PASSWORD AUTHENTICATED KEY EXCHANGE: FROM TWO PARTY METHODS TO GROUP SCHEMES

STEPHEN MELCZER, TARAS MYCHASKIW AND YI ZHANG

Abstract. This project investigates group password authenticated key exchange methods (GPAKEs). In particular, we detail the so-called 'fairy ring dance' method recently described by Hao et. al. [8] which allows for the extension of two party password authenticated key exchange methods (PAKEs) with explicit key confirmation to a group setting with an arbitrary number of users without increasing round complexity (the computational complexity, of course, increases with the number of users). This paper presents two new GPAKEs constructed through these means, based on the Dragonfly and PAK/PPK two party protocols, and includes timings comparing them to previous GPAKEs of Hao et. al. [8].

1. Introduction

Since their introduction in the 1990s, password-authenticated key exchange (PAKE) methods – also known as password-authenticated key agreement methods – have become popular for their ability to allow agents sharing a (typically low entropy) password to securely establish shared cryptographic keys (see Bellovin and Merritt [4] or Jablon [10] for early examples, and Hao and Ryan [7] for a more recent paper). Although they have been around for decades, most research on PAKEs has focused on key establishment between two parties. For our project, we have studied the problem of establishing Group PAKEs (GPAKEs) – that is, using a low entropy password shared between many agents to set up cryptographic keys. This has modern applications with the rise of the so-called 'Internet of Things', where many consumer devices connected through a local Internet connection wish to securely communicate (such schemes would allow, for instance, secure communication between a smart television, DVD player, and cable box after their owner inputs a short shared password into each upon purchase).

The major issue in designing an efficient GPAKE is to minimize the number of rounds of communication between the agents involved, as the latency of such a protocol is determined by the slowest responder in each round. A recent pre-print of Hao et. al. [8] proposes a construction which allows for the extension of any secure two-party PAKE with key confirmation to a multi-party PAKE, without adding any extra rounds of communication (if the underlying two-party PAKE only allows for key authentication, then the associated GPAKE has one extra round of communication). The authors continue on to give two explicit schemes following from this template: SPEKE+ (using two rounds of communication, adapted from the SPEKE [10] protocol) and J-PAKE+ (using three rounds of communication, adapted from the J-PAKE [7] protocol).

The structure of this document is as follows: Section 2 begins by giving a survey of classical two-party PAKEs – including explicit descriptions of the PAKEs which will be extended into the group setting and background information on the zero knowledge proof protocols these use. Section 3 starts with a description of the theoretical methodology developed by Hao et. al. [8] to extend two-party PAKEs into a group setting. After this theoretical background, we give two explicit GPAKEs (SPEKE+ and J-PAKE+) constructed by Hao et. al. using this methodology, followed by two explicit GPAKEs which we have derived through the same means (a group variant of the IEEE 802.11-2012 standard Dragonfly protocol [9], and a variant of the PAK/PPK protocol [5]). Security properties of these GPAKEs follow from the security properties of the underlying two party PAKES, in a manner described by Hao et. al. [8] and in Section 3 of this document. In Section 4 we test the practical efficacy of our new methods against the Java implementations of SPEKE+ and J-PAKE+ given by Hao et. al. [8]. Section 5 concludes with an overview of these results and possible directions for future work.

The main original contributions found in this project come from the two new group PAKEs we have constructed – see sub-Sections 2.3 and 2.4 – and timings which compare these methods against previous Java implementations of SPEKE+ and J-PAKE+ (the code for this project is available at [ADD GITHUB LINK]). We also survey the relevant background material on Zero Knowledge Proofs and classical PAKEs missing from Hao et. al. [8] (which had constrained space as a conference abstract), and fixed some minor Java implementation errors which could cause the timings in that paper to be slightly inaccurate.

2. Two Party Password-Authenticated Key Exchange (PAKE) [1]

The genesis of password-authenticated key exchange is widely credited to the 1992 work of Bellovin and Merrit [4], whose protocol – known as Encrypted Key Exchange, or EKE, for short – came to be known as the first PAKE (previous password based protocols, like the one proposed in 1989 by Lomas et.al. [11], contained key features of PAKEs such as the offline dictionary attack resistance detailed below, although they still relied on one party having another's public key). All PAKEs aim for two main goals: to require their users to provide a zero knowledge proof of a short password known to both parties *a priori* (that is, before the protocol has begun) and to leverage knowledge of this password to facilitate an authenticated key exchange. As password are assumed to be low entropy – for instance, they are often treated as human memorable passwords (typically assumed to be approximately 20-30 bits of entropy) – if the passwords themselves were broadcast they would need to be protected, for instance using SSL. This would require Public Key Infrastructure, such as a Trusted Authority / Certificate Authority to maintain public keys, which can be expensive. The ability to work around such infrastructure is often the point of PAKE protocols, which essentially use pre-established shared knowledge (of the common password) as an alternative to Trusted Authorities.

Indeed, it is somewhat miraculous that PAKEs – which transform a low entropy shared secret into a much larger and more complicated shared key – exist at all. Although the EKE protocol of Bellovin and Merrit was later shown to have weaknesses (see Jaspan [10], for example) its great contribution was to show that such schemes can be achieved. Due to its historical significance, we outline the Diffie-Hellman variant of the method here (an RSA variant, also by Bellovin and Merrit, was later shown to be insecure). Given a symmetric encryption function $[\cdot]_\pi$ which uses a password π shared by agents Alice and Bob as a key, the algorithm does the following:

Setup: Let α be a primitive root of $\mathbb{Z}_p$

Alice		Bob
randomly choose $x_a \in \mathbb{Z}_p^*$		randomly choose $x_b \in \mathbb{Z}_p^*$
	$\xrightarrow{[\alpha^{x_a}]_\pi}$	
	$\xleftarrow{[\alpha^{x_b}]_\pi}$	
Compute $K = \alpha^{x_a \cdot x_b}$		Compute $K = \alpha^{x_a \cdot x_b}$

Figure 1. The flow diagram for EKE

At the end, both Alice and Bob share the key $K = \alpha^{x_a \cdot x_b}$. The weaknesses of the algorithm stems from the issues discussed above: as the password has low entropy in order for the scheme to be secure the input into $[\cdot]_\pi$ must essentially look like a random number. But a 1024 bit number modulo p is not random, and a passive attacker can try candidate passwords π' to attempt to decipher $[\alpha^{x_a}]_\pi$ and immediately rule out any passwords giving a result in the range $[p, 2^{1024} - 1]$.

[1]The background information in this section, and details about SPEKE and J-PAKE, are mainly based on the presentation in Hao and Ryan [7]. The information on the Dragonfly protocol was taken from Harkins [9] and Clarke and Hao [6]. The sub-section about PAK/PPK is based on the work of Boyko et. al. [5].

Although EKE has this, and other, weaknesses, it was extremely influential and its general characteristics are reflected in many of the more advanced protocols we outline below (there is also a minor variant known as EKE2, which was shown to be secure by Bellare, Pointcheval, and Rogaway [3]). Before giving these methods, we must outline what constitutes a good measure of security for a PAKE. To begin, a secure two party PAKE satisfies each of the following four properties (which correspond to desired properties of general key exchange protocols):

> **(Offline dictionary attack resistance)**
> The PAKE does not leak any information to a passive or active attacker which can be used by the attacker to determine the password through a brute force search (the protocol cannot reveal a hash of the password, for instance).
>
> **(Forward secrecy for established keys)**
> If the password is disclosed, past session keys cannot be computed by an attacker. This implies that a *passive* attacker who knows the password he cannot learn a session key by observing communication between Alice and Bob (of course, an active attacker could establish a shared key with one of the participants as he would have access to all of their secret information).
>
> **(Known session security)**
> If an attacker learns all secrets of a protocol in progress, it does not reveal any information about other established sessions.
>
> **(Online dictionary attack resistance)**
> An active attacker can only test one password per protocol execution (this is the best that we can reasonably assume, as any attacker can randomly guess a password and run the protocol – at some point the key must be confirmed, either explicitly through the PAKE or when the key is used in some other protocol, and the attacker will know whether or not his guess was correct).

These properties are illustrated in sub-Section 2.2 when we outline their proof for the J-PAKE protocol. In the modern literature, a full proof of security essentially requires showing that an attacker can only gain information about established keys or a shared password if he is active, and that even an active attacker can gain extremely little information (for instance, can only guess one password per protocol execution). The formal model of Bellare, Pointcheval, and Rogaway [3] is commonly used as a standard, and three of the four PAKEs discussed later in this section have been proven secure – under various assumptions, see Figure 2 – in this model (the fourth, which is the Dragonfly protocol of sub-Section 2.3, we include despite a formal proof of security as it is an IEEE 802.11-2012 standard). We refer an interested reader to the work of Abdalla et. al. [2] and MacKenzie [13] for in-depth discussions of PAKE security and attack models.

		Assumptions[a]			Complexity	
	Rounds / Flows	ROM	AAM		Communication[b]	Time[c]
J-PAKE with Schnorr	2 / 4 or 3 / 3	✗	✗	DSDH or (CSDH + DDH)	$12 \times G + 6 \times \mathbb{Z}_p$	28 exp
SPEKE	1 / 2	✗		DIDH[d]	$2 \times G$	8 exp
PPK	2 / 2	✗		DDH	$2 \times G$	6 exp

[a] *CRS*: common reference string, *ROM*: random-oracle model, *ICM*: ideal-cipher model, *AAM*: algebraic-adversary model;
[b] G: group elements, $\mathbb{Z}_p$: scalars;
[c] *exp*: number of exponentiations;
[d] *DIDH*: decision inverted-additive Diffie-Hellman assumption

FIGURE 2. Table taken from Abdalla et. al. [2] comparing the security assumptions and complexity of the methods discussed below

[Re-do table ourselves] *

Some PAKEs satisfy the additional requirement that an attacker not be allowed to impersonate other users to some fixed target after obtaining (through illicit means) password verification files for those users which were stored by the target. The schemes with this additional property are known as augmented PAKEs, although some (for instance, Hao et. al. [8]) have argued that such a requirement is not useful as the low entropy of the password means that it will soon be discovered through an offline dictionary attack on the verification files. Nevertheless, augmented variants exist for a number of balanced PAKEs (for example Augmented-EKE for EKE, B-SPEKE for SPEKE and PAK-X for PAK/PPK).

2.1. **SPEKE.** A more advanced PAKE which we consider is the Simple Password Exponential Key Exchange (SPEKE) protocol, designed by Jablon [10] in 1996. SPEKE tries to work around the deficiencies of EKE by using the shared password π of the two participants to change the generator of a Diffie-Hellman like scheme. The protocol runs as follows:

Setup: Let $p = 2q + 1$ where p and q are prime

$$
\begin{array}{ccc}
\text{Alice} & & \text{Bob} \\
\hline
\text{randomly choose } x_a \in \mathbb{Z}_q^* & & \text{randomly choose } x_b \in \mathbb{Z}_q^* \\
 & \xrightarrow{\left(\pi^2\right)^{x_a}} & \\
 & \xleftarrow{\left(\pi^2\right)^{x_b}} & \\
\text{Compute } K = \pi^{2 \cdot x_a \cdot x_b} & & \text{Compute } K = \pi^{2 \cdot x_a \cdot x_b}
\end{array}
$$

FIGURE 3. The flow diagram for SPEKE

Note that the password is squared so that the exponentiations in the protocol can occur in a subgroup of prime order q (the participants must check that that $\pi^2 \not\equiv \pm 1 \bmod p$ – if this is not the case, all work is carried out in the order 2 subgroup of $\mathbb{Z}_p^*$ which renders the protocol insecure, and a new password or prime p must be chosen).

There are drawbacks to using the password directly: mainly that an attacker can guess multiple passwords in one execution of the protocol (as multiple passwords may have the same square mod p) and that the size of the subgroup in which the protocols occur is large (if p is a 1024-bit prime, for example, then q is a 1023-bit prime). Although it is troubling to allow an active attacker multiple guesses at the password, for practical purposes as long as they are limited to a small constant number of guesses per protocol execution the security of the protocol can be safely assumed. Indeed, a variant of the basic protocol presented in Figure 3 where a hash of the password is squared (as defined by the IEEE P1363.2 standard regarding SPEKE[2]) was later proven secure by MacKenzie [12] in a common formal model (proposed by Boyko, MacKenzie and Patel [5]) under the assumptions of the random-oracle model and the hardness of the decision inverted-additive Diffie-Hellman (DIDH) problem[3]. Here, the notion of 'secure' is relaxed to allow an active attacker to rule out a (small) constant number of guesses per protocol execution.

[2] Our implementation of SPEKE uses this variant

[3] This somewhat non-standard Diffie-Hellman assumption asks one to distinguish between an element $g^{(x+y)^{-1}}$ and a random group element, given the elements $X = g^{x^{-1}}$ and $Y = g^{y^{-1}}$. It has been shown that if the typical computational Diffie-Hellman problem (CDH) is hard, so is the computational inverted-additive Diffie-Hellman problem. Furthermore, if the Decision Square Diffie-Hellman problem is hard (which is assumed in the security proof of the J-PAKE protocol, discussed below) then the DIDH problem is hard. See Figure 2 of Abdalla et. al. [2] for a comparison of all the Diffie-Hellman type assumptions used in the protocols presented here.

2.2. **J-PAKE.** We start this protocol by describing the Password Authenticated Key Exchange by Juggling (J-PAKE) method of Hao and Ryan [7]. Let G be a subgroup of $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$ with prime order q, where the Decision Diffie-Hellman (DDH) problem is considered intractable. Let $g \in G$ be a generator of the subgroup and $s \in [1, q-1]$ be the shared password between Alice and Bob, where $[a, b]$ is notation meaning the elements of $\mathbb{Z}^*$ between a and b: $[a, b] := \{a, a+1, \ldots, b\}$.

The protocol consists of a setup round followed by two rounds of communication (for details on zero knowledge proofs see sub-Section **??** – in our implementation detailed later the XXX ZKP is used [Which? Schnorr?]):

$$
\begin{array}{ccc}
\underline{\text{Alice}} & & \underline{\text{Bob}} \\
\text{Pick } x_1 \in_R \mathbb{Z}_q \text{ and } x_2 \in_R \mathbb{Z}_q^* & & \text{Pick } x_3 \in_R \mathbb{Z}_q \text{ and } x_4 \in_R \mathbb{Z}_q^* \\
 & \xrightarrow{g^{x_1}, g^{x_2}, ZKP\{x_1\}, ZKP\{x_2\}} & \\
 & \xleftarrow{g^{x_3}, g^{x_4}, ZKP\{x_3\}, ZKP\{x_4\}} & \\
\text{Verify } ZKP\{x_3\}, ZKP\{x_4\} \text{ and } g^{x_4} \neq 1 & & \text{Verify } ZKP\{x_1\}, ZKP\{x_2\} \text{ and } g^{x_2} \neq 1 \\
 & \xrightarrow{A = g^{(x_1+x_3+x_4)x_2 \cdot s} \text{ and } ZKP\{x_2 s\}} & \\
 & \xleftarrow{B = g^{(x_1+x_2+x_3)x_4 \cdot s} \text{ and } ZKP\{x_4 s\}} & \\
\text{Verify } ZKP\{x_4 s\} & & \text{Verify } ZKP\{x_2 s\} \\
K = (B/g^{x_2 x_4 s})^{x_2} & & K = (A/g^{x_2 x_4 s})^{x_4}
\end{array}
$$

FIGURE 4. The J-PAKE protocol.

At this point, both are able to determine

$$K = \underbrace{(B/g^{x_2 x_4 s})^{x_2}}_{\text{Computable by Alice}} = g^{(x_1+x_3)x_2 x_4 s} = \underbrace{(A/g^{x_2 x_4 s})^{x_4}}_{\text{Computable by Bob}}.$$

The shared session key is then taken to be $\kappa = H(K)$, where H is a hash function. [Why do they hash here?] Note that this scheme has implicit key confirmation: each of Alice and Bob believe only the other can calculate the shared key; an explicit key confirmation can then be performed if desired (which will increase the number of rounds of communication by at least one).
We now show that the J-PAKE protocol satisfies the four properties required to be considered a secure PAKE, in order to illustrate the properties and the methods by which they can be proven: without loss of generality we may assume that Alice is honest, and let $x_a := x_1 + x_3 + x_4$.

Lemma 2.1. *With high probability (approximately 2^{-160} for q a 160-bit prime) the element $g_a := g^{x_a}$ is a generator for the subgroup G, and Alice can verify this.*

Proof. Since $|G| = q$ is prime, it is sufficient to prove that $g_a \neq 1$ (any non-identity element generates G) with high probability. As Alice verifies that x_3 and x_4 are known to Bob due to his zero knowledge proofs in round 1, and x_1 is chosen randomly by Alice, x_a must be a random value from Bob's perspective. In other words, $x_a \neq 0$ with high probability, even if Bob is an active adversary. Alice can verify that g_a is a generator for g as she knows the value of x_a. □

An analogous proof shows that when Bob is honest the element $g_b := g^{x_1+x_2+x_3}$ is a generator of G with high probability. We first show resistance to an offline attack with an active adversary.

Theorem 2.2 (Offline resistance to active attack)**.** *Under the DDH assumption, when g_a is a generator of G any attacker Oscar cannot distinguish Alice's ciphertext from a random non-identity element in the subgroup G.*

Proof. Suppose that Alice communicates with Oscar, who does not know the password. After the protocol is run, Oscar knows

$$g^{x_1}, g^{x_2}, A = g_a^{x_2 s}, \text{ and ZKPs of the exponents } x_1 \text{ and } x_2.$$

By definition, with high probability the zero knowledge proofs reveal only one bit of information: that Alice knows the values of the exponents. As argued in the proof of the last lemma, with high probability g_a is a (random) generator of the group G. Furthermore, as x_2 is chosen randomly it follows that $x_2 s \in [1, q-1]$ is random and thus unknown to Oscar. Thus, the only way to distinguish A from a random non-identity element would be for Oscar to solve an instance of the Decision Diffie-Hellman problem. □

The result in the case of a passive attacker follows in a straightforward manner (note that this case must be proven separately as above the active attacker does not know the password s, but when he passively observes a session Alice is communicating with Bob, who does know s).

Theorem 2.3 (Offline resistance to passive attack)**.** *Under the DDH assumption, given that g_a and g_b are generators of G, the ciphertexts*

$$A = g_a^{x_2 s} \text{ and } B = g_b^{x_4 s}$$

do not leak information for password verification.

Proof. Our above work shows that the value A looks random to Bob, and analogously that B looks random to Alice. Thus, both must look random to a passive adversary, who has less information about the protocol's secrets than either Alice or Bob. □

Next we show forward secrecy under the assumption that the Square Computational Diffie-Hellman problem is hard (this problem, which has been shown to be equivalent to the Computational Diffie-Hellman problem, asks one to compute g^{a^2} given the value g^a with a some unknown value). Since the zero-knowledge proofs imply that Alice and Bob know the values of x_1 and x_3, with high probability $x_1 + x_3 \neq 0$ in $\mathbb{Z}_q$ – which implies $K = g^{(x_1+x_3)x_2x_4s} \neq 1$ with high probability, even if one of Alice or Bob is an active adversary.

Theorem 2.4 (Forward Secrecy)**.** *Under the Square Computational Diffie-Hellman assumption, when $K \neq 1$, past session keys derived from the protocol remain incomputable even when s is later disclosed.*

Proof. Knowing s, the attacker wants to compute $\kappa = H(K)$ given

$$\{g^{x_1}, g^{x_2}, g^{x_3}, g^{x_4}, g^{(x_1+x^3+x_4)x_2}, g^{(x_1+x_2+x_3)x_4}.\}$$

Suppose the attacker can compute K, and thus $g^{(x_1+x_3)x_2x_4}$ from the above information – we show how he can act as an oracle to solve the Square Computational Diffie-Hellman problem. Let $x_5 = x_1 + x_3 \bmod q$ (which is non-zero when $K \neq 1$). Then ... □

[To be continued]

2.3. **Dragonfly.** Dragonfly is based on discrete logarithm cryptography, which means one can use operations either in a finite field or an elliptic curve. In it's definition in [9], no assumptions are made about the underlying group, only that calculating discrete logarithms is difficult enough to provide a baseline level of security.

As an example execution of the protocol, we will look at the finite field case. Let p be a large prime. We will denote Q as a cyclic subgroup of $\mathbb{Z}_p^*$ with prime order q – hence $q|p-1$. In addition to p and q, a hash function H is also agreed upon. The protocol then executes as follows:

(1) Alice and Bob have a shared password which they both map to an element $\pi \in Q$. The protocol specification maps the password arbitrarily (but deterministically) to the element π, and includes some example algorithms to perform the actual mapping. These examples are omitted here.
(2) Alice chooses two random values $r_A, m_A \in_R \mathbb{Z}_q^*$. She computes $s_A = r_A + m_A \mod q$ and the element $E_A = \pi^{-m_A} \mod p$. If $s_A < 2$ (to avoid the small subgroup attack), start this step over. She sends s_A and E_A to Bob.
(3) Bob chooses two random values $r_B, m_B \in_R \mathbb{Z}_q^*$. He computes $s_B = r_B + m_B \mod q$ and the element $E_B = \pi^{-m_B} \mod p$. If $s_B < 2$, start this step over. He sends s_A and E_A to Alice.
(4) Each member verifies that one of $E_A \neq E_B$ or $s_A \neq s_B$ is true to avoid a reflection attack.
(5) Alice computes the shared secret $ss = (\pi^{s_B} E_B)^{r_A} = \pi^{r_A r_B} \mod p$. Alice sends $A = H(ss||E_A||s_A||E_B||s_B)$ to Bob.

(6) Bob computes the shared secret $ss = (\pi^{s_A} E_A)^{r_B} = \pi^{r_A r_B} \mod p$. Bob sends $B = H(ss||E_B||s_B||E_A||s_A)$ to Alice.

(7) Alice and Bob both confirm the received hash values are correct and compute the shared key $K = H(ss||E_A \times E_B||(s_A + s_B) \mod q)$.

This protocol is illustrated in Figure 5.

Alice		Bob
repeat: randomly choose $r_A, m_A \in \mathbb{Z}_q^*$		repeat: randomly choose $r_B, m_B \in \mathbb{Z}_q^*$
$s_A = r_A + m_A \mod q$ until $s_A \geq 2$		$s_B = r_B + m_B \mod q$ until $s_B \geq 2$
$E_A = \pi^{-m_A} \mod p$	$\xrightarrow{s_A, E_A}$	$E_B = \pi^{-m_B} \mod p$
	$\xleftarrow{s_B, E_B}$	
Verify $E_A \neq E_B$ or $s_A \neq s_B$		Verify $E_A \neq E_B$ or $s_A \neq s_B$
$ss = (\pi^{s_B} E_B)^{r_A} = \pi^{r_A r_B} \mod p$		$ss = (\pi^{s_A} E_A)^{r_B} = \pi^{r_A r_B} \mod p$
$A = H(ss\|\|E_A\|\|s_A\|\|E_B\|\|s_B)$	$\xrightarrow{A}$	$B = H(ss\|\|E_B\|\|s_B\|\|E_A\|\|s_A)$
	$\xleftarrow{B}$	
Verify B		Verify A
$K = H(ss\|\|E_A \times E_B\|\|(s_A + s_B) \mod q)$		$K = H(ss\|\|E_A \times E_B\|\|(s_A + s_B) \mod q)$

Figure 5. The Dragonfly protocol.

Steps 2 and 3 were modified in the most recent update to the Dragonfly protocol. A small subgroup attack was discovered in [6]. The protocol works around this now by checking s_A and s_B and repeating the step until the values generated are safe.

Dragonfly claims to be resistant to offline dictionary attacks, but does not provide any security proofs. However, due to the small exponentiation size (limited by q), it is very quick. As our results show in Section 4, Dragonfly+ is the fastest protocol tested.

2.4. **PAK/PPK.** The PAK/PPK protocols were first introduced by Boyko, MacKenzie and Patel in [5] in 2000 as a Diffie-Hellman based provably secure PAKEs. The PAK protocol contained explicit key confirmation while PPK did not. An augmented version of it, namely PAK-X, was also introduced in the same paper.

The setting of PAK/PPK is similar to all other Diffie-Hellman based PAKEs. What is different is it's dependencies on perfect hash functions.

Let π be the password. Let p, q be large primes and $p = rq + 1$, where r, q are relatively prime. Let g be a generator of a subgroup of $\mathbb{Z}_p^*$ of size q where the Decision Diffie-Hellman (DDH) problem is infeasible. Let H_1, H_{2a}, H_{2b}, H_3 be independent random hash functions. The PAK and PPKprotocols are described as in Figures 2.4 and 2.4. [Yi: Any idea what's going on here?] *

As for the security of the protocols, Boyko, MacKenzie and Patel has developed a new formal model for PAKEs in [5], with which they have proved that PAK is secure in the random oracle model and PPK is secure in the implicit-authentication model, provided DDH is intractable. The newly proposed model and definition of security does not correspond directly to the four properties mentioned earlier in this section. However, it can be extrapolated from the proof that under the appropriate model and assumptions, the protocols are indeed able to provide forward secrecy as well as secure against online and offline dictionary attacks.

3. Group Password-Authenticated Key Exchange (GPAKE)

Background info from paper

$$
\begin{array}{lcr}
A & & B \\
x \in_R Z_q & & \\
m = g^x \cdot (H_1(A,B,\pi))^r & \xrightarrow{\quad m \quad} & \text{Test } m \overset{?}{\not\equiv} 0 \bmod p \\
 & & y \in_R Z_q \\
 & & \mu = g^y \\
 & & \sigma = (\frac{m}{(H_1(A,B,\pi))^r})^y \\
\sigma = \mu^x & \xleftarrow{\quad \mu, k \quad} & k = H_{2a}(A,B,m,\mu,\sigma,\pi) \\
\text{Test } k \overset{?}{=} H_{2a}(A,B,m,\mu,\sigma,\pi) & & \\
k' = H_{2b}(A,B,m,\mu,\sigma,\pi) & & \\
K = H_3(A,B,m,\mu,\sigma,\pi) & \xrightarrow{\quad k' \quad} & \text{Test } k' \overset{?}{=} H_{2b}(A,B,m,\mu,\sigma,\pi) \\
 & & K = H_3(A,B,m,\mu,\sigma,\pi)
\end{array}
$$

FIGURE 6. The PAK protocol.

$$
\begin{array}{lcr}
A & & B \\
x \in_R Z_q & & \\
m = g^x \cdot (H_1(A,B,\pi))^r & \xrightarrow{\quad m \quad} & \text{Test } m \overset{?}{\not\equiv} 0 \bmod p \\
 & & y \in_R Z_q \\
 & & \mu = g^y \cdot (H_1(A,B,\pi))^r \\
 & & \sigma = (\frac{m}{(H_1(A,B,\pi))^r})^y \\
\text{Test } \mu \overset{?}{\not\equiv} 0 \bmod p & \xleftarrow{\quad \mu \quad} & K = H_3(A,B,m,\mu,\sigma,\pi) \\
\sigma = (\frac{\mu}{(H_1(A,B,\pi))^r})^x & & \\
K = H_3(A,B,m,\mu,\sigma,\pi) & &
\end{array}
$$

FIGURE 7. The PPK protocol.

3.1. **SPEKE+ and J-PAKE+.** Info about SPEKE+ and J-PAKE+

3.2. **Dragonfly+.**

- implementation details (how password was mapped, source of primes, etc)
- change of small subgroup attack – we don't restart the step, instead the other person verifies the value received
- brief overview of d+ – can mostly reference dragonfly and general construction
- brief mention security

We present the group extension to the Dragonfly protocol using the general construction outlined in [8]. The construction follows the Dragonfly protocol closely, with only minor modifications to the first round of communication and of course adding the final round to establish the group key. The setup is the same as Dragonfly (see Section 2.3), except the generator of the subgroup Q is required, called g. The Dragonfly+ protocol executes as follows:

Setup: Every participant P_i have a shared password which they map to an element $\pi \in Q$.

Round 1: Every participant selects $r_{ij}, m_{ij} \in_R \mathbb{Z}_q^*$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$. They each compute $s_{ij} = r_{ij} + m_{ij} \mod q$ and the element $E_{ij} = \pi^{-m_{ij}} \mod p$. If any $s_{ij} < 2$, start this step over. Each member then also selects $y_i \in_R \mathbb{Z}_q$. P_i then broadcasts $s_{ij}, E_{ij}, g^{y_i} \mod p$ and ZKP$\{y_i\}$. [maybe just say each member performs a pairwise round 1 with each other member, and does y_i]

Define $z_i = y_{i+1}/y_{i-1}$ (with cyclic index i). Each member is able to compute $g^{z_i} = g^{y_{i+1}}/g^{y_{i-1}}$, and check:

- $g^{z_i} \neq 1 \mod p$;
- One of $E_{ij} \neq E_{ji}$ or $s_{ij} \neq s_{ji}$ is true for all $j \in \{1, \ldots, n\} \setminus \{i\}$;
- the received $\mathrm{ZKP}\{y_j\}$ for all $j \in \{1, \ldots, n\} \setminus \{i\}$ is valid.

Round 2 [compute shared secret, send hash] [verify hash]

Round 3 [compute pairwise key, group stuff]

***

3.3. **PPK+.** In this section, we present the Group PAKE extension of the PPK protocol, the simpler of the PAK/PPK suite, using the Fairy Ring Dance construction from [8]. The PPK+ protocol is described as follows:

Round 1: Every participant P_i selects $x_i \in_R \mathbb{Z}_q$ and $y_i \in_R \mathbb{Z}_q$ and broadcasts $m_{ij} = g^{x_i} \cdot (H_1(i, j, \pi))^r$ for all $j \neq i$ as well as g^{y_i} together with a zero-knowledge proof, denoted as $\mathrm{ZKP}\{y_i\}$, for proving the knowledge of the exponent y_i.

We define $z_i = y_{i+1} - y_{i-1}$. Then everyone is able to compute $g^{z_i} = g^{y_{i+1}}/g^{y_{i-1}}$. Every participant P_i also computes $\sigma_{ij} = \left(\frac{m_{ji}}{H_1(j,i,\pi)^r}\right)^{x_i} = g^{x_i x_j}$ and checks:

- $g^{z_i} \neq 1$ for $i = 1, \ldots, n$.
- $m_j \neq 0$ for $j = 1, \ldots, n, j \neq i$.
- the received $\mathrm{ZKP}\{y_j\}$ for $j = 1, \ldots, n, j \neq i$ are valid.

Similar to all GPAKE constructions, the $\mathrm{ZKP} y_i$ are standard Schnorr non-interactive zero knowledge proofs outlined in [8].

Round 2: Every participant P_i broadcasts $(g^{z_i})^{y_i}$ and a zero knowledge proof, $\mathrm{ZKP}\{\tilde{y_i}\}$ for providing the equality of the discrete logarithm of $(g^{z_i})^{y_i}$ to the base g^{z_i} and the discrete logarithm g^{y_i} to the base g. Everyone then computes the raw pairwise keys K_{ij} according to PPK, namely $K_{ij} = H_3(i, j, m_{ij}, m_{ji}, \sigma_{ij}, \pi)$, and the derived authentication and confirmation keys, $\kappa^{\mathrm{MAC}} = H(K_{ij}, \text{“MAC”})$, $\kappa^{\mathrm{KC}} = H(K_{ij}, \text{“KC”})$. Furthermore, let $A_{ij} = g^{y_i} \mid\mid \mathrm{ZKP}\{y_i\} \mid\mid K_{ij} \mid\mid \mathrm{ZKP}\{\tilde{y_i}\}$, P_i broadcast $t_{ij}^{\mathrm{MAC}} = \mathrm{HMAC}(\kappa_{ij}^{\mathrm{MAC}}, A_{ij})$ and $t_{ij}^{\mathrm{KC}} = \mathrm{HMAC}(\kappa_{ij}^{\mathrm{KC}}, \text{“KC”} \mid\mid i \mid\mid j \mid\mid m_i \mid\mid m_j)$ for each $j \neq i$.

When this round finishes, everyone checks:

- the received $\mathrm{ZKP}\{\tilde{y_j}\}$ for $j = 1, \ldots, n, j \neq i$ are valid.
- the received key confirmation strings t_{ji}^{KC} for $j = 1, \ldots, n, j \neq i$ are valid.
- the received message authentication tags t_{ji}^{MAC} for $j = 1, \ldots, n, j \neq i$ are valid.

Again, the zero knowledge proofs are standard Chaum-Pedersen ZKP used in [8].

At the end of the two rounds, the same formula is used for calculating the group key. [Yi: Steve please put the formula in section 3.1 and I'll reference it here.] *

The security of the GPAKE protocol follows directly from the security of the two party protocol PPK.

For simplicity of our demonstration, there are a few tweaks we made in our implementation of the protocol. Firstly, the supposedly independent random hash functions H_1 and H_3 are implemented as *SHA-256* shifted by two different constants. Obviously it has security implications on the protocol, but it will certainly not be used in real life implementations, nor does it have any impact on the run time performance that we are measuring.

Secondly, it should be noted that the formula for calculating the raw pairwise keys K_{ij} at the end of round 1 is not symmetric between i and j. A modification is therefore made such that when calculating K_{ij}, we set $i < j$ without loss of generality. In this case both parties would be able to calculate the same pairwise key.

4. Implementation Results

All of the protocols were implemented in Java 6 on a server (3GHz AMD processor, 6GB of RAM) running Ubuntu 12.04.

5. Conclusion

The conclusion

References

[1] NIST cryptographic toolkit, July 2014. http://csrc.nist.gov/groups/ST/toolkit/.

[2] M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the j-pake password-authenticated key exchange protocol. In *SP*, 2015.

[3] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *LNCS, B. Preneel, Ed., vol. 1807. Springer*, pages 139 – 155. EUROCRYPT 2000, May 2000.

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

[5] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *LNCS 1807,Springer-Verlag, Berlin*, pages 156–171. Eurocrypt 2000, 2000.

[6] Dylan Clarke and Feng Hao. Cryptanalysis of the dragonfly key exchange protocol. In *IET Information Security*, pages 283–289, 2014.

[7] F. Hao and P. Ryan. J-PAKE: Authenticated key exchange without PKI. *Transactions on Computational Science XI, Lecture Notes in Computer Science*, 6480:192–206, 2010.

[8] F. Hao, X. Yi, L. Chen, and S. F. Shahandashti. The fairy-ring dance: Password authenticated key exchange in a group. Cryptology ePrint Archive, Report 2015/080, 2015. http://eprint.iacr.org/2015/080.

[9] D. Harkins. Dragonfly key exchange – internet research task force internet draft, 2015. http://datatracker.ietf.org/doc/draft-irtf-cfrg-dragonfly/.

[10] B. Jaspan. Dual-workfactor encrypted key exchange: efficiently preventing password chaining and dictionary attacks. In *Proceedings of the Sixth Annual USENIX Security Conference*, pages 43–50, July 1996.

[11] T. M. A. Lomas, L. Gong, J. H. Saltzer, and R. M. Needham. Reducing risks from poorly chosen keys. In *ACM Operating Systems Review, 23(5)*, pages 14–18. Proceedings of the 12th ACM Symposium on Operating System Principles, Dec 1989.

[12] P. MacKenzie. On the security of the speke password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001.

[13] Philip MacKenzie. The pak suite: Protocols for password-authenticated key exchange. In *IEEE P1363.2*, 2002.