An Offline Dictionary Attack against zkPAKE Protocol

José Becerra[1], Petra Sala[1 2], and Marjan Škrobot[1]

[1] University of Luxembourg
{petra.sala, marjan.skrobot, jose.becerra}@uni.lu,
[2] École Normale Supérieure, Computer Science Department

Abstract. Password Authenticated Key Exchange (PAKE) allows a user to establish a strong cryptographic key with a server, using only knowledge of a pre-shared password. One of the basic security requirements of PAKE is to prevent offline dictionary attacks.
In this paper, we revisit zkPAKE, an *augmented* PAKE that has been recently proposed by Mochetti, Resende, and Aranha (SBSeg 2015). Our work shows that the zkPAKE protocol is prone to offline password guessing attack, even in the presence of an adversary that has only eavesdropping capabilities. Therefore, zkPAKE is insecure and should not be used as a key exchange mechanism.

Keywords: Password Authenticated Key Exchange, zkPAKE, Augmented PAKE, Offline Dictionary Attack, Zero-knowledge Proofs.

1 Introduction

Password Authenticated Key Exchange (PAKE) is a primitive that allows two or more users that start only from a low-entropy shared secret – which is a typical user authentication setting today – to agree on the cryptographically strong session key. Since the introduction of PAKE in 1992, a plethora of protocols trying to achieve secure PAKE has been proposed. However, due to patent issues, PAKEs only recently have started to be considered for wide-scale use (e.g., in Firefox Sync [1] and the Thread network protocol [2]).

From deployment perspective, the biggest advantage of using PAKE compared to a typical key exchange protocol is that it avoids dependence on functional Public Key Infrastructure (PKI). On the downside, the use of low-entropy secret as the primary means of authentication comes with the price: PAKEs are inherently vulnerable to *online* dictionary attacks. To mount this attack, an adversary only has to repeatedly send candidate passwords to the verifying server in order to test for their validity. In practice, this type of attack can be avoided in two-party setting by limiting the number of guesses (i.e., wrong login attempts) which can be made in a given time frame. At the same time, a well-designed PAKE must be resistant against *offline* dictionary attacks. In such attack scenario, the adversary typically operates in two phases: in the first (usually online) phase, the adversary – either by eavesdropping or impersonating a user – tries to collect a function of the password that is being targeted to serve him as the password verifier. Later, in the second (offline) phase, the adversary has to correlate the verifier that has been collected in the first step with offline password guesses to determine the correct password.

In terms of design, PAKEs can follow symmetric or asymmetric approach with respect to the value that is used as an authenticator. For instance, the first PAKE to be proposed, EKE [5], follows symmetric design strategy: Both client and server are required to know their joint password in clear to successfully run the EKE protocol. Such protocols are usually called *balanced* PAKEs. Over time it has been realized that the risk of loosing a large number of passwords in case of a server compromise increases if passwords are kept in clear. Damage inflicted from such loss could be very high, especially today when most people typically use many online services while authenticating with only a few related passwords. One way to mitigate such treat is to use asymmetrically designed PAKE, also known as *augmented* PAKE. This type of PAKE guarantees that the password is not stored on the server side as a plaintext, but, in fact, as an image of the password. Nevertheless, for long it has been argued, from a theoretical perspective, that augmented PAKEs do not add much benefit over balanced PAKEs, since the brute-force attack on a stolen password file (a list containing password hashes) would quickly yield a number of underlying passwords. With the introduction of *sequential memory-hard hash functions* such as Scrypt [17] and Argon2 [7] and use of *salt*, which can be used to significantly slow down password cracking, this may not be the case anymore.

1.1 Our contribution

Recently, Mochetti, Resende and Aranha [15] proposed (without a proof) a simple augmented PAKE called zkPAKE, which they claim is suitable for banking applications, requiring the server to store only the image of a password under a one way function. Their main idea was to use zero-knowledge proof of knowledge (password) to design an efficient PAKE. However, here we present an offline dictionary attack against the zkPAKE protocol. In addition, we show that the same attack works on a slight variant of zkPAKE that has been proposed later in [16]. Our dictionary attack can be carried out in two ways: passively - by eavesdropping on the zkPAKE protocol execution, or actively - by impersonating the server and having the client attempt to log in.

1.2 Previous works

Password Authenticated Key Exchange was introduced by Bellovin and Meritt [5] in 1992. Their EKE protocol was first to show that it is possible to securely bootstrap a low-entropy string into a strong cryptographic key. Few years later, Jablon proposed an alternative - the SPEKE protocol [11]. During next 25 years plenty more PAKE proposals have surfaced [13,14,10]. In parallel, augmented versions of different PAKEs were introduced (e.g. A-EKE[6], B-SPEKE[12]).

As explained above, augmented PAKEs have an additional security property compared to balanced PAKEs: if implemented well, it is considered to be more resistant to server compromise in a sense that clients' passwords are not immediately revealed once the password file is leaked, since the attacker still has to perform password cracking.

Security of early PAKE proposals was argued only informally by showing that protocol can withstand all known attacks. Starting from 2000, formal models of security for PAKE appeared in [4] and [8]. More specifically, following game-based approach Bellare, Pointcheval and Rogaway have argued in [4] that a provably secure PAKE protocol must provide indistinguishability of the session key and satisfy authentication property. The Real-or-Random (RoR) variant of their model from [3], along with the Universally Composable PAKE model from [9] are considered to be state-of-the-art models that rigorously capture PAKE security requirements.

Since in this paper we exclusively deal with an offline dictionary attack on zkPAKE, we keep the discussion here short and refer readers to Pointcheval's survey [18] for more detailed overview of PAKE research field.

1.3 Organization

The rest of the paper is organized as follows. Section 2 describes the zkPAKE protocol and its variant. In Section 3, we present an offline dictionary attack against the zkPAKE protocol. Finally, we conclude the paper in Section 4.

2 The zkPAKE protocol and its variant

In this section, we review the zkPAKE protocol. We will start with the variant of zkPAKE from [16] whose description is presented in Figure 1, and then point out the differences with the original design from [15]. The reason for this order of presentation is because the variant of zkPAKE that is proposed later is slightly more elaborate than the original zkPAKE, so we want to show that zkPAKE does not stand against our attack even with proposed modification.

2.1 Protocol description

zkPAKE as described in [16] is a two-party augmented PAKE protocol meant to provide authenticated key exchange between a server S and a client C.

Initialization phase. The protocol starts with an enrollment phase, which is executed for every client only once. In this phase, a client and a server (e.g., bank) share a secret value of low entropy that can be remembered by the client. More specifically, in case of zkPAKE, the client must remember the password π, while the server only stores an image of the password R. Before the server computes the corresponding image R, public parameters must be chosen and agreed on: 1) a finite cyclic group $\mathbb{G}$ of prime order q and a random generator g of the group $\mathbb{G}$; 2) Hash functions H_1 and H_2 whose outputs are k-bit strings, where k is the security parameter representing the length of session keys.

Protocol execution. Once the enrollment phase is executed and the public parameters are established, the zkPAKE protocol (see Figure 1) will run in three communication rounds as follows:

Initialization

Public: $\mathbb{G},\ g,\ q; \quad H_1, H_2 : \{0,1\}^* \to \{0,1\}^k;$

	Server S		Client C
Secret:	$R := g^{H_1(\pi)}$		π
	$n \leftarrow \mathbb{Z}_q$		
	$N := g^n$	$\xrightarrow{N}$	
			$r := H_1(\pi)$
			$v \leftarrow \mathbb{Z}_q$
			$t := N^v$
			$c := H_1(g, g^r, t, N)$
			$u := v - H_1(c)r \ mod \ q$
		$\xleftarrow{u, H_1(c)}$	$sk_c := H_2(c)$
	$t' := g^{un} R^{nH_1(c)}$		
	$c' := H_1(g, R, t', N)$		
	abort if $H_1(c') \neq H_1(c)$		
	$sk_s := H_2(c')$	$\xrightarrow{H_1(sk_s)}$	
			abort if $H_1(sk_s) \neq H_1(sk_c)$

Fig. 1: The zkPAKE protocol.

1. First, the server S chooses a random value n from $\mathbb{Z}_q$, computes N that is supposed to act both as a nonce and Diffie-Hellman value, and sends it to the client C.
2. Now, upon receiving the nonce N, the client C inputs his password, computes the hash of the password - r, chooses a random element v from $\mathbb{Z}_q$, and computes $t := N^v$. Then, C computes $c := H_1(g, g^r, t, N)$ and obtains $u := v - H_1(c)r$ that should lie in $\mathbb{Z}_q$. Next, C computes the session key $sk_c := H_2(c)$ and sends u and $H_1(c)$ to the S.
3. Upon receiving $H_1(c)$ and u, S recovers t' by computing $g^{un}R^{nH_1(c)}$. Then, S calculates $c' := H_1(g, R, t', N)$. Next, S checks if $H_1(c')$ echoes $H_1(c)$. If it does, S computes the session key $sk_s := H_2(c')$ and sends $H_1(sk_s)$ to C. Otherwise, he aborts the protocol.

4. Similarly, upon receiving $H_1(sk_s)$, C checks if $H_1(sk_s)$ and $H_1(sk_c)$ match. If values are equal, C saves computed session key sk_c and terminates.

As we said before, the authors of zkPAKE have presented two variants of it. The original proposal from [15] differs from the follow-up version in two places: Nonce N is left underspecified, and value t on the client side is computed without involving received nonce. This difference also affects the computation of t' from the server side. In more details, the original zkPAKE protocol runs as follows:

1. The server sends his nonce N to the client C.
2. The client calculates the hash of his password r, chooses a random parameter $v \leftarrow \mathbb{Z}_q$, and computes $t := g^v$. Then, C computes $c := H_1(g, g^r, t, N)$ and obtains $u := v - H_1(c)r$ in $\mathbb{Z}_q$. Next, C computes the session key $sk_c := H_2(c)$ and sends u and $H_1(c)$ to the S.
3. Upon receiving $H_1(c)$ and u, S recovers t' by computing $g^u R^{H_1(c)}$. Then, S calculates $c' := H_1(g, R, t', N)$. Next, S checks if $H_1(c')$ echoes $H_1(c)$. If it does, S computes the session key $sk_s := H_2(c')$ and sends $H_1(sk_s)$ to C. Otherwise, he aborts the protocol.
4. Finally, upon receiving $H_1(sk_s)$, C checks if $H_1(sk_s)$ echoes $H_1(sk_c)$. If values are equal, C saves computed session key sk_c and terminates.

3 Offline dictionary attack on zkPAKE

In the next section, we will show how both variants of the zkPAKE protocol are vulnerable to an offline dictionary attack. Our attack exploits the fact that r, which is a hash of clients password, is of low entropy.

Let the enrollment phase be established and let an attacker $\mathcal{A}$ be allowed only to eavesdrop on the communication between two honest parties. The attack on the version of zkPAKE protocol presented on Figure 1 proceeds as follows:

Step 1. The execution of the protocol starts and S sends his first message, N. The attacker $\mathcal{A}$ sees the message and stores it in his memory.

Step 2. C does all the computations demanded by the protocol and sends u and $H_1(c)$ in the second transmission to S. $\mathcal{A}$ observes the second message and obtains u and $H_1(c)$.

Step 3. The adversary that now holds N, u and $H_1(c)$ from the first two message rounds may go offline to perform a dictionary attack. His goal is to compute a candidate c'_i and then use stored $H_1(c)$ as a verifier. The adversary will compute c'_i by hashing $H_1(g, g^{r_i}, t'_i, N)$. Two intermediate inputs to hash function are obtained by first choosing a candidate password π_i, and then computing the corresponding r_i and t'_i. Note that the adversary can easily compute $t'_i = N^{v_i}$, since $v_i := u + H_1(c)r_i$. Finally, the adversary checks if his guess $H_1(c'_i)$ echoes $H_1(c)$.

Step 4. The adversary repeats step 3 until he guesses the correct password.

As for the original zkPAKE protocol, the same attack works in a very similar way: steps 1,2, and 4 are the same while in step 3 we need to make a minor change:

Step 3a. The adversary that now holds N, u and $H_1(c)$ from the first two message rounds may go offline to perform a dictionary attack. Same as above, the adversary aims to obtain candidate c_i' by computing a hash $H_1(g, g^{r_i}, t_i', N)$. Here the only difference is that $t_i' = g^{v_i}$, while formula for computing v_i stays the same.

Note that one can mount a similar dictionary attack by impersonating a server. In this case, the only difference with the eavesdropping attack described above is that the value of the nonce N is picked by the attacker. Such knowledge, however, does not additionally help the adversary in our attack. Once the adversary receives clients reply, he can continue with steps 3 and 4 from the eavesdropping attack.

4 Conclusion

In this paper, we showed that both versions of the zkPAKE protocol [15,16] are vulnerable to offline dictionary attacks. To make matters worse, the adversary in case of zkPAKE only needs eavesdropping capabilities to mount the attack.

Taking a wider view on zkPAKE, the problem with its design lies in a fact that variable r, which is of low-entropy, is used as a mask for the secret value v. In contrast, in a typical zero-knowledge proof of knowledge, which was used as an inspiration for zkPAKE design, such value is of high entropy. By showing this vulnerability, we hope that in future protocol designers will be more careful in claiming the security of proposed protocols, especially when a proof of security does not back those claims.

References

1. Firefox Sync (2015), `https://www.mozilla.org/en-US/firefox/sync/`
2. Thread Protocol (2015), `http://threadgroup.org/`
3. Abdalla, M., Fouque, P., Pointcheval, D.: Password-Based Authenticated Key Exchange in the Three-Party Setting. In: Vaudenay, S. (ed.) Public-Key Cryptography – PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer (2005)
4. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated Key Exchange Secure Against Dictionary Attacks. In: Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer (2000)
5. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: 1992 IEEE Symposium on Research in Security and Privacy, SP 1992. pp. 72–84 (1992)

6. Bellovin, S.M., Merritt, M.: Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise. In: Denning, D.E., Pyle, R., Ganesan, R., Sandhu, R.S., Ashby, V. (eds.) CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993. pp. 244–250. ACM (1993)
7. Biryukov, A., Dinu, D., Khovratovich, D.: Argon2: New Generation of Memory-Hard Functions for Password Hashing and Other Applications. In: IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016. pp. 292–302. IEEE (2016)
8. Boyko, V., MacKenzie, P.D., Patel, S.: Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman. In: Preneel, B. (ed.) Advances in Cryptology – EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer (2000)
9. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.D.: Universally Composable Password-Based Key Exchange. In: Cramer, R. (ed.) Advances in Cryptology – EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer (2005)
10. Hao, F., Ryan, P.: J-PAKE: Authenticated Key Exchange without PKI. Transactions on Computational Science 11, 192–206 (2010)
11. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. ACM SIGCOMM Computer Communication Review 26(5), 5–26 (1996)
12. Jablon, D.P.: Extended Password Key Exchange Protocols Immune to Dictionary Attacks. In: 6th Workshop on Enabling Technologies (WET-ICE '97), Infrastructure for Collaborative Enterprises, 18-20 June 1997, MIT, Cambridge, MA, USA, Proceedings. pp. 248–255. IEEE Computer Society (1997)
13. Katz, J., Ostrovsky, R., Yung, M.: Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In: Pfitzmann, B. (ed.) Advances in Cryptology – EUROCRYPT 2001. LNCS, vol. 2045, pp. 475–494. Springer (2001)
14. MacKenzie, P.: The PAK Suite: Protocols for Password-Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002)
15. Mochetti, K., Resende, A., Aranha, D.: zkPAKE: A Simple Augmented PAKE Protocol. In Brazilian Symposium on Information and Computational Systems Security (SBSeg) (2015)
16. Mochetti, K., Resende, A., Aranha, D.: zkPAKE: A Simple Augmented PAKE Protocol (2015), `http://www2.ic.uff.br/~kmochetti/files/abs01.pdf`
17. Percival, C.: Stronger Key Derivation via Sequential Memory-hard Functions. Self-published pp. 1–16 (2009)
18. Pointcheval, D.: Password-Based Authenticated Key Exchange. In: Fischlin, M., Buchmann, J.A., Manulis, M. (eds.) Public Key Cryptography - PKC 2012. LNCS, vol. 7293, pp. 390–397. Springer (2012)