1-bit: whether $s' = s$	active	CDH

Table 2. Summary of J-PAKE security properties

The design principles underlying EKE and SPEKE are similar; both protocols can be seen as a *slight* modification of the basic Diffie-Hellman (DH) key exchange protocol. However, in the protocol design, even the slightest modification could cause profound effects. The EKE and SPEKE designs are no exception, as we explain below.

Bellovin and Merrit introduced two EKE constructs: using RSA (which was later shown insecure [29]) and DH. Here, we only describe the later, which modifies a basic DH protocol by symmetrically encrypting the exchanged items. Let α be a primitive root modulo p. In the protocol, Alice sends to Bob $[\alpha^{x_a}]_s$, where $x_a \in_R \mathbb{Z}_p^*$ and $[\ldots]_s$ denotes a symmetric cipher using the password s as the key. Similarly, Bob sends to Alice $[\alpha^{x_b}]_s$, where $x_b \in_R \mathbb{Z}_p^*$. Finally, Alice and Bob compute a common key $K = \alpha^{x_a \cdot x_b}$. More details can be found in [13].

Apparently, a straightforward implementation of the above protocol is insecure [26]. Since the password is too weak to be used as a normal encryption key, the content within the symmetric cipher must be strictly random. But, for a 1024-bit number modulo p, not every bit is random. Hence, a passive attacker can rule out candidate passwords by applying them to decipher $[\alpha^{x_a}]_s$, and then checking whether the results fall within $[p, 2^{1024} - 1]$.

There are suggested countermeasures. In [13], Bellovin and Merrit recommended to transmit $[\alpha^{x_a} + r \cdot p]_s$ instead of $[\alpha^{x_a}]_s$ in the actual implementation, where $r \cdot p$ is added using a non-modular operation. The details on defining r can be found in [13]. However, this solution was explained in an ad-hoc way, and it involves changing the protocol of its existing form. Due to lack of a complete description of the final protocol, it is difficult to assess its security. Alternatively, Jaspan suggested to address this issue by choosing p as close to a power of 2 as possible [26]. This might alleviate the issue, but does not resolve it.

One implication of the above issues is that proving the security of an EKE is difficult. To address this, Bellare, Pointcheval and Rogaway introduced an "ideal-cipher model", and proved that an EKE is secure under that model [7]. However, the "ideal cipher" was not concretely defined in [7]; it was later clarified by Boyd et al. in [9]: the assumed cipher works like a random function in encryption, but must map fixed-size strings to elements of G in decryption (also see [39]). Yet, no such ciphers are readily available in practice; indeed, several proposed instantiations of such an "ideal cipher" were easily broken [39].

Another limitation with the EKE protocol is that it does not securely accommodate short exponents. The protocol definition requires α^{x_a} and α^{x_b} be uniformly distributed over the whole group $\mathbb{Z}_p^*$ [13]. Therefore, the secret keys x_a and x_b must be randomly chosen from $[1, p-1]$, and consequently, an EKE must use 1024-bit exponents if the modulus p is chosen 1024-bit. Unlike our protocol, an EKE cannot operate in groups with distinct features,

such as a subgroup with prime order – a passive attacker would then be able to trivially uncover the password by checking the order of the decrypted item. Since the cost of exponentiation is linear with the bit-length of the exponent [36], one exponentiation in an EKE is equivalent in cost to 6-7 exponentiations in a J-PAKE (for the 1024-bit p and 160-bit q setting). Hence, though an EKE only requires two exponential operations per user, the computational cost is approximately the same as that of our protocol.

Jablon proposed a different protocol, called Simple Password Exponential Key Exchange (SPEKE), by replacing a fixed generator in the basic DH protocol with a password-derived variable [22]. In the description of a fully constrained SPEKE, the protocol defines a safe prime $p = 2q + 1$, where q is also a prime. Alice sends to Bob $(s^2)^{x_a}$ where s is the shared password and $x_a \in_R \mathbb{Z}_q^*$; similarly, Bob sends to Alice $(s^2)^{x_b}$ where $x_b \in_R \mathbb{Z}_q^*$. Finally, Alice and Bob compute $K = s^{2 \cdot x_a \cdot x_b}$. The squaring operation on s is to make the protocol work within a subgroup of prime order q.

Recently, Zhang pointed out the risk of using a password-derived variable as the base [38]. Since some passwords are exponentially equivalent, an active attacker may exploit that equivalence to test multiple passwords in one go. This problem is particularly serious if a password is a Personal Identification Numbers (PIN). One countermeasure might be to hash the password before squaring, but that does not resolve the problem. Hashed passwords are still confined to a pre-defined small range. There is no guarantee that an attacker is unable to formulate exponential relationships among hashed passwords; existing hash functions were not designed for that purpose. Hence, at least in theory, this reported weakness disapproves the original claim in [22] that a SPEKE only permits one guess of password in one attempt.

Similar to the case with an EKE, a fully constrained SPEKE uses long exponents. For a 1024-bit modulus p, the key space is within $[1, q-1]$, where q is 1023-bit. In [22], Jablon suggested to use 160-bit short exponents in a SPEKE, by choosing x_a and x_b within a dramatically smaller range $[1, 2^{160} - 1]$. But, this would give a passive attacker side information that the $1023 - 160 = 863$ most significant bits in a full-length key are all '0's. The security is not reassuring, as the author later acknowledged in [25]. Therefore, for the same reason explained earlier, the computational cost in a SPEKE is roughly the same as ours, despite that a SPEKE only requires two exponentiations per user.

To sum up, an EKE requires changing the protocol in its existing form for a secure implementation. As for a SPEKE, it has the drawback that an active attacker may test multiple passwords in one protocol execution. Furthermore, neither protocol – in the original form – accommodates short exponents securely. Finally, neither protocol is provably secure; formal security proofs seem unlikely without introducing new security assumptions [7] or relaxing security requirements [33].

We choose to solve the PAKE problem using a different approach. The novelty of our design is that we encrypt the password by juggling the public keys in a way that can be verified. As a result, our scheme is provably secure, allows flexible use of short exponents, and strictly limits an active attacker to test only one password per protocol execution. A similar use of this juggling technique was seen in solving the Dining Cryptographers problem[8] in 2006 [20]. One difference is that [20] works in a multi-party setting, while ours in a two-party one. Yet both schemes

[8] The Dining Cryptographers problem was first introduced by Chaum in 1988 [11]

use the same technique, which we call "public key juggling" – i.e., re-arranging the public keys in a particular way, and encrypting data by introducing quadratic terms on the exponents in a verifiable way. To our best knowledge, this construct is significantly different from all past PAKE protocols. In the area of PAKE research – which has been troubled by many patent arguments surrounding existing schemes [16] – a new construct may be helpful.

5 Conclusion

In this paper, we proposed a protocol, called J-PAKE, which authenticates a password with zero-knowledge and then subsequently creates a strong session key if the password is correct. We proved that the protocol fulfills the following properties: it prevents off-line dictionary attacks; provides forward secrecy; insulates a known session key from affecting any other sessions; and strictly limits an active attacker to guess only one password per protocol execution. A Java implementation of the protocol demonstrates that the total computation time for the session-key setup is merely 75 ms. As compared to the de facto internet standard SSL/TLS, J-PAKE is more lightweight in password authentication with two notable advantages: 1). It requires no PKI deployments; 2). It protects users from leaking passwords (say to a fake bank website).

Acknowledgments

We are obliged to Ross Anderson and Piotr Zieliński for their invaluable comments.

References

1. M. Abdalla and D. Pointcheval, "Simple password-based encrypted key exchange protocols," Topics in Cryptology – CTRSA'05, LNCS 3376, pp. 191–208, 2005.
2. R.J. Anderson, R. Needham, "Robustness principles for public key protocols," Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology, LNCS 963, pp. 236–247, 1995.
3. R.J. Anderson, *Security Engineering : A Guide to Building Dependable Distributed Systems*, New York, Wiley 2001.
4. The official UK National Grid Service website: `http://www.grid-support.ac.uk/`
5. B. Beckles, V. Welch, J. Basney, "Mechanisms for increasing the usability of grid security," International Journal of Human-Computer Studies, Vol. 63, No. 1-2, pp. 74-101, July 2005.
6. F. Bao, R.H. Deng, H. Zhu, "Variations of Diffie-Hellman problem," Proceeding of Information and Communication Security, LNCS 2836, pp. 301–312, 2003.
7. M. Bellare, D. Pointcheval, P. Rogaway, "Authenticated key exchange secure against dictionary attacks," Eurocrypt'00, LNCS 1807, pp. 139–155, 2000.
8. D. Boneh, "The decision Diffie-Hellman problem," Proceedings of the Third International Symposium on Algorithmic Number Theory, LNCS 1423, pp. 48–63, 1998.

9. C. Boyd, A. Mathuria, *Protocols for authentication and key establishment*, Springer-Verlag, 2003.
10. E. Bresson, O. Chevassut, D. Pointcheval, "New security results on Encrypted Key Exchange," Proceedinsg of the 7th International Workshop on Theory and Practice in Public Key Cryptography, PKC'04, LNCS 2947, pp. 145–158, 2004.
11. D. Chaum, "The dining cryptographers problem: unconditional sender and recipient untraceability," *Journal of Cryptology*, Vol. 1, No. 1, pp. 65–67, 1988.
12. J. Camenisch and M. Stadler, "Proof systems for general statements about discrete logarithms," Technical report TR 260, Department of Computer Science, ETH Zürich, March 1997.
13. S. Bellovin and M. Merritt, "Encrypted Key Exchange: password-based protocols secure against dictionary attacks," Proceedings of the IEEE Symposium on Research in Security and Privacy, May 1992.
14. S. Bellovin and M. Merritt, "Augmented Encrypted Key Exchange: a password-based protocol secure against dictionary attacks and password file compromise," Proceedings of the 1st ACM Conference on Computer and Communications Security, pp. 244–250, November 1993.
15. S. Bellovin and M. Merritt, "Cryptographic protocol for secure communications," U.S. Patent 5,241,599. `http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=5241599`
16. E. Ehulund, "Secure on-line configuration for SIP UAs," Master thesis, The Royal Institute of Technology, August 2006.
17. W. Ford, B.S. Kaliski, "Server-assisted generation of a strong secret from a password," Proceedings of the 9th International Workshops on Enabling Technologies, pp. 176–180, IEEE Press, 2000
18. I. Foster, C. Kesselman, G. Tsudik , S. Tuecke, "A security architecture for computational grids," Proceedings of the 5th ACM conference on Computer and Communications Security, pp.83–92, November 1998.
19. C. Gentry, P. MacKenzie, Z. Ramzan, "A method for making password-based key exchange resilient to server compromise," Crypto'06, LNCS 4117, pp. 142–159, 2006.
20. F. Hao, P. Zieliński, "A 2-round anonymous veto protocol," Proceedings of the 14th International Workshop on Security Protocols, SPW'06, Cambridge, UK, May 2006.
21. J. Juels, J. Brainard, "Client Puzzles: a cryptographic countermeasure against connection depletion attacks," Proceedings of Networks and Distributed Security Systems, pp. 151–165, 1999.
22. D. Jablon, "Strong password-only authenticated key exchange," *ACM Computer Communications Review*, Vol. 26, No. 5, pp. 5–26, October 1996.
23. D. Jablon, "Extended password protocols immune to dictionary attack", Proceedings of the WETICE'97 Enterprise Security Workshop, pp. 248–255, June 1997.
24. D. Jablon, "Cryptographic methods for remote authentication," U.S. Patent 6,226,383. `http://patft.uspto.gov/netacgi/nph-Parser?patentnumber=6226383`
25. D. Jablon, "Password authentication using multiple servers," Topics in Cryptology – CT-RSA, pp. 344–360, LNCS 2020, April 2001.
26. B. Jaspan, "Dual-workfactor Encrypted Key Exchange: efficiently preventing password chaining and dictionary attacks," Proceedings of the Sixth Annual USENIX Security Conference, pp. 43-50, July 1996.
27. K. Kobara, H. Imai, "Pretty-simple password-authenticated key-exchange under standard assumptions," *IEICE Transactions*, Vol. E85-A, No. 10, pp. 2229–2237, 2002.

28. Paul C. Van Oorschot, M.J. Wiener, "On Diffie-Hellman key agreement with short exponents," Advances in Cryptology, EUROCRYPT'96, LNCS 1070, pp. 332–343, 1996.
29. S. Patel, "Number theoretic attacks on secure password schemes," Proceedings of the IEEE Symposium on Security and Privacy, May 1997.
30. R. Perlman, C. Kaufman, "Secure password-based protocol for downloading a private key," Proceedings of the Network and Distributed System Security, February 1999.
31. P. MacKenzie, T. Shrimpton and M. Jakobsson, "Threshold password-authenticated key exchange," CRYPTO'02, August 2002.
32. p. MacKenzie, "The PAK suite: protocols for password-authenticated key exchange," Technical Report 2002-46, DIMACS, 2002.
33. P. MacKenzie, "On the Security of the SPEKE Password-Authenticated Key Exchange Protocol," Cryptology ePrint Archive: Report 057, 2001.
34. IEEE P1363 Working Group, P1363.2: Standard Specifications for Password-Based Public-Key Cryptographic Techniques. Draft available at:
`http://grouper.ieee.org/groups/1363/`
35. J.F Raymond, A. Stigic, "Security issues in the diffie-hellman key agreement protocol," Technical report, Zeroknowledge Inc., September 2000.
36. D. Stinson, *Cryptography: theory and practice*, Third Edition, Chapman & Hall/CRC, 2006.
37. C.P. Schnorr, "Efficient signature generation by smart cards," Journal of Cryptology, Vol. 4, No. 3, pp. 161–174, 1991.
38. Muxiang Zhang, "Analysis of the SPEKE password-authenticated key exchange protocol," *IEEE Communications Letters*, Vol. 8, No. 1, pp. 63-65, January 2004.
39. Z. Zhao, Z. Dong, Y. Wang, "Security analysis of a password-based authentication protocol proposed to IEEE 1363," *Theoretical Computer Science*, Vol. 352, No. 1, pp. 280–287, 2006.