An overview of symmetric fuzzy PAKE protocols

Johannes Ottenhues[1]

Abstract: Fuzzy password authenticated key exchange (fuzzy PAKE) protocols enable two parties to securely exchange a session-key for further communication. The parties only need to share a low entropy password. The passwords do not even need to be identical, but can contain some errors. This may be due to typos, or because the passwords were created from noisy biometric readings. In this paper we provide an overview and comparison of existing fuzzy PAKE protocols. Furthermore, we analyze certain security properties of these protocols and argue that the protocols can be expected to be slightly more secure in practice than can be inferred from their theoretical guarantees.

Keywords: Cryptographic protocols; Fuzzy password authenticated key exchange; Biometrics

1 Introduction

The notion of password authenticated key exchange (PAKE) has become increasingly popular, due to its strong security guarantees and good password protection. In the PAKE setting two parties aim to establish a fresh session-key in an adversarial environment, i. e. where the attacker has full control over the communication channel. The only advantage that the parties have, is a shared (potentially low entropy) password. However, deriving the session-key directly from the password is not possible, as it would compromise forward security and expose the system to offline attacks. This is not acceptable due to the potentially low entropy of the password. There have been plenty of proposals for PAKE protocols (e. g. [BPR00, HL18, Ja96]), recently also with a focus on post-quantum security [Be23, PZ23]. There are many different applications for PAKE protocols, for example PACE is used in machine readable passports [Be12, Int21]. Another example is the use of PAKE protocols for improving security in wireless networks [Cr22]. In 2019 and 2020 the IETF ran a selection process for PAKE protocols in which CPace [HL18] was chosen as a winner.

Depending on the use case, PAKE can be seen as having two disadvantages. In a typical online authentication setting, where a client wants to authenticate to a server, the server would need to store the password in clear text. This problem is addressed by *asymmetric* PAKE (aPAKE). Another shortcoming of PAKE is in settings where exactly matching passwords cannot be guaranteed. One such setting is when typos in the password are frequent, e. g. because the passwords are long and complicated, or they are typed on a smartphone. The necessity to frequently repeat the protocol due to typos decreases usability and may prevent adoption altogether. Another setting were slightly mismatching passwords are not

[1] University of St. Gallen, School of Computer Science, johannes.ottenhues@posteo.org

only frequent but almost certain is the use of biometric traits as credentials. Such traits can be iris scans [Da09], fingerprints [Ja99], face images [SKP15] or even vein patterns [Wi10]. However, biometric readings are inherently noisy, so that two biometric readings of the same person are almost always slightly different. This immediately rules out their use in PAKE protocols.

In order to solve the problem of not exactly matching but close passwords or credentials, the notion of fuzzy PAKE has been introduced by [Du18]. A fuzzy PAKE protocol allows two parties to agree on a session-key, if the credentials they hold are similar enough. Similarity is usually quantified by requiring that the distance of the passwords must be below a certain threshold. The exact notion of distance depends on the use case and the concrete fuzzy PAKE protocol.

In this paper we explain and compare the three existing (symmetric) fuzzy PAKE protocols [Du18, Bo23] that have a security analysis in the framework of universal composability (UC) [Ca01] in Sect. 3[2]. The UC framework requires a very rigorous specification of the security properties of a protocol and also provides strong composability guarantees for the protocols that satisfy their specification[3]. Composability is important, as fuzzy PAKE protocols are almost never run for their own sake, but are used alongside other protocols, most often to bootstrap secure communication channels. If an attacker correctly guesses a password close to the one of the attacked party, the specification of fuzzy PAKE from [Du18, Bo23] allows the attacker to learn additional information about the password and to choose the session-key. This is reasonable, because the password is the only information that protects the key exchange and if the attacker knows it, one cannot expect to retain much security. Nevertheless, in Sect. 4 we analyze whether in practice these protocols can be expected to fare even better than their security specification. Finally, in Sect. 5 we highlight some open problems and challenges in the area of fuzzy PAKE protocols.

2 Preliminaries

The full preliminaries can be found in Appendix A.

3 Comparison of fuzzy PAKE protocols

In this section we give an overview of three symmetric fuzzy PAKE protocols. To facilitate better understanding, we simplified the protocols and omitted certain details (such as session-identifiers). Some of these details are important for the security of the protocols. Therefore, before building on top of these protocols or implementing them, one should consult the original references ([Du18], [Bo23]).

[2] We restrict our attention to *symmetric* fuzzy PAKE, thus, excluding the *asymmetric* protocols in [Er20].

[3] In the UC framework this specification is an ideal functionality. By saying that a protocol π satisfies its specification, we mean that π *UC-realizes* this ideal functionality.

3.1 Fuzzy PAKE based on garbled circuits

The idea underlying the fPAKE-GC protocol is to use a garbled circuit that takes both passwords as input and checks if they are close enough. The circuit has a one bit output that indicates the result of the comparison. However, this simple approach is, first, not yet sufficient to exchange a key and, second, only secure against a semi-honest garbler[4]. Instead, [Du18] use a variant of the *dual execution* technique of [MF06] and [HKE12] and additionally let the parties not decode the output of the garbled circuit, but instead let them use the output-wire label directly as part of the session-key.

Let us look at the protocol in a bit more detail. A simplified version of the protocol is shown in Fig. 1. First, each party garbles a circuit that takes the passwords as input and has a one bit output to indicate whether the passwords are close enough. In the garbled circuit this one bit output is represented by one of two random bit-strings (so called wire labels), where one bit-string corresponds to the output 1 (i. e. "close") and one bit-string corresponds to the output 0 (i. e. "not-close"). In Fig. 1 for example l_A^{close} refers to the output wire label that corresponds to the output 1 (i. e. close passwords) in the garbled circuit C_A created by A.

In the next step each party evaluates the garbled circuit of the other party and obtains the corresponding output wire label (l_A, resp. l_B). Since the protocol is symmetric, wlog. we now view the protocol from the perspective of party A. Evaluating B's garbled circuit requires A to obtain the input wire labels of B's garbled circuit that correspond to A's password. For security it is important that B does not learn A's password and that A only learns one (and not two) wire labels per input bit. Therefore, the wire labels are transmitted via oblivious transfer, which has exactly the required properties. In the actual protocol in [Du18] both parties first perform the oblivious transfer and then exchange the garbled circuits. From evaluating B's garbled circuit, A obtains the output wire label l_B, where $l_B \in \{l_B^{\text{close}}, l_B^{\text{not-close}}\}$. Finally, A XORs their own l_A^{close} label with l_B to obtain the session-key.

If the passwords are close, then $l_B = l_B^{\text{close}}$ and $l_A = l_A^{\text{close}}$ and, therefore, both parties will get $l_B^{\text{close}} \oplus l_A^{\text{close}}$ as session-key. If the passwords do not match, then both parties obtain different session-keys, which is exactly what is required by the specification. Even if B acts maliciously, they cannot obtain the same session-key as A without knowing a password that is close to the one of A. Intuitively this is because when the passwords are not close, B will only obtain $l_A^{\text{not-close}}$ and learn nothing about l_A^{close}. Because l_A^{close} is unknown and uniformly random for B, the same holds for A's session-key $l_B \oplus l_A^{\text{close}}$.

Note that in [Du18] the authors apply the *split transformation* from [Ba05] to this protocol, in part to get the authenticated channels that are necessary for the OT protocol. This is very similar to the *split authenticated channels* used in [Bo23] (cf. Sect. 3.3).

[4] This means that the approach is not secure if the party who garbles the circuit tries to cheat.

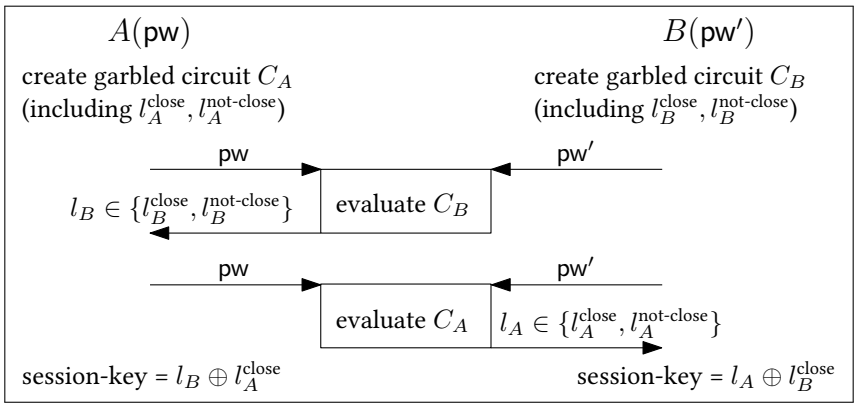

Fig. 1: A simplified presentation of the fPAKE-GC protocol from [Du18].

3.2 Fuzzy PAKE based on error correcting codes (ECC)

In this section we describe the ECC based protocol from [Du18], hereafter fPAKE-ECC. In Appendix B we also discuss a security problem with this protocol, discovered by [Bo23]. The fPAKE-ECC protocol has been implemented for the use in [Fo21]. The implementation is available at `https://github.com/seemoo-lab/fastzip/tree/main`.

While fPAKE-GC can handle any arbitrary password distance measure, fPAKE-ECC is limited to the Hamming distance to the benefit of better performance. The high level idea is to transform the passwords into vectors of high entropy secret keys via PAKE protocols. Then an ECC is used to ensure that key agreement is reached if the key vectors, and equivalently the passwords, coincide on sufficiently many entries (i. e. their Hamming distance is small).

The protocol is shown in Fig. 2. Party A starts by generating a keypair of a signature scheme. The verification key vk is transferred to B via the l-iPAKE instances and is supposed to bind together the l-iPAKE instances and the last protocol message. Both parties then expand their passwords into vectors $\mathbf{K}$ and $\mathbf{K}'$ of high entropy secret keys, by running several PAKE protocol instances. The idea is to interpret a password as vector of its individual characters and then transform it into a vector of high entropy secret keys by running one PAKE instance per password character. By the properties of PAKE, the key vectors contain the same random keys in those positions where the passwords coincide and different random keys in the positions where the passwords differ. Party A now chooses a random value and encodes it using an error correcting code, which is able to correct δ errors. A then blinds the codeword $\mathbf{C}$ by adding the key vector $\mathbf{K}$ and sends the result $\mathbf{E}$ together with the signature σ and vk to B.

Upon receiving this message, B first verifies that vk and the signature are correct. Then, B subtracts their own key vector $\mathbf{K}$ from $\mathbf{E}$ to get $\mathbf{C}'$, thereby undoing most of A's blinding. Now,

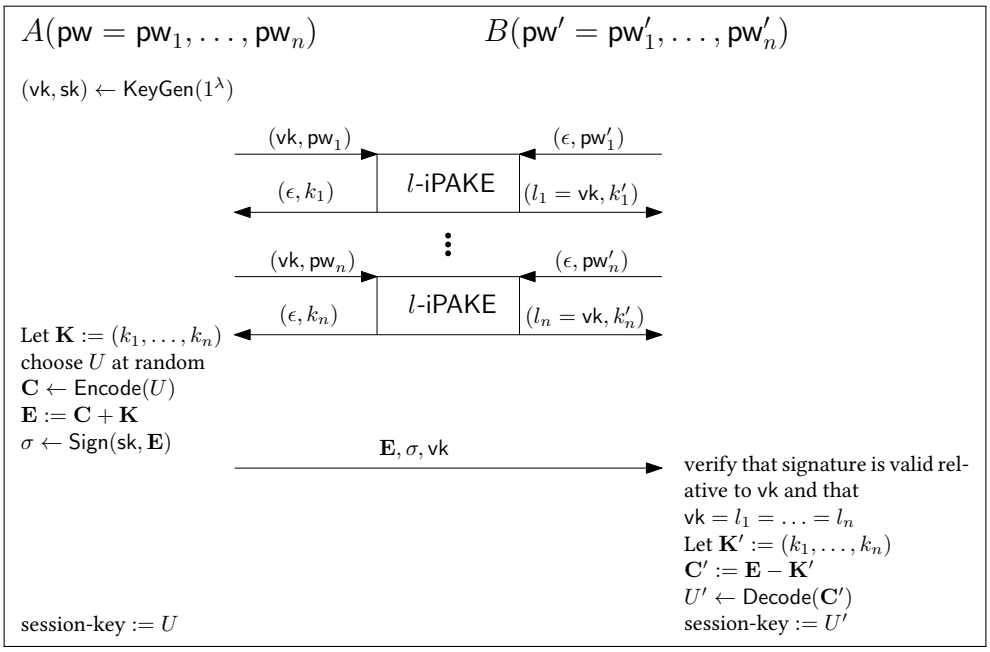

Fig. 2: A slightly simplified version of the fPAKE-ECC protocol from [Du18]

$\mathbf{C}$ and $\mathbf{C}'$ only differ on those positions where the key vectors differ, which are exactly those positions where the passwords differ. Therefore, the distance of the codewords is exactly the distance of the passwords. If the Hamming distance of the passwords $d_H(\mathsf{pw}, \mathsf{pw}')$ is smaller or equal to δ, then the error correcting code is able to correctly recover the original message, so $U' = U$. On the other hand, if $d_H(\mathsf{pw}, \mathsf{pw}') > \delta$ then B only gets a different (and random) value U'[5]. Now both parties use their value U, U' as session-key. These session-keys are the same if and only if $d_H(\mathsf{pw}, \mathsf{pw}') \leq \delta$, which is exactly a requirement of fuzzy PAKE.

They grey-zone - an unfortunate property of codes The ECC based construction has an unfortunate property that is due to properties of error correcting codes. It may allow an attacker to learn extra information even if the attacker only knows a password with a bit more than δ errors. Let us consider the following toy example. We have a password with 10 characters and want to be able to tolerate up to $\delta = 2$ errors. When we have a maximum-distance-separable (MDS) code like the Reed-Solomon code, we can choose $n = 10, k = 6$ and get $\delta = \lfloor \frac{n-k}{2} \rfloor = 2$. So when a party plays the protocol with A and knows a password that differs from A's password in at most 2 positions, then this party can

[5] The fact that U' is random if $d_H(\mathsf{pw}, \mathsf{pw}') > \delta$, is not a property that every code has, but is a property of the concrete construction in [Du18]

reconstruct the secret U and thereby the session-key. This is the intended function of fuzzy PAKE. However, an attacker who only knows a password that differs from A's password in 4 positions may still be able to recover U. This is because a linear error correcting code can only hide its message as long as the attacker knows fewer than $k = 6$ correct elements. If the attacker's password is wrong in 4 positions, they still have 6 correct elements of the codeword. This suffices to recover the message (and thereby U), if the attacker knows which positions are the correct ones.

This means that we have an undesirable grey-zone in which the key exchange fails for two honest parties, but an attacker could still make it succeed. More generally, the key exchange succeeds if the passwords differ in at most δ positions. If the passwords differ in more than $\gamma = 2 \cdot \delta$ positions, then the key exchange is guaranteed to fail. However, if the passwords' distance is in $\{\gamma = 2 \cdot \delta, \ldots, \delta - 1\}$ then an attacker could make the key exchange succeed, even though it should fail. This is the grey-zone. Essentially, now we have two thresholds δ and γ, where δ is the number of errors an honest party can have, while still successfully exchanging a key and γ is the number of errors the attacker can have and still succeed. The grey-zone is shown graphically in Fig. 3.

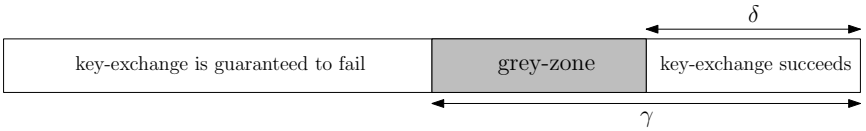

Fig. 3: This figure depicts the grey-zone, which is the zone where an attacker can make the key-exchange succeed, even though it should fail.

The goal of [Bo23] is to make the grey-zone smaller, essentially by increasing δ while keeping γ the same.

Insecurity of the fPAKE-ECC protocol In Appendix B we provide a short description of the attack on the fPAKE-ECC protocol that was discovered by [Bo23].

3.3 Fuzzy PAKE based on ECC with list decoding

The authors of [Bo23] build upon fPAKE-ECC from [Du18] and construct a protocol that manages to leverage the better decoding abilities provided by list-decoding. Their goal is to reduce the grey-zone (Sect. 3.2) described in the previous section. When using a linear $[n, k]$ code with minimal distance δ, unique decoding is possible up to $\lfloor \frac{\delta-1}{2} \rfloor$ errors. Because the Singleton bound limits δ to $d \leq n - k + 1$, the best one can do is to use a code with $\delta = n - k + 1$[6]. Thus, the best one can do with linear codes and unique decoding is to correct up to $\delta = \lfloor \frac{n-k}{2} \rfloor$ errors.

[6] Such codes are called maximum-distance-separable (MDS) codes.

The authors of [Bo23] construct their protocol (hereafter generalized-fPAKE-ECC) from the observation that unique decoding is not necessary. They use list-decoding, which is a kind of decoding that returns a list of possible candidates. In order to narrow down the list of candidates to the one correct codeword, they let one party send a *hint*, which is a hash of the correct codeword. The use of list-decoding enables reconstruction and, thus, successful key exchange even for values $\delta > \lfloor \frac{n-k}{2} \rfloor$. At the same time the value γ (cf. Sect. 3.2) remains unchanged which simply means that the grey-zone is smaller.

The generalized-fPAKE-ECC protocol is depicted in Fig. 4. First, both parties A and B establish a *split authenticated channel*. This is a technique described in [Ba05] for settings in which no authenticated channels are available. Split authenticated channels provide most of the guarantees of authenticated channels. If A and B set up a split authenticated channel, then they are guaranteed that they will be talking to the same party for the entire protocol. A cannot be sure that they are talking to B, because the attacker could impersonate B to A. However, A can be sure that they are either talking to B all the time, or that they are talking to the attacker all the time. This is precisely what prevents the attack that [Bo23] discovered on fPAKE-ECC, because it prevents the attacker from disturbing individual iPAKE sessions.

After setting up the split authenticated channels, both parties continue almost as in fPAKE-ECC (cf. Sect. 3.2). They expand their passwords into key vectors by running one iPAKE session for each password character. However, in this protocol they do not need the labeled iPAKE, instead the un-labeled version suffices. Finally A sends to B the blinded codeword $\mathbf{E}$ and the hint h. Then B unblinds the codeword and uses the list decoding algorithm to obtain a list of candidate codewords. Thereafter, B identifies the correct codeword via the hint h and decodes it to obtain the initial message $\mathbf{s}$, which they then use to compute the session-key.

3.4 Comparison of the protocols

In this section we compare the fuzzy PAKE protocols. We focus on fPAKE-GC from [Du18] and generalized-fPAKE-ECC from [Bo23], thus, excluding fPAKE-ECC from [Du18] because of the security issue, which [Bo23] found with this protocol.

The fPAKE-GC protocol (cf. Sect. 3.1) has two main advantages over the generalized-fPAKE-ECC protocol (cf. Sect. 3.3). The first one is that fPAKE-GC is more flexible, as it can be used to evaluate any distance measure between the passwords, whereas generalized-fPAKE-ECC only covers the Hamming distance. While the Hamming distance is often used for Iris biometrics, there are plenty of other distance measures such as the Euclidean distance used in face biometrics [SKP15] or edit distance for passwords. However, this flexibility should be taken with a grain of salt. In fPAKE-GC the respective distance measure must be evaluated inside the garbled circuit. Although [Du18] provides a very efficient circuit for the Hamming distance, it should be expected that circuits for other distance measures

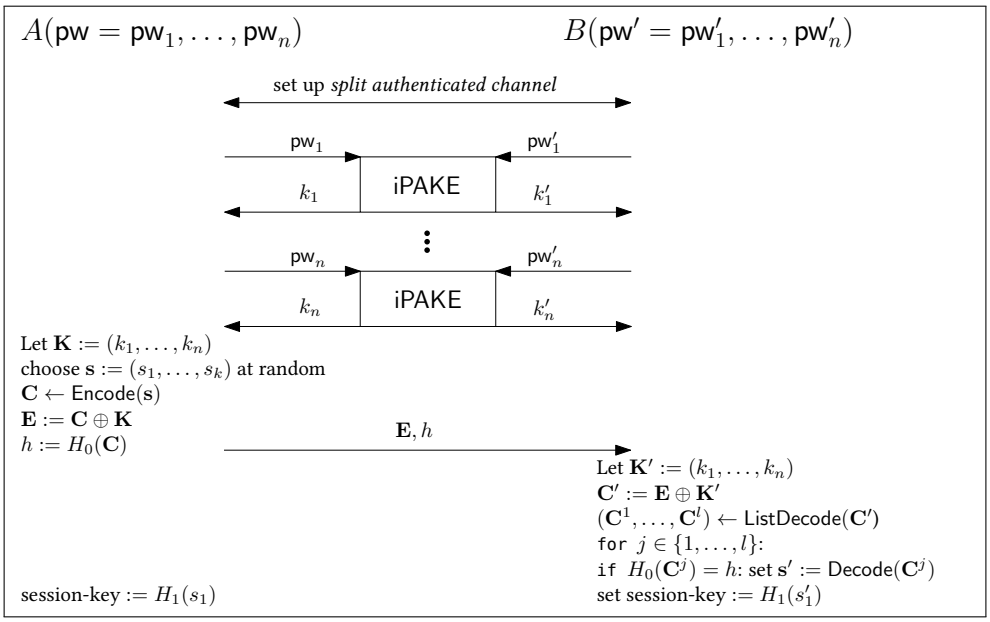

Fig. 4: A slightly simplified version of the generalized ECC based protocol from [Bo23]

are much larger. For example computing the Euclidean distance of vectors of n elements requires n multiplications, which are quite costly in garbled circuits[7].

The second main advantage of the GC based protocol is that it does not have a grey-zone (cf. Sect. 3.2), which means that it is harder for an attacker to successfully guess the password.

The authors of [Du18] compare the efficiency of fPAKE-GC (with a circuit for Hamming distance) and fPAKE-ECC. From their comparison we can conclude that fPAKE-ECC is more efficient, because it needs less exponentiations, hashes, encryptions and signatures than fPAKE-GC. The main difference in efficiency between fPAKE-ECC and generalized-fPAKE-ECC is that generalized-fPAKE-ECC uses split authenticated channels, which adds the exchange of a signature verification key and the need to sign/verify all messages sent/received. On the other hand, generalized-fPAKE-ECC avoids the need to transmit the signature verification key in every iPAKE session. Because generalized-fPAKE-ECC essentially uses the same number of exponentiations, hashes and encryptions as fPAKE-ECC it can also be expected to be more efficient than fPAKE-GC.

Note, however, that there is only an implementation of fPAKE-ECC, so it is not possible to quantify the exact differences in running time between the protocols.

[7] Strictly speaking the Euclidean distance also requires the computation of a square root. However, this can be mitigated by using the squared Euclidean distance instead.

4 Analysis of additional security properties

In this section we perform a heuristic analysis of the security properties that remain in the case where the attacker successfully guesses the password. Due to the forward security of the fuzzy PAKE protocols the attacker is not able to learn anything about the keys from previous sessions. However, when the attacker uses the correct password in an online session, there are not many guarantees that can be retained for that specific session. After all, there is nothing that distinguishes an attacker, who knows the password, from an honest party. The specification in [Du18, Bo23] allows the attacker in that case to both learn the exact password of the attacked party[8] and to decide on the exact session-key. Nevertheless, in this section we argue that in fPAKE-GC from [Du18] an attacker playing as either A or B can only learn one bit of the other party's password. Also we argue that in the generalized-fPAKE-ECC protocol from [Bo23] an attacker playing as A cannot learn anything about B's password and an attacker playing as B has no influence on the session-key. However, these results should be taken with a grain of salt, as we do not provide formal proofs.

4.1 Setting the session-key

Here, we discuss whether an attacker, who compromised a session, can set the session-key.

GC based fPAKE from [Du18] It seems that a party (wlog. A), who knows a close password, can set the session-key to any desired value k. For that, A first evaluates B's garbled circuit to obtain $l_B = l_B^{\text{close}}$. Then A creates C_A and sets the label $l_A^{\text{close}} := l_B^{\text{close}} \oplus k$. Via evaluation of C_A, B obtains l_A^{close} and computes the session-key as $l_B^{\text{close}} \oplus l_A^{\text{close}} = k$.

Generalized ECC based fPAKE from [Bo23] If the attacker plays the role of B, they cannot influence A's session-key. A chooses $\mathbf{s}$ at random and then uses a hash of s_1 as session-key. This is independent of the messages that B sends. On the other hand, A can influence the session-key, because B will use $H(s_1)$ as session-key, where s_1 is chosen by A. So A cannot directly choose the session-key, but they can choose its hash preimage.

4.1.1 Learning information about the password of the other party

In this section we discuss to what extend an attacker, who knows a password close to the password of one of the parties, can learn extra information about that party's password.

[8] Strictly speaking, the specification of the protocols in Sect. 3.2 and Sect. 3.3 allows only the leakage of the indices in which the passwords match. However, for binary passwords this is the same as leaking the exact password.

GC based fPAKE from [Du18] The authors of [Du18] discuss the information an attacker with a close password can obtain. They observe that an attacker (either A or B), who knows a close password, can learn only one bit of the other party's password via the following steps. Wlog. A can do this via garbling a circuit that outputs the label l_A^{close} depending on one bit of B's password. By observing whether the session-keys are the same, A can then learn this one bit of B's password.

Generalized ECC based fPAKE from [Bo23] In the common case of *binary* Hamming distance an attacker playing the role of B, can learn A's exact password, provided that the attacker knows a close password. This is because B learns which positions of the codeword $\mathbf{C}'$ were wrong from the output of the decoding algorithm. These are exactly those positions, where the passwords of A and B differed, so B can recover A's password by inverting their own password in the positions where the codeword was wrong.

On the other hand, it seems that A cannot learn more information about B's password beyond the fact whether both passwords are close enough. The iPAKE sessions do not leak anything about B's password. The other way that A could learn something about B's password is through the session-key that B uses. However, the session-key only depends on B's password, in that it is $H_1(s_1)$ if the passwords are close, or a uniformly random value, otherwise. To see that A cannot make B's session-key depend on B's password, consider the following. Through the hint h, A is committed to the codeword $\mathbf{C}$. Because the decoding algorithm deterministically either outputs the correct message, or "failure", A is also committed to $\mathbf{s}$ via the codeword. Hence, A cannot choose $\mathbf{C}$ and h such that B's session-key depends on their password.

5 Challenges and open problems

Despite the existing research on symmetric fuzzy PAKE protocols, several challenges and open problems remain. One important open problem is to further reduce the size of the grey-zone (cf. Sect. 3.2) in ECC based protocols. Even though [Bo23] already makes progress on this problem, it would still be good to further reduce the size of the grey-zone, or even get rid of it completely.

Another useful direction is to strengthen fuzzy PAKE against attackers, who know the password. Although, as discussed in Sect. 4, in practice the protocols are stronger than their theoretical guarantees, the extra leakage and power of the attacker still seems necessary in the formal proofs of the protocols. Thus, an interesting open question is, whether one can come up with proof techniques or protocols, which allow formally showing these stronger security properties of the protocols.

Finally, it would be useful to also have implementations of generalized-fPAKE-ECC and fPAKE-GC to be able to better compare their efficiency.

Bibliography

[Ba05] Barak, Boaz; Canetti, Ran; Lindell, Yehuda; Pass, Rafael; Rabin, Tal: Secure computation without authentication. In: Advances in Cryptology–CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings 25. Springer, pp. 361–377, 2005.

[Be12] Bender, Jens; Dagdelen, Özgür; Fischlin, Marc; Kügler, Dennis: The PACE| AA protocol for machine readable travel documents, and its security. In: Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers 16. Springer, pp. 344–358, 2012.

[Be23] Beguinet, Hugo; Chevalier, Céline; Pointcheval, David; Ricosset, Thomas; Rossi, Mélissa: GeT a CAKE: Generic Transformations from Key Encaspulation Mechanisms to Password Authenticated Key Exchanges. In: International Conference on Applied Cryptography and Network Security. Springer, pp. 516–538, 2023.

[BHR12] Bellare, Mihir; Hoang, Viet Tung; Rogaway, Phillip: Foundations of garbled circuits. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 784–796, 2012.

[Bo23] Bootle, Jonathan; Faller, Sebastian; Hesse, Julia; Hostáková, Kristina; Ottenhues, Johannes: Generalized Fuzzy Password-Authenticated Key Exchange from Error Correcting Codes. Cryptology ePrint Archive, 2023.

[BPR00] Bellare, Mihir; Pointcheval, David; Rogaway, Phillip: Authenticated key exchange secure against dictionary attacks. In: International conference on the theory and applications of cryptographic techniques. Springer, pp. 139–155, 2000.

[Ca01] Canetti, Ran: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. IEEE, pp. 136–145, 2001.

[Cr22] Cremers, Cas; Naor, Moni; Paz, Shahar; Ronen, Eyal: CHIP and CRISP: protecting all parties against compromise through identity-binding PAKEs. In: Annual International Cryptology Conference. Springer, pp. 668–698, 2022.

[Da09] Daugman, John: How iris recognition works. In: The essential guide to image processing, pp. 715–739. Elsevier, 2009.

[Du18] Dupont, Pierre-Alain; Hesse, Julia; Pointcheval, David; Reyzin, Leonid; Yakoubov, Sophia: Fuzzy password-authenticated key exchange. In: Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part III 37. Springer, pp. 393–424, 2018.

[Er20] Erwig, Andreas; Hesse, Julia; Orlt, Maximilian; Riahi, Siavash: Fuzzy asymmetric password-authenticated key exchange. In: Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26. Springer, pp. 761–784, 2020.

[Fo21] Fomichev, Mikhail; Hesse, Julia; Almon, Lars; Lippert, Timm; Han, Jun; Hollick, Matthias: Fastzip: Faster and more secure zero-interaction pairing. In: Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services. pp. 440–452, 2021.

[HKE12] Huang, Yan; Katz, Jonathan; Evans, David: Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In: 2012 IEEE Symposium on Security and Privacy. IEEE, pp. 272–284, 2012.

[HL18] Haase, Björn; Labrique, Benoît: AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. Cryptology ePrint Archive, 2018.

[Int21] International Civil Aviation Organisation. Security Mechanisms for MRTDs, 2021. 8th edition, `https://www.icao.int/publications/Documents/9303_p11_cons_en.pdf`.

[Ja96] Jablon, David P: Strong password-only authenticated key exchange. ACM SIGCOMM Computer Communication Review, 26(5):5–26, 1996.

[Ja99] Jain, Anil K; Prabhakar, Salil; Hong, Lin; Pankanti, Sharath: FingerCode: a filterbank for fingerprint representation and matching. In: Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149). volume 2. IEEE, pp. 187–193, 1999.

[MF06] Mohassel, Payman; Franklin, Matthew: Efficiency tradeoffs for malicious two-party computation. In: Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9. Springer, pp. 458–473, 2006.

[PZ23] Pan, Jiaxin; Zeng, Runzhi: A Generic Construction of Tightly Secure Password-based Authenticated Key Exchange. Cryptology ePrint Archive, 2023.

[SKP15] Schroff, Florian; Kalenichenko, Dmitry; Philbin, James: Facenet: A unified embedding for face recognition and clustering. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 815–823, 2015.

[Wi10] Wilson, Chuck: Vein pattern recognition: a privacy-enhancing biometric. CRC press, 2010.

[Ya17] Yakoubov, Sophia: A gentle introduction to yao's garbled circuits. preprint on webpage at https://web.mit.edu/sonka89/www/papers/2017ygc.pdf, 2017.

A Deferred preliminaries

In this section we give a short introduction to the concepts that help to understand this paper.

Attack model In (fuzzy) PAKE protocols usually two parties communicate over an insecure channel. This implies that the attacker has full control over the communication, which means that they can drop, delay and modify the parties' messages, as well as send their own messages in the name of any party. Furthermore, the attacker can corrupt any of the parties. The fuzzy PAKE protocols that we discuss in this paper all consider static corruptions, which means that it is assumed that the attacker only corrupts a party before the protocol begins, but not while its running.

Hamming distance The Hamming distance of two vectors is the number of entries in which these two vectors differ. Formally we can write the Hamming distance of two vectors $\mathbf{x} = (x_1, \ldots, x_n)$ and $\mathbf{y} = (y_1, \ldots, y_n)$ as $d_H(\mathbf{x}, \mathbf{y}) = |\{i : x_i \neq y_i\}|$. The Hamming distance is used as measure of password closeness in Sect. 3.2 and Sect. 3.3.

Oblivious transfer Oblivious Transfer (OT) allows two parties (a sender and a receiver) to exchange a message in the following way. The sender provides two messages (m_0, m_1) and the receiver can choose one of them. An important property of OT protocols is that the receiver can only learn one of the two messages, but not the other one. At the same time, the sender does not learn which message the receiver chose. OT is a useful tool in many protocols and is particularly important for the use with garbled circuits.

Garbled circuits Garbled Circuits (GC) are a tool from secure multi party computation that allows two parties to jointly evaluate a function on their (private) inputs without revealing anything about their respective inputs to the other party (except of what can be inferred from the output itself). Garbled circuits are (in the standard construction) a set of tables of ciphertexts that are set up in such a way that one can —by using the correct *wire labels* —decrypt one ciphertext from each table. For a more detailed introduction to garbled circuits we refer to [Ya17] and for more formal definitions, constructions and security properties we refer to [BHR12].

Error correcting codes (ECC) In this paper we use linear block codes over finite fields. These codes can be represented by a generator matrix $G \in \mathbb{F}^{n \times k}$ with $n, k \in \mathbb{N}$ and $n > k$. A message $m \in \mathbb{F}^k$ can be encoded by computing $G \cdot m$. We say that the code can correct up to δ errors, if there is a sufficiently efficient decoding algorithm that deterministically outputs the correct message, if the codeword contains no more than δ errors.

Password authenticated key exchange (PAKE) In a PAKE protocol two parties interact. They each use a password as input and both obtain the same (uniformly random) session-key if and only if their passwords match. This session-key is then usually used to set up a secure communication channel. The security guarantees provided by most PAKE protocols are that no information about the password is leaked beyond the fact whether they match. Furthermore it must not be possible for an attacker to perform an offline attack, so that the best an attacker can do is an online attack, which is unavoidable. The PAKE protocol should also be forward secure, i. e. if an attacker later learns the correct password, they must not be able to deduce any information about the session-key.

The same security guarantees also apply for fuzzy PAKE protocols, except that the passwords are only required to be similar instead of identical.

Implicit-only PAKE and labeled implicit-only PAKE A notion that is relevant to [Du18] and [Bo23] is the notion of implicit-only PAKE (iPAKE). A PAKE protocol has the property that both parties compute the same random session-key if and only if their input passwords are the same. However, the security specification doesn't say anything about whether the parties learn if they got the same key (i. e. have the same password). On the contrary, the notion of implicit-only PAKE requires that even a maliciously behaving party cannot figure out if they have the same password as their counterpart. Both the fPAKE-ECC (cf. Sect. 3.2) and the generalized-fPAKE-ECC (cf. Sect. 3.3) protocol crucially rely on this property.

An extension of iPAKE that is used in fPAKE-ECC is *labeled* iPAKE (l-iPAKE). In l-iPAKE, additional to the key-exchange, a public label can be transmitted. The authenticity of the label is guaranteed, given that both parties compute the same key from the l-iPAKE. This is used in the fPAKE-ECC (cf. Sect. 3.2) to allow A to transmit their signature verification key to B such that the attacker cannot tamper with it.

B Attack on the ECC based protocol from [Du18]

The authors of [Bo23] describe an attack on the fPAKE-ECC protocol that can allow an attacker to learn one bit of the parties password per online session in which both A and B participate. We present the attack in the simplest possible setting. For this, let us assume that $\delta = 0$, i. e. the passwords need to be identical for the key exchange to succeed. Let us further assume that the passwords are binary so that A and B run one session of l-iPAKE per password bit and that the attacker wants to learn the first bit of the password. Finally, we assume that A and B use the same password $\mathsf{pw} = (\mathsf{pw}_1, \ldots, \mathsf{pw}_n)$, so that the key exchange succeeds, if the attacker does not interfere.

To carry out the attack, the attacker disturbs the first l-iPAKE session in such a way that A and B get the same key from that session if $\mathsf{pw}_1 = 0$ and different keys, otherwise[9]. The attacker lets through all other messages without modification. If both parties got the same key from the l-iPAKE session, then they will compute the same fPAKE session-key. However, if they got different keys from the l-iPAKE session, then they will also compute different fPAKE session-keys, because the error tolerance was $\delta = 0$. Thus, the fact whether A and B get the same fPAKE session-key depends on whether $\mathsf{pw}_1 = 0$. Now the attacker only has to find out, if both parties got the same fPAKE session-key. This is usually easy, because A and B will often use this session-key to exchange encrypted messages. However, if their keys are different, they will not be able to decrypt each others messages. This will

[9] In [Du18] the l-iPAKE is modeled as ideal functionality. The attacker can make both parties get the same key conditioned on $\mathsf{pw}_1 = 0$ by sending the following queries to this ideal functionality, after both A and B started the l-iPAKE session with session-identifier sid and label l. The attacker sends (TestPwd, sid, A, $\mathsf{pw} = 0$, l) and (TestPwd, sid, B, $\mathsf{pw} = 0$, l) to the ideal functionality. If $\mathsf{pw}_1 = 0$, then both sessions are now *compromised* and if $\mathsf{pw}_1 = 1$, then both sessions are *interrupted*. Then the attacker sends (NewKey, sid, A, sk) and (NewKey, sid, B, sk) to the ideal functionality, for random sk. If the sessions had been compromised, then the ideal functionality gives sk to both A and B as key. Otherwise, both parties get distinct random keys.

likely cause them to behave differently, e.g. they may attempt to run the fPAKE protocol again. By observing this the attacker can finally deduce the first bit of the password.

Note that variants of this attack can also be mounted when $\delta > 0$. However, in that case the attacker may need multiple sessions to reliably find out a specific bit of the password.