An Offline Dictionary Attack against zkPAKE Protocol

José Becerra[1], Peter Y. A. Ryan[1], Petra Šala[1,2], and Marjan Škrobot[1]

[1] University of Luxembourg
6, Avenue de la Fonte
L-4364 Esch-sur-Alzette, Luxembourg
{name.lastname}@uni.lu,
[2] École Normale Supérieure, Computer Science Department
45 rue d'Ulm, 75230, Paris, France
{name.lastname}@ens.fr

Abstract. Password Authenticated Key Exchange (PAKE) allows a user to establish a secure 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. Results of performance evaluation show that our attack is practical and efficient.Therefore, zkPAKE is insecure and should not be used as a password-authenticated key exchange mechanism.

Keywords: Password Authenticated Key Exchange, Augmented PAKE, zkPAKE, 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, only recently have PAKEs begun to be considered for a wide-scale use: SRP [28] has been used in password manager called 1Password [2], J-PAKE of Hao and Ryan [14] was used in Firefox Sync [12], while *Elliptic Curve* (EC) version of the same protocol (EC-J-PAKE [11]) has been used to enable authentication and authorization for network access for *Internet-of-Things* (IoT) devices under the Thread network protocol [13].

From a deployment perspective, the most significant 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, all an adversary needs to do is repeatedly send candidate passwords to the verifying server to test for their validity. In practice, this type of attack can be relatively easily avoided in a two-party setting by limiting the number of guesses (i.e., wrong login attempts) that 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.

Concerning of design, PAKEs can follow a symmetric or asymmetric approach concerning the value that is used as an authenticator. For instance, the first PAKE to be proposed, EKE [6], 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 losing a large number of passwords in case of a server compromise increases if passwords are kept in the 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[3]. This type of PAKE guarantees that the password is not stored on the server side as 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 [25] and Argon2 [8] and use of *salt*, which can be used to slow down password cracking significantly, this may not be the case anymore.

1.1 Our contribution

Recently, Mochetti, Resende and Aranha [23] proposed (without exhibiting a security 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 [24]. We also provide a prototype and share the benchmarks of

[3] For the latest results on augmented PAKE check Jarecki et al. [20].

the attack to demonstrate its feasibility. 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 [6] in 1992. Their EKE protocol was first to show that it is possible to bootstrap a low-entropy string into a strong cryptographic key. A few years later, Jablon proposed an alternative - the SPEKE protocol [18]. Over the next 25 years, plenty more PAKE proposals have surfaced [21,22,4,14]. In parallel, augmented versions of different PAKEs were introduced (e.g. A-EKE[7], B-SPEKE[19]). 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. Finally, a number of them have been standardized in IEEE [16], IETF [15] and ISO [17].

Security of early PAKE proposals was argued only informally by showing that a protocol can withstand all known attacks. Starting from 2000, the two formal models of security for PAKE appeared in [5] and [9]. More specifically, following a game-based approach Bellare, Pointcheval and Rogaway have argued in [5] that a provably secure PAKE protocol must provide the indistinguishability of the session key and satisfy the authentication property. The Real-or-Random (RoR) variant of their model from [3], along with the Universally Composable PAKE model from [10] are considered to be state-of-the-art models that rigorously capture PAKE security requirements.

Since we exclusively deal with an offline dictionary attack on zkPAKE, in this paper, we keep the discussion here short and refer readers to Pointcheval's survey [26] for a 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 both variants of the zkPAKE protocol. Finally, we conclude the paper in Section 4.

2 The zkPAKE protocol

In this section, we review the zkPAKE protocol. We will start with the variant of zkPAKE from [24] whose description is presented in Figure 1, and then point out the differences with the original design from [23]. 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 modifications.

2.1 Protocol description

zkPAKE, as described in [24], 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:

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; it 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 [23] 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.

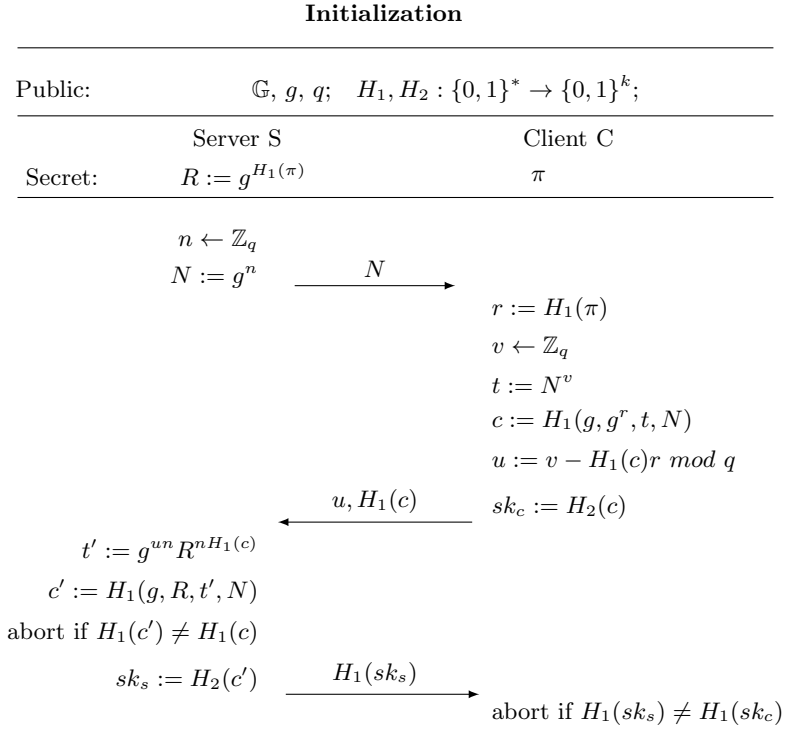

Fig. 1. The zkPAKE protocol.

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.

3.1 Attack description

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 in 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' and then use stored $H_1(c)$ as a verifier. The adversary will compute c' by hashing $H_1(g, g^r, t', N)$. Two intermediate inputs to the hash function are obtained by first choosing a candidate password π, and then computing the corresponding r and t'. Note that the adversary can easily compute $t' = N^v$, since $v := u + H_1(c)r$. Finally, the adversary checks if his guess $H_1(c')$ 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 the 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 the attacker picks the value of the nonce N. 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.

3.2 Attack implementation

We implemented a prototype[4] in Python 3 to simulate the attack described above. Our simulation consists of two steps: in the first step, a password is randomly chosen from one of three fixed dictionaries that vary in size and the zkPAKE protocol is executed between two honest parties. Then, in the second step (see Algorithm 1), the adversary is given access to honestly generated values as described in Section 3.1. With this information in hand, the adversary can easily perform an offline dictionary attack against chosen password.

[4] Available under GPL v3 at https://github.com/PetraSala/zkPAKE-attack.

Input: Values N, u, $H(c)$ and a dictionary of passwords P
Output: pw', K
for *each pw' in dictionary* P **do**
  $c' = hash.sha256(pw')$
  $r = c' \bmod q$
  $v = (u + H(c) * r) \bmod q$
  $t = N^v \bmod p$
  $R = g^r \bmod p$
  $H(c') = hash.sha256(g, g^r, t, N)$
  if $H(c) == H(c')$ **then**
    $K = HKDF(c')$
    Return $(pw', H(c'), K)$
  end
end

Algorithm 1: Offline search algorithm

We performed a set of experiments, using a 224-bit subgroup of a 2048-bit finite field Diffie-Hellman group[5], to determine the time it takes to complete an offline dictionary attack depending on the size of a selected dictionary. Each set of experiments involved mounting the attack by enumerating dictionaries that contain 1000, 10000, and 100000 random password elements. Each experiment was performed 50 times.

Results. The times it took the adversarial algorithm described above to find a matching password for each given dictionary are summarized in Table 1. Our

Dictionary size	**Average Time until the Correct Password is found** (ms)	**Std Dev**
1000	3694	1898
10000	27322	17461
100000	244540	178465

Table 1. Results for different dictionary sizes.

results demonstrate that there is a linear relationship between the size of the dictionary and the average time to find a matching password, and shows that an attack is feasible for any adversary with even a small computational power[6]. As expected, the total time for cracking a 100000 password-size dictionary is less than 5 min, and thus we conclude that the attack would be feasible for

[5] Selected group parameters, which are originally coming from the standard NIST cryptographic toolbox, are specified in Appendix A.

[6] In all cases the experiments were run under Windows 10 on a 2.8GHz PC with 8GB of memory.

dictionaries with significantly more elements. We also note that there are more powerful tools to create more efficient dictionaries, such as HashCat [27] or John the Ripper [1], which would make the offline search more effective.

4 Conclusion

In this paper, we showed that both versions of the zkPAKE protocol [23,24] are vulnerable to offline dictionary attacks. To make matters worse, the adversary in case of zkPAKE only needs eavesdropping capabilities to mount the attack.

By 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. Since zkPAKE protocol core design is flawed beyond repair and there already exist many mature PAKE alternatives, we do not pursue further study to improve on the zkPAKE protocol.

References

1. John the ripper password cracker. https://www.openwall.com/john/ (25-02-2019)
2. 1Password Security Design. https://1password.com/files/1Password%20for%20Teams%20White%20Paper.pdf (27-02-2018)
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. Abdalla, M., Pointcheval, D.: Simple Password-Based Encrypted Key Exchange Protocols. In: Menezes, A. (ed.) Topics in Cryptology - CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer (2005)
5. 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)
6. Bellovin, S.M., Merritt, M.: Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In: IEEE Symposium on Research in Security and Privacy. pp. 72–84 (1992)
7. 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. pp. 244–250. ACM (1993)
8. 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. pp. 292–302. IEEE (2016)
9. 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)

10. 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)
11. Cragie, R., Hao, F.: Elliptic Curve J-PAKE Cipher Suites for Transport Layer Security (tls) (2016), https://tools.ietf.org/html/draft-cragie-tls-ecjpake-01
12. Foundation, T.M.: Firefox Sync. https://www.mozilla.org/en-US/firefox/sync/ (28-02-2018)
13. Group, T.: Thread Protocol. http://threadgroup.org/ (06-04-2017)
14. Hao, F., Ryan, P.: J-PAKE: Authenticated Key Exchange without PKI. Transactions on Computational Science **11**, 192–206 (2010)
15. Harkins, D.: Dragonfly Key Exchange. RFC 7664, RFC Editor (November 2015)
16. Standard Specifications for Password-based Public Key Cryptographic Techniques. Standard, IEEE Standards Association, Piscataway, NJ, USA (2002)
17. ISO/IEC 11770-4:2006/cor 1:2009, Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets. Standard, International Organization for Standardization, Genève, Switzerland (2009)
18. Jablon, D.P.: Strong Password-Only Authenticated Key Exchange. ACM SIGCOMM Computer Communication Review **26**(5), 5–26 (1996)
19. 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. pp. 248–255. IEEE Computer Society (1997)
20. Jarecki, S., Krawczyk, H., Xu, J.: OPAQUE: An Asymmetric PAKE Protocol Secure Against Pre-Computation Attacks. In: Dunkelman, O. (ed.) Advances in Cryptology – EUROCRYPT 2018. LNCS, Springer (2018)
21. 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)
22. MacKenzie, P.: The PAK Suite: Protocols for Password Authenticated Key Exchange. DIMACS Technical Report 2002-46 (2002)
23. Mochetti, K., Resende, A., Aranha, D.: zkPAKE: A Simple Augmented PAKE Protocol. In Brazilian Symposium on Information and Computational Systems Security (SBSeg) (2015)
24. Mochetti, K., Resende, A., Aranha, D.: zkPAKE: A Simple Augmented PAKE Protocol. http://www2.ic.uff.br/~kmochetti/files/abs01.pdf (2015)
25. Percival, C.: Stronger Key Derivation via Sequential Memory-hard Functions. Self-published pp. 1–16 (2009)
26. 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)
27. Team Hashcat: hashcat - advanced password recovery. https://hashcat.net/hashcat/ (25-02-2019)
28. Wu, T.D.: The Secure Remote Password Protocol. In: Proceedings of the Network and Distributed System Security Symposium, NDSS 1998. The Internet Society (1998)

A Appendix

The group parameters are taken from the NIST crytpographic toolbox using 2048 modulus, and are shown in Table 2.

Parameter	Value (Base 16)
Prime Modulus	AD107E1E 9123A9D0 D660FAA7 9559C51F A20D64E5 683B9FD1 B54B1597 B61D0A75 E6FA141D F95A56DB AF9A3C40 7BA1DF15 EB3D688A 309C180E 1DE6B85A 1274A0A6 6D3F8152 AD6AC212 9037C9ED EFDA4DF8 D91E8FEF 55B7394B 7AD5B7D0 B6C12207 C9F98D11 ED34DBF6 C6BA0B2C 8BBC27BE 6A00E0A0 B9C49708 B3BF8A31 70918836 81286130 BC8985DB 1602E714 415D9330 278273C7 DE31EFDC 7310F712 1FD5A074 15987D9A DC0A486D CDF93ACC 44328387 315D75E1 98C641A4 80CD86A1 B9E587E8 BE60E69C C928B2B9 C52172E4 13042E9B 23F10B0E 16E79763 C9B53DCF 4BA80A29 E3FB73C1 6B8E75B9 7EF363E2 FFA31F71 CF9DE538 4E71B81C 0AC4DFFE 0C10E64F
Generator	AC4032EF 4F2D9AE3 9DF30B5C 8FFDAC50 6CDEBE7B 89998CAF 74866A08 CFE4FFE3 A6824A4E 10B9A6F0 DD921F01 A70C4AFA 00C29F52 C57DB17C 620A8652 BE5E9001 A8D66AD7 C1766910 1999024A F4D02727 5AC1348B B8A762D0 521BC98A E2471504 22EA1ED4 09939D54 DA7460CD B5F6C6B2 50717CBE F180EB34 118E98D1 19529A45 D6F83456 6E3025E3 16A330EF BB77A86F 0C1AB15B 051AE3D4 28C8F8AC B70A8137 150B8EEB 10E183ED D19963DD D9E263E4 770589EF 6AA21E7F 5F2FF381 B539CCE3 409D13CD 566AFBB4 8D6C0191 81E1BCFE 94B30269 EDFE72FE 9B6AA4BD 7B5A0F1C 71CFFF4C 19C418E1 F6EC0179 81BC087F 2A7065B3 84B890D3 191F2BFA
Subgroup order	801C0D34 C58D93FE 99717710 1F80535A 4738CEBC BF389A99 B36371EB

Table 2. Group parameters