Auditable PAKEs: Approaching Fair Exchange Without a TTP

A W Roscoe, P Y A Ryan

March 13, 2017

Abstract

Roscoe recently showed how HISPs, a class of protocol to allow humans to contribute to the creation of secure authentic channels between them, can be made auditable in the sense that a failed attack on them cannot be disguised as communication failure. In this paper we study the same issue for PAKEs: password authenticated key exchanges. We find that because this second style of protocol relies on long term state, it is harder to make them auditable, and that to do so we have to develop new ideas on how to approximate fair exchange without a TTP.

1 Introduction

In [11], the first author showed how HISPs (Human-Interactive Security Protocols [9]) could be transformed to allow a pair of users to detect when an attacker had deliberately tried to break into their run. This was achieved by using the construct $delay(x, T)$ which allows a user to put the data x beyond anyone's reach for time T, but which the users themselves can eventually open. Some options for implementing *delay* are discussed in [11]. These are based on trusted third parties or alternatively sequential computation. We say that a protocol is *auditable* if the legitimate players can distinguish online guessing attacks from network failures.

The purpose of this paper is to describe how PAKEs (Password Authenticated Key Exchange protocols) may be rendered auditable in a manner analogous to HISPs. Like HISPs, conventional PAKEs are vulnerable to guess and abort style attacks: an attacker attempts a protocol run with a legitimate party and aborts as soon as he knows that his guess at the password is false. By suitable choice of role and timing the attacker can ensure

that the legitimate party does not hold enough information to determine whether she has been interacting with an attacker or has simply been the victim of a network failure.

PAKEs are structurally quite different to HISPs. As a consequence the technique used to render HISPs auditable is not enough to achieve complete auditability in PAKEs. In essence this is because PAKEs, unlike HISPs, have persistent state, meaning we need to protect this state (the password) between runs. To do this we need a more elaborate scheme to achieve auditability, and in fact only achieve a stochastic result: if an attacker makes a guessing attempt then in order to have a chance of gaining information, he has to yield an approximately equal chance of being caught. To achieve this we develop some new stochastic approximations to *fair exchange*, which we hope will find other applications. Nevertheless, the *delay* construct used to make HISPs auditable is once again invaluable.

The rest of this paper is structured as follows. In the next section we outline PAKEs and their structure, and in appendix A we give a number of examples. In Section 2 we examine the important differences between HISPs and PAKEs, and show that copying the auditability transformation used in [11] for HISPs does not, in fact, work because PAKEs have an extra attack vector that is not present in HISPs. To solve this problem we require an exchange of information to be as fair as possible, and therefore in the full version of the paper we describe a new solution to achieving exchange using *delay* that, while not completely fair, can be made as close to *stochastic fairness* as we choose. We then introduce the transformation to the confirmation stage of PAKEs that gives them an auditability property.

We believe the stochastic fair exchange mechanism we introduce here will find uses beyond making PAKEs auditable, and discuss this in Section 5. Appendices give an overview of PAKEs and of key derivation and confirmation mechanisms.

2 Reviewing PAKEs

Password Authenticated Key Establishment protocols (PAKEs) provide a way to establish secure channels in the absence of a PKI (or web of trust or similar), and without requiring empirical channels as is the case for HISPs. Instead, they rely on the parties having a previously shared, low-entropy secret: such as a password or similar. We assume that the parties have been able to previously agree this secret via some authenticated, private channel, e.g. meeting in person over a cocktail. This password will be used

repeatedly in authenticating future session key establishment protocol runs.

The goal of a PAKE is for the parties to establish a secret, mutually authenticated, high-entropy session key that can subsequently be used for secure communication. What PAKEs and HISPs have in common is that authentication typically involves human participation, in one case agreeing, remembering and typing in passwords, and in the other communicating a short seemingly random string between devices. In both cases this means that there is a distinct trade-off between humans' appetite and willingness for cognitive effort, and the amount of security obtained. In both cases the compromise in security is a small but non-negligible probability that an attacker can break authentication. In the case of PAKEs this is that the password may not be strong enough and so might be guessed. A crucial difference is that HISPs are stateless while PAKEs are stateful, in the form of the password.

PAKEs work by authenticating a key establishment protocol using common knowledge of a password between the parties involved, all over insecure (Dolev-Yao) channels. They work by each of the parties P_i computing a value V_i that is a function of the fresh entropy exchanged and the password. Subsequent key confirmation involves exchanges of values based on the V_i allow the various parties to determine whether the other parties with whom they have been running the protocol indeed computed the same values, and by implication share the same password.

Note that, in contrast to AKEs where a PKI is available, it is not possible to explicitly authenticate the messages of the PAKE. If we were to try to it would have to be on the basis of the password and so would provide an attacker with the possibility to launch offline dictionary attacks. Consequently the authentication in a PAKE has to be implicit as a result of the key confirmation steps.

Our proposal is to add features to PAKE protocols, including delay terms and time-outs, so that by the time any man-in-the middle has discovered whether or not the password guess he has made is correct, he has, with high probability, had to deliver confirmation messages (perhaps delayed) to the attacked party, or parties, which will ultimately reveal his presence in the case of an incorrect guess.

For concreteness in what follows we will describe the modifications required in the case of two-player protocols, but the techniques can be readily adapted to multi-party protocols.

The PAKEs known to us all follow the same high-level pattern: some key establishment steps, typically Diffie-Hellman based, with the password s folded into the calculation of the session key in some way. Thus, each party

contributes fresh, high-entropy values and the session key is computed as a function of this fresh entropy and the low-entropy, long-term password. If the messages are not corrupted and both parties used the same password they will compute the same session key.

The key establishment phase is followed by a key confirmation phase that allows the parties to confirm that they have computed the same key, thus providing implicit authentication. Note that the key establishment and key confirmation phases may overlap: the first party to reach a state where it can compute the session can issue key confirmation data along with key establishment data in the next emitted message.

Care must be taken in the design of the entire protocol, key establishment and confirmation phases, to ensure that an attacker, passive or active, cannot launch an offline dictionary attack based on terms derived from runs of the protocol. Thus, the attacker, with polynomially bound computational power, should never derive enough information to confirm or eliminate password guesses with better than negligible probability.

The attacker can of course always launch online guessing attacks: simply interacting with a legitimate party with a guess at the password and observing if this succeeds. This cannot be avoided. The goal therefore is to ensure that this is the optimal attack strategy. Thus, the goal of the design is to ensure that an attacker can test at most one password per attempted run with a legitimate party. The situation is thus analogous to that for HISPs, which seek to ensure that no meaningful combinatorial attack can improve the attacker's chances to more than what they would be with a single completely random attempt at a man-in-the-middle attack.

We identify a difficulty with existing PAKEs analogous to the one discussed for HISPs in [11]: an attacker can disguise attempted guessing attacks as network communication failures. Specifically, he can arrange the attack so that he learns whether his guess is good before the legitimate participants learn of a confirmation failure. When the attacker learns that his attack has failed, i.e. his guess was wrong, he can block subsequent messages. When the attacked party time-out, there is no difference in what it will have seen to a communications failure at the same point in the protocol. Consider the following example based on the SPEKE protocol [6] with key confirmation:

1. $A \rightarrow B : X := hash(s_A)^{2x}$
2. $B \rightarrow A : Y := hash(s_B)^{2y}$

A computes $K_A = K = Y^x = hash(s_B)^{2yx}$ and B computes $K_B = K = X^y = hash(s_A)^{2xy}$.

3. $A \rightarrow B : hash_1(K_A, A, B)$
4. $B \rightarrow A : hash_2(K_B, A, B)$

Here the first two messages allow A and B each to compute a proposed key based on their respective values s_A and s_B of the shared secret. The key feature of SPEKE is that the DH generator is not publicly known but rather is computed by the parties as a function of the password, hence K is a function of the password. The final two messages allow them to confirm that they have computed the same K and hence, with high probability, $s_A = s_B$, implying that they have each been running the protocol with the intended party. $hash$, $hash_1$ and $hash_2$ are different cryptographic hash functions which each offer no information about each other in the sense that knowing the result of applying each to any x conveys no information that allows us to compute the others.

Note that the agent playing the role of Bob receives Message 3, the one which confirms (if true) that the party she is running the protocol shares knowledge of the password $K_A (= K_B)$, before Alice gets the reverse confirmation, provided by Message 4. Thus if "Bob" is actually fraudulent and has made a guess at the password, he will discover if this guess is correct before he has to send the message that will give Alice the same information. If this guess was correct, he will carry on, and Alice will be (incorrectly) convinced of the authentication. If it was not, he can abandon the run at this point and Alice will have no direct evidence of the attacker's presence: all she sees can be explained by a communications failure. This analysis replicates that for HISPs done in [11], except that here life is easier for the attacker because there is no need for the real Alice and Bob *both* to be present. The danger is that the attacker may be able to perform repeated attacks without the participants being sure that they are subject to attack. In the case of PAKEs he can potentially attack both of the owners of a password separately, doubling his chance of obtaining it.

Even if a legitimate party in a PAKE reached the point at which she sees that the computed keys do not match, there may be an innocent explanation: that one of the parties mistyped their password. It would therefore be greatly to our advantage if we could eliminate such mistakes for PAKE users: we hope to introduce such a method in a subsequent paper. In the rest of our present discussion we will assume that a password mismatch in any run is indicative of an attack.

With the strategy of making failed guesses look like communications failures, the attacker can expect to increase his chances of success, by allowing him to make more attempts before the legitimate parties start to suspect

an attack and take evasive action. Such a strategy would be particularly effective where the attacker attacks many nodes in parallel.

In [11], one of us showed how to transform HISPs to prevent this by systematic transformation of the protocols. In other words, we demonstrated how all HISPs could be transformed in essentially the same way to eliminate these attacks.

The fact that known PAKEs essentially all follow the same structure, of a key sharing phase followed by a confirmation phase, suggests firstly that a similar approach might work for PAKEs and that the place to concentrate on is the confirmation phase.

Our objective is thus to find a similar transformation for PAKEs that ensures that the legitimate parties can distinguish with high probability any online guessing attack from a communications failure of the network. The legitimate parties will therefore be able to take remedial action and warn others.

While there are similarities between HIPSs and PAKEs, there are also some significant differences.

- The absence of an out-of-band channel and the need to keep passwords secret make it much harder for Alice and Bob to discriminate between a mistake they may have made and an attack. If Alice simply types in the wrong password, Bob may think he is being attacked.

- A second, and more significant, difference is that PAKEs employ long term shared secret state (the password between a pair of parties) whereas all runs of HISPs are completely independent. It follows that even though an attack on a PAKE may have little chance of succeeding in the sense of getting a faked connection for the attacker on the same session, it may reveal useful things to him about the long term state. This difference means that we cannot simply apply the transformation that Roscoe developed for HISPs to PAKEs in order to render them auditable. While this can prevent an attacker successfully carrying out a one-off attack that simultaneously carries no chance of revealing the attacker and yet gives him a chance of completing a connection, it does not prevent one that can possibly reveal the password to be used in a second session.

  Roscoe's HISP transformation sends data to allow comparison under a *delay* that opens too late to complete the present session: this prevents the attacker getting any *useful* information about whether this attack will succeed or fail. Nor does it assist in future attacks, since no state is

shared between HISP runs. However, the PAKE attacker can afford to wait for this delay to open when he aborts immediately after receiving it: if his password guess was right or wrong in *this* session it will also be right or wrong in the *next*. And even if it is wrong the attacker can eliminate it from his search.

For example, Roscoe's transformation on SPEKE would replace the final two key confirmation messages by the following sequence:

$$\begin{aligned} 3.\quad & A \to B : delay(hash_1(K_A, A, B), T) \\ 4.\quad & B \to A : hash_2(K_B, A, B) \\ & \qquad A : intime(hash_1(K_A, A, B)) \\ 5.\quad & A \to B : hash_1(K_A, A, B) \end{aligned}$$

in which the first message is sent delayed so that B must send his confirmation before he can know whether the delayed confirmation from A was correct or not.[1]

If B is an attacker, he has no chance of using a correct guess at the password to obtain connection in *this* run without giving away his presence (if is guess was wrong) by sending Message 4. The same holds if A is an attacker, because she has had to give away the delayed version of the confirmation for Bob to continue to Message 4, If she sent it honestly, Bob will be able to open the delay eventually and discover if her password guess was correct, and if she sends a wrongly formatted Message 3 this will also give her away.

On the other hand, Bob as attacker can now simply accept Message 3 and then abandon the run. This particular run will fail to complete, but he can wait for $delay(H(K_A, A), T)$ to open and thus find out if his guess s_B at the shared secret was correct. Thus he can either use the password in a future run, or reduce his search space for future attempts. No similar strategy is useful for HISPs.

- A third difference is that the security level can easily be varied dynamically in HISPs: the length of a digest can be increased when there is

[1] There is no need for B to check that the correct value was delayed in Message 3 if he gets the correct Message 5 here. This would have not have been the case if the third and fourth messages of the original protocol were $H(H(K_A))$ and $H(K_B)$, namely nested hashing, because the former can be computed from the latter in the case where $K_A = K_B$ without knowledge of K_B. Since opening delays is potentially expensive, this explains why we used the form of confirmation messages we did.

evidence or suspicion of attack attempts. In PAKEs the password preagreed by Alice and Bob cannot be changed so easily: essentially the only way of reducing the likelihood of an attack in the presence of a known attacker is to restrict or ban use of the protocol.

It is clear that in performing this type of fishing-for-information attack, it is necessary to play the role of the first of the two parties who gets the confirmation message in the original PAKE (the responder B in the case of our SPEKE/key confirmation example above).

In all standard PAKEs, the issue of who sends this type of information first is determined by which role a participant in playing: Alice or Bob. We need to improve on this. Note that the difficulty with the transformed SPEKE above is that the attacker knows exactly at what point he has the relevant information and so he can abort as soon as he reaches this point, even if the information is temporarily inaccessible to him due to a delay wrapper. In the next section we exploit this observation by introducing a stochastic element to the points at which the key information is transmitted.

A slight improvement on the above is to have the parties in effect toss a coin to decide who goes first. (There are cryptographic solutions to doing this fairly, including have each of the two send a delayed random bit[2] with each requiring the other before its own delay opens, and xor-ing them. This is a case where Roscoe's transformation does work, because no state is carried from one toss to the next.) Using this would mean that the attacker would not know at the outset whether he will be able to get Alice to reveal her confirmation value to him first, but he will know that he will have a 50% chance of achieving this and can abort the run if the coin goes against him. We need to improve on this.

3 Two-party Stochastic Exchange

Ideally we would like to achieve *fair exchange* of the confirmation values V_A and V_B: we would like B to get V_A if and only if A gets V_B. We assume that these can be revealed publicly at appropriate junctures without giving away the the key: they may for example be hashed versions of the two sides' keys.

It is known[10] that complete fair exchange, where one party gets what they want if and only if the other party or parties do also, is not possible

[2]In the case where the delay construction is determinisitic, it will be necessary to salt these bits with a random nonce.

without a Trusted Third Party (TTP) or a majority of honest parties. In this context we have only two parties, and the use of a TTP seems inappropriate: it necessitates additional trust assumptions and introduces a bottleneck and single point of failure[3]. In particular this means that in the two party setting we cannot achieve complete fairness. We can however get close to achieving what we term *stochastic fairness*: ensuring that the probability of each party getting what it wants from an aborted exchange is always equal.

What we actually achieve is to bound the difference between these probabilities to be below any positive tolerance. This is done by ensuring that A and B are ignorant of which of a number of messages they send or receive actually communicates V_A or V_B. This is done by a combination of randomisation, blinding and delay: each sends the other a series of messages, knowing that one of the messages in each direction actually communicates V_A or V_B but such that neither of them know which until this process is complete because either these messages or something that enables them is delayed. Here, by "communicates", we mean that if A, say, receives all messages from B up to and including this one, then it will *eventually* (i.e. perhaps after waiting for one or more delays to open) be able to calculate the value V_B without any further input from B (or anyone else).

We will first examine the extent that this can be done without *delay*.

After some preliminary communication, each of A and B creates a number of messages MA_i and MB_i. For the sake of argument we will assume they each create the same natural number k of these, but it is not essential that they create the same number. In each case exactly one of these messages will, given the other's knowledge up to that point (and without the other MA_j or MB_j), reveal the value of (respectively) V_A or V_B or, in the case where it is intended that V_A and V_B are equal, allow the receiving agent to discover this. Each of A and B chooses a blinding key b_A and b_B. Here, blinding means applying some encryption $B(b, M)$ to M which has the properties:

(i) receiving multiple messages encrypted under the same key does not significantly diminish security (as it would with Vernam, for example) and

(ii) $B(b1, B(b2, M)) = B(b2, B(b1, M))$ for all M, $b1$ and $b2$.

(iii) For each b there is a key b^{-1} such that $B(b^{-1}(B(b, x)) = x$ for all x. Thus unblinding fits into the commutative framework implied by (ii).

[3] *Optimistic* fair exchange protocols, e.g. [1] where TTPs are only used in the case of disagreement, counter some of these.

The natural candidate for such a blinding is exponentiation in a suitable Diffie-Hellman type group, i.e. one in which taking discrete logs is deemed intractable. Thus, for example if we work in Z_p^* for a suitable, large prime p:

$$B(b, m) := m^b \ (mod\ p)$$

These two sets of blinded messages are then sent in arbitrary order, either as separate messages or arranged into a list, to the other party. Thus A now has all the messages MB_i created by B, though blinded under b_B, and vice-versa. These messages, because they are blinded, reveal nothing about V_A or V_B, and because the index of the crucial member of this list is kept secret, the recipient does not know which, if unblinded, would reveal these values.

Each of A and B then re-blind the messages, so these become respectively $B(b_A, B(b_B, MB_i))$ and $B(b_B, B(b_A, MA_i))$, and each performs a random permutation on the results. These are once more passed to the other party, who then removes the blinding he or she has performed, which is possible thanks to the assumed commutativity of blinding.

Now A and B respectively hold $B(b_B, MA_i)$ and $B(b_A, MB_i)$, but in each case neither agent knows which of these messages provides the revelation of V_A or V_B. The most crucial step is how these messages are passed over once more: note that if B receives $B(b_B, MA_i)$ he can unblind it to produce MA_i, meaning that exactly one of these messages will reveal V_A to him, and vice-versa. We term this the *reveal* phase. As far as each of the parties is concerned, a random member of each list will do the revealing and neither has any idea which (either of the ones it is sending or the ones it is receiving).[4]

Imagine A and B exchanging the $B(b_B, MA_i)$ and $B(b_A, MB_i)$, in pairs. Typically we will determine who has the job of sending the first of each pair, and who the second. We might, for example, give A the job of sending the first of the first pair, then B the first of the second pair, and so continuing to alternate so that A sends 1 message, then each sends 2 alternately until there is only one left (which will be sent by A if k is even and B if it is odd). For symmetry, assume that the parties "toss a coin" as discussed

[4]It is at this point that we can argue that the use of Vernam encryption would be insecure for blinding: necessarily A has to use the same key in blinding all of the messages MA_i because she does not know which is which when she unblinds them. As soon as B receives one of the $B(k_B, MA_i)$ he will know k_A under Vernam because he has already seen $B(k_A, B(k_B, MA_i))$. Because he has seen *all* the terms of the latter sort he no longer needs to participate in the exchange to learn all the MA_j.

above to decide which goes first. Suppose A is communicating with some impostor I pretending to be B. One of the $B(b_A, MB_i)$ that I holds will probably reveal to A that I is an impostor, but I does not know which one. If I now aborts before he sends any of these he now has only a $\frac{1}{2k}$ chance of being able to check if $V_A = V_B$. To improve his chance of obtaining this information he should send some of his messages. Let's examine what happens if he decides to send his messages until he gets the crucial one from A, he has approximately a 50% chance of "winning" (i.e. checking his to see if $V_A = V_B$ without letting A doing the same.) He is able to do this because he knows immediately when the crucial $B(b_B, MA_i)$ is sent, and can then abort.

This does not apply if he cannot determine which of the messages is the *crucial* one that communicates V_X until beyond a time by which all the exchanges must have taken place. The use of such delays means that I must effectively decide *ab initio* how many of his messages to risk in order to get some of A's: he does not get any useful information about those he receives during the process to modify his strategy. This means that in order to get chance α of discovering if $V_A = V_B$ he has to give away at least an $\alpha - \frac{1}{k}$ of A discovering this (given the pattern of exchange described above).

The obvious way of introducing this delay is to apply it to the $B(b_B, MA_i)$ and $B(b_A, MB_j)$ that are sent in the exchange. However an equivalent and probably more efficient one is for each of A and B to send a single delayed message such as a key which is needed to reveal which of the MX_i is crucial. Thus they might exchange $delay(k_A)$ and $delay(k_B)$, with the MA_i and MB_i being encryptions of k different things, only one of which is V_A or V_B respectively. There would be no need to add extra messages since the delayed keys could be included in an existing one, provided it was no later than the first sent by the respective party in the exchange. The improved efficiency of this approach would be apparent when auditing a failed run since only a single delay term would need to be opened.

This technique set out here is not a fair exchange in the classical sense defined above, because either side can abort the exchange when either might unknowingly have made the reveal to the other. However it approximates what we termed stochastic fairness, meaning that in this case the probabilities of the reveal having been made either way is itself fair. Because of the particular approach we have taken to swapping messages in the reveal phase, the agents will alternately have a $\frac{1}{k}$ advantage in this probability.

We call this technique and the variants introduced below *stochastic exchange*.

There are two important things we have not addressed in the above:

efficiency and preventing fraud by I in what messages he sends. The former, as in the auditable protocols in [11], can be helped by ensuring that $delay(x, T)$ terms only need to be opened during the auditing process as opposed to normal complete protocol runs. The amount of cryptography (particularly with expensive functions) and number of messages sent and received plainly have a major effect on efficiency. It follows that there will be a trade-off between how much work the parties do and how close they come to stochastic fairness.

The latter is achieved by commitment to subsequent sends in advance, on the assumption that the attacker does not want to be caught cheating.

One way of achieving this is as follows: assume that the parties hold V_A and V_B which they wish to exchange.

3.1 Putting the pieces together

We have described the various pieces of plumbing we require and we now put these together in the full protocol. For ease of presentation we opt for the $(1, 2, 2 \cdots, 2)$ pattern of exchanges where both parties generate $k - 1$ fake values. We suppose that A and B have just run a PAKE and so have computed K_A and K_B respectively, and they now wish to establish in a probabilistically fair fashion if $K_A = K_B$. In fact, they will not of course reveal these in the clear but rather values derived in a one way fashion from these, e.g:

$$V_A := Hash(K_A, A, B)$$
$$V_B := Hash(K_B, B, A)$$

3.1.1 Phase 1-Setup

A generates a random seed value s from which the the fake key values $M_{A,i}$ will be generated, an index c_A for the real V_A, a fresh key k_A, and blinding factors b_A and b'_A:

$$s \in_R \mathcal{K},$$
$$k_A \in_R \mathcal{K}$$
$$c_A \in_R \{1, \cdots, k\}$$
$$b_A \in_R \mathcal{B}$$
$$b'_A \in_R \mathcal{B}$$
$$\pi_A \in_R \Pi_k$$

Here $\in_R \mathcal{X}$ indicates drawn at random from the set $\mathcal{X}$ and Π_k denotes the set of permutations of k objects. $\mathcal{K}$ is the space of keys, which we will assume is the same as the blocksize of the cipher $\mathcal{C}$, e.g. 128 bits. $\mathcal{B}$ is the space from which the blinding factors are drawn. The unprimed blinding factors b_X will be used by the parties to blind their own M terms, while the primed factors b'_X will re-blind the M terms received from the other party.

A now computes:

$$M_{A,i} := Hash(i, s), i \in \{1, \cdots, k\}/c_a$$
$$M_{A,c_A} := \{V_A\}_{k_A}$$

A now blinds the $M_{A,i}$ terms using her first blinding factor b_A. Note all the terms are blinded with the same factor in order to allow A to unblind them later without knowing which is which.

B performs the corresponding calculations with $A \leftrightarrow B$.

3.1.2 Phase 2-Commitment

They now exchange Delay terms containing the encryption key, the index c_X (to the term containing their real V_X), their W_X terms and their permutation. At the same time they exchange the blinded M_X terms, but without delay wrappers:

$$A \rightarrow B : Delay(k_A), Delay(c_A), Delay(\pi_A), Delay(s_A), \langle B(b_A, M_{Ai}) \mid i \leftarrow \langle 1, \ldots, k \rangle \rangle$$
$$B \rightarrow A : Delay(k_B), Delay(c_B), Delay(\pi_B), Delay(s_B), \langle B(b_B, M_{Bi}) \mid i \leftarrow \langle 1, \ldots, k \rangle \rangle$$

Note that the Delay terms serve a dual purpose: to conceal the contents for some lower bounded time period and to commit the sender to the contents.

3.1.3 Phase 3-Reblind and Shuffle

Now each re-blinds the M_X terms that they have just received under their own second blinding key b'_X, they permute the resulting terms under their chosen permutation π_X and send the resulting list back to the other:

$$A \rightarrow B : \ \langle B(b'_A, B(b_B, M_{B,\pi_A(i)})) \mid i \leftarrow \langle 1 \ldots k \rangle \rangle$$
$$B \rightarrow A : \ \langle B(b'_B, B(b_A, M_{A,\pi_B(i)})) \mid i \leftarrow \langle 1 \ldots k \rangle \rangle$$

On receipt of these terms, each can strip off their own blinding factor to yield a list of their own M_X terms but blinded and shuffled by the other. Consequently, neither knows which term contains their real V_A. This A now holds the list: $\langle B(b'_B, M_{A,\pi_B(i)}) \mid i \leftarrow \langle 1 \dots k \rangle \rangle$, and similarly for B.

3.1.4 Phase 4-Fair Exchange

Now they are ready to start exchanging these terms progressively according the prescribed alternating schedule. Note that they should not apply any further shuffle at this point, they should send them in the order they received them. This is to ensure that later each is able to identify the critical term once they learnt the other's index value.

$$
\begin{aligned}
&A \rightarrow B : B(b'_B, M_{A,\pi_B(1)}) \\
&B \rightarrow A : B(b'_A, M_{A,\pi_A(1)}), B(b'_A, M_{A,\pi_A(2)}) \\
&A \rightarrow B : B(b'_B, M_{A,\pi_B(2)}), B(b'_B, M_{A,\pi_A(3)}) \\
&. \\
&. \\
&B \rightarrow A : B(b'_A, M_{A,\pi_A(k-1)}), B(b'_A, M_{A,\pi_A(k)})
\end{aligned}
$$

Each can strip off their own blinding factor to reveal the encryptions of the other's M_X terms, one of which should be the encryption of V_X and the others dummies.

3.1.5 Phase 5-Reveal

Assuming that this runs to completion they can now exchange their k_X and c_X terms:

$$
\begin{aligned}
A \rightarrow B &: k_A, c_A, s_A \\
B \rightarrow A &: k_B, c_B, s_B
\end{aligned}
$$

Now that they know the others index value, so with the knowledge of their own permutation they can identify which term contains the real V_X and they can decrypt this with the newly revealed k_X.

3.2 When things go wrong

If either party fails to receive the expected response in a timely fashion it sends no further message and abandons the protocol. If the protocol is

aborted before Phase 4, neither node has received enough to determine if $K_A = K_B$ in the exchange.

If it is aborted in Phase 4, and once the delays have opened, an honest node can audit the messages received so far to determine whether all the messages it has received are coherent, and whether it should hold sufficient information to determine the other's V_X. That is, it extracts the other party's index value from the Delay and from this along with knowledge of the permutation it applied it can determine if the real V_X term should be in the set received to that point. It also knows if its own V_X has been revealed.

If it is aborted after Phase 4, an honest node can audit as above but now certainly has the information to determine V_X, or establish that the other party cheated in the construction of the terms.

The delays only have to be opened in an aborted run because in a completed run the values V_A and V_B will have been exchanged and can be checked against the values committed earlier.

3.3 Auditing

Here we detail the auditing steps that a party X should take. In the event of the protocol aborting, these may have to tale place after the delay wrappers have been opened:

- Check that the other party Y has provided a valid key $k_Y \in \mathcal{K}$, a valid index c_Y in the range $(1, \ldots, k)$, a valid blinding key b'_X and a valid permutation π_Y over k objects and finally a valid seed value $s_Y \in \mathcal{K}$.

- Check, using knowledge of π_x and b'_x, whether Y performed the committed permutation of the re-blinding.

- If the protocol aborts at some point in Phase 4, say after X has received l terms from Y, then X computes $\pi_X(c_Y)$ and if this is less that l then the $\{V_Y\}_{k_Y}$ terms should be in the set X has received. X should decrypt this term and can now establish if $K_X = K_Y$. If $\pi_X(c_Y) > l$ then he has not received the critical term and cannot conclude whether $K_A = K_B$ or not.

- For all the terms received in Phase 4, X should check that they are correctly formed, i.e. of the form $M_{i,Y} = Hash(\pi_X^{-1}(i), s_Y)$, for all $i \leq l$ and $i \neq c_Y$. This check should be performed even if Phase 4 competes, i.e. if $l = k$.

By this means X will be able to tell if Y has followed the protocol correctly. If not an attacker is present. If so and he can tell that $K_A \neq k_B$ then Y has made an incorrect guess so is assumed to be an attacker.

4 Playing with probability

Imagine the following scenario. In a given network there are $2N$ parties who will pair off and try to connect to one another in N protocols runs. The probability of an individual password guess by an intruder being correct is ϵ. Each pair are prepared to make T tries at pairing if they think that communications failings are getting in their way. Therefore, if the pairs use some PAKE to establish their links without using our modified protocols, the intruder can expect to have approximately $NT\epsilon$ successes in breaking into the pairs without any chance of being discovered. (The approximation is because the intruder only needs to break each pair once, and so will stop attacking a given pair before the Tth if it succeeds before then. It is however a good approximation provided the probability of it succeeding in T tries is significantly less than 1, something we would expect in practice.)

If the protocol were now changed to one of our auditable PAKEs, this expectation would decrease to $\frac{N(T-1)\epsilon}{2k}$: the -1 is explained because in order to break in with no chance of being detected a run must be abandoned, so cannot be the last of T. (The discovered password is then used on a subsequent run within the T.) If on each run the intruder is willing to give away a $(1-\epsilon)/k$ chance of being caught (i.e. give away one of his MB_i) this will multiply the expected successes by 3 (rather than only getting only one share in half the runs he will now get one in half the runs and two in the other half), but will have an expectation of being discovered approximately $\frac{N(T-1)(1-\epsilon)}{2k}$ times. This means near certain discovery in many cases. In effect the intruder playing this game would give him 3ϵ successes for each time he is discovered.

With the pattern of exchange we nominated earlier, the ratio of successes gets worse for the attacker as the number of MB_i that he is willing to send increases. For example if he is willing to send 2 or 3 it reduces to $(5/3)\epsilon$ or $(7/5)\epsilon$. (In essence if prepared to give up r, the intruder has a 0.5 probability of giving up $r-1$ for r of Alice's and a 0.5 probability of giving up r for $r+1$, depending on who starts the exchange.

By picking a different sending strategy to the $1, 2, 2, 2, \cdots$ one we could keep all the ratios below 3ϵ and substantially reducing the number of messages sent. For example sending $1, 2, 3, 4, \cdots$ would still result in rations

converging to 1 as the attacker yields more shares, with first term 3ϵ, but use only about $\sqrt{k}$ exchanges. However picking k large enough to keep $\frac{N(T-1)\epsilon}{2k}$ sufficiently small may require nodes to create and blind more MX_i than is ideal.

One can reduce the amount of work the nodes have to do by replacing what we might term the *linear* division strategy (i.e. the probability of being able to deduce V_X grows linearly with the number of MX_i one has seen) by a *quadratic* one. In this there are two specific MX_i that are required to deduce V_X, meaning that the probability of having them both grows quadratically with the number one has, in the initial phase of the exchange. For example X might initially send $\{\{V_X\}_{kx1}\}_{kx2}$ under a delay, making the MX_i a number of keys that include $kx1$ and $kx2$. Of course one could extend this to higher degrees. In the quadratic case with k messages in all, one must have $r > 1$ of them to have any chance of having both the two crucial ones, with the probability then being $\frac{r(r-1)}{k(k-1)}$.

If, for example, we picked $k = 10$ and the MX_i exchanged using the $1, 2, 2, 2, \cdots$ pattern we assumed previously, the attacker could give away one key without fear of giving himself away, and could (if willing to go no further) expect a $\frac{1}{90}$ chance of having his guess checked, and therefore a $\frac{\epsilon}{90}$ chance of obtaining the key. This improves from the $\frac{\epsilon}{20}$ chance using $k = 10$ with the linear approach. The amount of blinding (which is likely to be comparatively the most expensive part of our approach, computationally) is exactly the same in these quadratic and linear examples.

Therefore this quadratic approach reduces the chance our attacker has of breaking any of the N runs without giving away a much bigger chance of being caught.

The designer of any implementation is of course free to choose an approach to devising and exchanging tokens that is believed to be optimal in terms of the perceived threat, communication costs and computation costs. What we have shown in this section is that there are many options other than the linear $1, 2, 2, 2, \cdots$ one adopted earlier.

5 Game playing: other uses of stochastically fair exchange

What we have demonstrated is how to build PAKE protocols in which we can make the rate at which an attacker can gain information about keys without yielding evidence of his existence as small as desired, while the number of guesses at passwords he can check by other means will give away

his existence to the trustworthy parties approximately the same number of times (stochastically).

The main part of this class of protocols is something we did not need at all for HISPs, a way of ensuring a nearly fair game between two players who are each supposed to give each other something, in the absence of a TTP.

Clearly the probabilistic fair exchange mechanisms proposed here could find application in other contexts, for example contract signing. In place of the V_X values we might have signatures and use the probabilistic fair exchange to provide *abuse-freeness*. The idea here is to avoid situations in which one party is able to gain some advantage by being able to prove to a third party that they have the ability to determine whether the protocol will complete successfully or abort.

6 Analysis

Our goal in this paper has been to devise a protocol to prevent an attacker being able to test guesses at the password in a way that will not be detected by the honest participants, or more precisely, in way that they cannot distinguish from network failures. Ideally we would like to ensure that if the attacker can determine that his guess is wrong then the honest party will also know that the keys do not match. We cannot achieve perfect fairness in this sense, at least in the two-party setting without TTP, but we argue that the protocol described here allows us to get arbitrarily close to this. More precisely, we achieve the following property:

Definition 1 *δ Probabilistic Fairness: A protocol Π satisfies δ probabilistic fairness if:*

Let p_X be the probability that X possesses the necessary information to establish whether $V_A = V_B$, possibly after delay terms have been revealed, then at any point in the execution of the protocol, we have:

$$|p_A - p_B| \leq \delta$$

Our basic protocol above satisfies $1/k$ probabilistic fairness.

6.1 Sketch proof of stochastic fairness

Up to the end of Phase 3, neither party has any access to the V terms. They do have access to terms that contain the other party's V terms but these are wrapped first by an encryption under a key that is under a Delay but

more importantly the encryption us shrouded by an unconditionally hiding blinding. In fact, both parties hold k such terms all blinded by the same factor which might seem dangerous, but in fact the security can be shown to reduce that that of RSA .

Thus, if the protocol aborts before the end of Phase 3 the attacker learns nothing about the validity or otherwise of his guess at the password.

If the protocol runs to the end of Phase 4, the fair exchange, then it is clear that both parties will have enough information to establish if $V_A = V_B$, or that the other party provided inconsistent terms. The interesting part is if the protocol aborts at some point during Phase 4.

First we will analyse phase 4 under the assumption that the two parties follow the protocol, later we will discuss how it deals with possible departures from the protocol if one of the parties is adversarial.

By the end of Phase 3, each party will have received back its M_X terms blinded and shuffled by the other party. We need to show that they cannot determine which term contains their real V. The blinding/shuffle construction is essentially the *exponentiation mix* construction of [12]. As long as the terms that are input to the mix are independent in the sense that there is no pair M_{Xi}, M_{Xj} for which the attacker knows y such that $(M_{Xi})^y = M_{Xj}$, then it can be shown that breaking the secrecy of the shuffle is reducible to the DDH assumption. On the other hand, if the attacker can arrange for a pair of inputs such that he knows the discrete log then it is easy for him to trace these through the mix. It is to avoid this that we use the hash function construction for the fake M_X values: the party has to commit to the seed value that is fed into the hash along with the index values to generate the M terms. Assuming that the hash is such that computing pre-images is intractable then he will not be able to produce dependent M_X values consistent with the committed seed s.

This establishes the key property: each party will not know which of the shuffled terms contains their real V and consequently they will not know at what point in phase 4 they pass this term to the other. Equally, they will not know at which point that have received the term containing the other's real V term because, although they know what shuffle they applied they do not know which index the other used. Thus, after the first send B will have a $1/k$ chance that he has V_A (still concealed under a Delay). After the next send of two terms from B, A will have a $2/k$ chance of having V_B, etc. We see that, for this schedule of exchanges, the difference in their advantages is always bounded by $1/k$. As explained earlier, we can improve on this bound by suitable threshold constructions and scheduling, but the arguments will be essentially the same.

Of course, either party may chose to depart from the protocol by for example sending random terms rather than the actual terms, but this will only result in either the other finding that the keys do not match or the deviation will be evident on audit in the event of an abort.

7 Conclusions

In this paper we show that PAKEs, like HIPSs, are vulnerable to an attacker disguising guessing attacks as network failures, We have shown that PAKEs present significantly more challenges than HISPs to making them auditable, and outlined the approach we will take to solve this. The new construction, stochastic fairness, is likely to find application beyond PAKEs, e.g. in fair exchange schemes.

Appendix A: a brief survey of PAKEs

Here we briefly describe a number of representative PAKE protocols. This is purely for illustrative purposes; the techniques we describe below should work for all PAKEs. For simplicity we omit various checks that need to be performed. A comprehensive survey of PAKEs can be found in chapter 40 of the Computer And Information Security Handbook 2nd Edition, Ed J Vacca, Elsevier 2013.

Such protocols come in two phases: key establishment and key confirmation. The first establishes a key based on the shared password, and the second allows each party to confirm that the other knows the password, implying that the key establishment phase was run with the intended party. These phases can generally be chosen independently of each other.

PAKE key establishment

EKE (Encrypted Key Exchange)

The original EKE, [2], is essentially Diffie-Hellman with the DH terms encrypted with a symmetric key s^* derived from the shared password s using a public, deterministic function f, $s^* = f(s)$:

$$A \rightarrow B : \{g^x\}_{s^*}$$
$$B \rightarrow A : \{g^y\}_{s^*}$$

The session key is formed as $K = g^{xy}$.

The original EKE has undergone several fixes to counter flaws, notably the fact that an attacker can eliminate a large number of putative passwords by decrypting the exchanged terms with a guessed password and observing if the resulting plaintext lies in the subgroup.

SPEKE (Simple Password Exponential Key Establishment)

SPEKE, [6], is essentially a D-H protocol but with the difference that the generator is not fixed and public but rather is computed as an agreed function of the shared secret s, for example:

$$h(s) := (H(s))^2 \pmod{p}$$

The squaring guarantees that g lies in the appropriate subgroup assuming that we are assuming a safe prime p where $p = 2q - 1$ with q also prime. The protocol is thus essentially a D-H protocol using the shared secret generator.

$$\begin{aligned} A \to B &: h(s)^x \\ B \to A &: h(s)^y \\ K &= g(s)^{ab} \end{aligned}$$

PKK

A rather elegant protocol, PKK due to MacKenzie and Boyko [3], is in simplified form for illustration:

$$\begin{aligned} A \to B &: X := h(s_A) \cdot g^x, \\ B \to A &: Y := h(s_B) \cdot g^y \end{aligned}$$

Here h denotes a suitable mapping from the password space to the DH group.

A computes: $K_A := (Y/h(s_A))^x$

B computes: $K_B := (X/h(s_B))^y$

J-PAKE

J-PAKE, [5], uses a quite different approach: the so-called juggling of D-H terms. The original J-PAKE involved both parties generating and transmitting two D-H terms. For simplicity of presentation we describe here a lightweight version, [8], that requires just one D-H term from each party but involves a so-called Common Reference String (CRS) construction.

J-PAKE-CRS

Here we assume that there is an agreed element h of the group G with unknown log w.r.t. g (in effect a so-called Common Reference String CRS).

$$A \rightarrow B : g^x, ZKP(x, g)$$
$$B \rightarrow A : g^y, ZKP(y, g)$$

Round two:

$$A \rightarrow B : X := (h \cdot g^y)^{(x.s)}, ZKP(x.s, h \cdot g^y)$$
$$B \rightarrow A : Y := (h \cdot g^x)^{y.s}, ZKP(y.s, h \cdot g^x)$$

A computes: $K_A := (Y/g^{y.x.s})^x$
B computes: $K_B := (X/g^{x.y.s})^y$
Thus, if $s_A = s_B (= s)$ then $K_A = K_B = h^{x.y.s}$
$ZKP(x, y)$ denotes Zero-Knowledge Proofs of knowledge of a discrete log of x w.r.t. the base y.

Appendix B: Key Derivation and Confirmation

Having established a DH shared secret we typically need to derive a suitable session key for a symmetric algorithm. Various approaches have been proposed and we will not go into the details here but we refer the interested reader to, for example the NIST recommendations, [4], and Krawczyk et al. [7]. A typical approach is to derive the key from the DH value is to use a suitable hash function that yields a close to flat distribution over the key space and include parameters associated with the session:

$$SK := Hash_1(K, A, B)$$

Where K is the DH value. Now the parties have to compare their keys, which they might do by, for example, exchanging hashes of the form:

$$A \rightarrow B : Hash_2(1, K_A, A, B)$$
$$B \rightarrow A : Hash_2(2, K_B, A, B)$$

An alternative approach is to segment the derived key into three parts:

$$SK = sk||k_A||k_B$$

Where $||$ denotes concatenation. A and B now exchange the appropriate segments as follows:

$$A \to B : k_A$$
$$B \to A : k_B$$

A and B now check that the received values agree with those they computed internally. Assuming that they do indeed find agreement they can proceed to use sk as the session key. This may require the calculation of a much larger key than normal, possibly requiring key expansion, see [7].

References

[1] N. Asokan, Victor Shoup, and Michael Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, 18(4):593–610, 2000.

[2] 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.

[3] Victor Boyko, Philip MacKenzie, and Sarvar Patel. Provably secure password-authenticated key exchange using diffie-hellman. In *Proceedings of the 19th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'00, pages 156–171, Berlin, Heidelberg, 2000. Springer-Verlag.

[4] Lily Chen. Nist special publication 800-56c recommendation for key derivation through extraction-then-expansion. 2011.

[5] Feng Hao and Peter Y. A. Ryan. Password authenticated key exchange by juggling. In *Proceedings of the 16th International Conference on Security Protocols*, Security'08, pages 159–171, Berlin, Heidelberg, 2011. Springer-Verlag.

[6] David P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, October 1996.

[7] Hugo Krawczyk. Cryptographic extraction and key derivation: The hkdf scheme. In *Proceedings of the 30th Annual Conference on Advances in Cryptology*, CRYPTO'10, pages 631–648, Berlin, Heidelberg, 2010. Springer-Verlag.

[8] Jean Lancrenon, Marjan Skrobot, and Qiang Tang. Two more efficient variants of the j-pake protocol. Cryptology ePrint Archive, Report 2016/379, 2016. http://eprint.iacr.org/2016/379.

[9] L.H. Nguyen and A.W. Roscoe. Authentication protocols based on low-bandwidth unspoofable channels: a comparative survey. *Journal of Computer Security*, 19, 2011.

[10] Henning Pagnia and Felix C Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Citeseer, 1999.

[11] A.W. Roscoe. Detecting failed attacks on human-interactive security protocols. 2016.

[12] Douglas Wikström. A sender verifiable mix-net and a new proof of a shuffle. In *Proceedings of the 11th International Conference on Theory and Application of Cryptology and Information Security*, ASIACRYPT'05, pages 273–292, Berlin, Heidelberg, 2005. Springer-Verlag.