Oblivious PAKE: Efficient Handling of Password Trials

Franziskus Kiefer(✉) and Mark Manulis

Surrey Centre for Cyber Security Department of Computer Science,
University of Surrey, Surrey, UK
mail@franziskuskiefer.de, mark@manulis.eu

Abstract. In this work we introduce *Oblivious Password based Authenticated Key Exchange* (O-PAKE) and show how ordinary PAKE protocols can be transformed into O-PAKE. O-PAKE allows a client that holds multiple passwords and is registered with *one* of them at some server to use any *subset* of his passwords in a PAKE session with that server. The term *oblivious* is used to emphasise that the only information leaked to the server is whether the one password used on the server side matches any of the passwords input by the client. O-PAKE protocols can be used to improve the overall efficiency of login attempts using PAKE protocols in scenarios where users are not sure (e.g. no longer remember) which of their passwords has been used at a particular web server. Using special processing techniques, our O-PAKE compiler reaches nearly constant run time on the server side, independent of the size of the client's password set; in contrast, a naive approach to run a new PAKE session for each login attempt would require linear run time for both parties. We prove security of the O-PAKE compiler under standard assumptions using the latest game-based PAKE model by Abdalla, Fouque and Pointcheval (PKC 2005), tailored to our needs. We identify the requirements that standard PAKE protocols must satisfy in order to suit our O-PAKE transformation and give two examples.

1 Introduction

Authentication with passwords is the most common (and perhaps most critical) authentication mechanism on the modern Internet. The dominating approach today is when clients send passwords (or some function thereof) to the server over a secure channel (e.g. TLS [18]). This approach requires PKI and its security relies solely on the secure channel and the client's ability to correctly verify the server's certificate. Any impersonation of the certificate leads to password exposure. Even if no impersonation takes place, any password input on the client side is revealed to the server. This creates a different problem based on statistics, indicating that many users operate with a small set of passwords but often do not remember their correct mapping to the servers. If a user types in a password that is not shared with this server but with another one then its exposure may lead to subsequent impersonation attacks on the client. The studies in [20,21]

J. Lopez and C.J. Mitchell (Eds.): ISC 2015, LNCS 9290, pp. 191–208, 2015.
DOI: 10.1007/978-3-319-23318-5_11

show that every user has 6.5 passwords on average, used on 25 different websites and that on average 2.4 password trials are required until the user types in the correct password. These numbers suggest that in case where a server limits a number of failed attempts to say 3, in the worst case roughly 2 passwords from the client's set could potentially be revealed to the server within a single TLS session — a significant threat for the client. Note that the amount of work for processing failed login attempts on the server side is negligible since all trials are performed through the same secure channel.

The notion of *Password-based Authenticated Key Exchange (PAKE)*, introduced by Bellovin and Merritt [8], initially formalised in [6,14], and later explored in numerous further works [1–5,11,16,22,23,26,27], is considered as a more secure alternative to the above approach. The standard model of PAKE does not require any PKI and assumes that only a human-memorable password is shared between both parties. PAKE protocols solve the problem of potential password leakage, inherent to the previously described approach. They aim to protect against offline dictionary attacks but require the same method of protection against online dictionary attacks as the aforementioned TLS-based approach, namely by restricting the number of failed password trials. While passwords can be retransmitted and checked by the server, using the same TLS channel, the only way for current PAKE protocols to deal with failed password trials is to repeat the entire protocol. This however implies that the computational costs on the server side, in particular for (costly) public key-operations that are inherent to all PAKE protocols, increase *linearly* with the number of attempts. This can be seen as a reason for the limited progress on the adoption of PAKE on the Internet (in addition to unrelated issues such as browser incompatibility, patent considerations, and the lack of adopted standards).

While handling multiple password trials with PAKE may seem like a pure implementation problem at first sight, the problem becomes non-trivial if we want to avoid linear increase of public key operations on the server side. This seems to be avoidable only if in a single PAKE execution the client can use several passwords, while the server would use only the one password, shared with the client. Yet this idea alone is not sufficient for breaking the linear bound on the server side: for instance, assume that one PAKE execution is built out of n independent (possibly parallelised) runs of some secure PAKE protocol, where the client uses a different password in each run but the server uses the same one in all of them. The amount of work for the server in this case would still remain $O(n)$. Therefore, something non-trivial must additionally happen in order to reduce the amount of work on the server side to $O(1)$.

However, we still need to fulfil basic PAKE requirements like addressing the persistent threat of online dictionary attacks by enforcing that the number of passwords that can be tested by the client in one session remains below some threshold, which is set by the server. For the server there is no difference whether a client is given the opportunity to perform at most c independent PAKE sessions (password trials) with one input password per session, or only one session but with at most c input passwords. Finally, we must be able to prevent a possibly

malicious server from obtaining any password from the set of passwords input by the client.

1.1 Oblivious PAKE and Our Contributions

We solve the aforementioned problem of efficient handling of password trials on the server side by proposing a compiler that transforms PAKE protocols in a black-box way into what we call an *Oblivious PAKE* (O-PAKE). To describe and analyse the proposed O-PAKE notion we introduce a new algorithmic way to model PAKE protocols that also allows for easy compilation as done with O-PAKE, and real-world implementation.

The functionality of O-PAKE protocols resembles that of PAKE except that the client inputs a set **pw** of $n \in [1, c]$ passwords while the server's input is limited to one password pw. The use of **pw** does *not* increase the overall probability for online dictionary attacks in comparison to running a separate PAKE session for each tried password because the *maximum number* of passwords c that the client can try with O-PAKE is fixed by the server. The client can still input less than c passwords, i.e. if the client is confident about validity of some particular pw for a given server then pw can be used as the sole input, in which case O-PAKE is equivalent to PAKE. In general, O-PAKE protocol execution succeeds if and only if the server's password pw is part of the client's password set **pw**. We use the standard (game-based) PAKE model by Bellare, Pointcheval, and Rogaway [6] in its (stronger) Real-or-Random flavour from [4] and update it to account for the use of **pw** as client's input. In this model passwords are assumed to be distributed uniformly at random. In practice, the use of passwords with different strengths in the same O-PAKE session would lower the overall security to the probability for guessing the weakest password (irrespective of the adopted strength metric).

The crucial idea behind our O-PAKE compiler is to let each client execute n sessions of secure PAKE protocol in parallel and let the server execute only *one* PAKE session. The challenging part is to enable the server to actually identify the correct PAKE session in which the client used the correct password pw, while preserving security against offline dictionary attacks for all passwords in the client's password set **pw**. This is the trickiest part of the compiler. Intuitively, if the server can recover the messages of the correct PAKE session, it can answer them according to the specification of the PAKE protocol. By repeating this approach in each communication round of the given PAKE protocol both parties will be able to successfully accomplish the protocol. If identification of the correct PAKE session by the server requires only a constant amount of (costly) operations, then the total amount of server's work in the resulting O-PAKE protocol will also remain constant. The amount on the client side remains linear in the size n of input passwords. This stems from the obvious fact that the client has to compute messages for all PAKE sessions without knowing the correct password. We show how to apply our O-PAKE compiler to two concrete PAKE protocols: the SPAKE protocol from [5] and the PAKE protocol from [28] (for space limitations the second construction is given in the full version of this work [29]).

2 Oblivious PAKE Model

In this section we recall the PAKE security model from [4], tailored to the needs of O-PAKE. The security model for O-PAKE protocols is in the multi-user setting and utilises the Real-or-Random approach for AKE-security from [4,6]. Note that the AKE-security definition addresses the aforementioned security against malicious servers, trying to retrieve client passwords. A server learning information about the additional passwords in the client's password set **pw** can easily break AKE-security by using this password in another session with the same client.

Participants and Passwords. An O-PAKE protocol is executed between two parties P and P', chosen from the universe of participants $\Omega = \mathcal{S} \cup \mathcal{C}$, where $\mathcal{S}$ denotes the universe of servers and $\mathcal{C}$ the universe of clients, such that if $P \in \mathcal{C}$ then $P' \in \mathcal{S}$, and vice versa. We assume the scenario where every client in $\mathcal{C}$ is registered with every servers from $\mathcal{S}$. For each such pair $(P, P') \in \mathcal{C} \times \mathcal{S}$, a password $\mathrm{pw}_{P,P'}$ (shared between client P and server P') is drawn uniformly at random from the dictionary $\mathcal{D}$ of size $|\mathcal{D}|$. Execution of an oblivious PAKE protocol between P and P' uses $\mathrm{pw}_{P,P'}$ on the server and a password vector $\mathbf{pw}_P \subseteq \{\mathrm{pw}_{P,P'_{x_1}}, \ldots, \mathrm{pw}_{P,P'_{x_n}}\}$ for $1 \leq n \leq c$ and client-server pairs (P, P'_{x_i}) for $i \in [2, n]$ on the client side. For the protocol to be successful it is necessary that $\mathrm{pw}_{P,P'} \in \mathbf{pw}_P$. The value c is a global parameter with $c \leq |\mathcal{S}|$. We will sometimes write **pw** and pw instead of $\mathbf{pw}_P$ and $\mathrm{pw}_{P,P'}$ when the association with the participants is clear or if it applies to every participant. We will further write PAKE for O-PAKE protocols with $n = 1$, i.e. standard class of PAKE protocols where the client uses $\mathbf{pw}_P = \mathrm{pw}_{P,P'}$.

Protocol Instances. For $i \in \mathbb{N}$, we denote by P_i the i-th instance of $P \in \Omega$. In order to model uniqueness of P_i within the model we use i as a counter. For each instance P_i we consider further a list of parameters:

- $\mathtt{pid}_P^i$ is the partner id of P_i, defined upon initialisation, subject to following restriction: if $P_i \in \mathcal{C}$ then $\mathtt{pid}_P^i \in \mathcal{S}$, and if $P_i \in \mathcal{S}$ then $\mathtt{pid}_P^i \in \mathcal{C}$.
- $\mathtt{sid}_P^i$ is the session id of P_i, modelled as ordered (partial) protocol transcript $[m_{\text{in}}^1, m_{\text{out}}^1, \ldots, m_{\text{in}}^r, m_{\text{out}}^r]$ of incoming and outgoing messages of P_i in rounds 1 to r. $\mathtt{sid}_P^i$ is thus updated on each sent or received protocol message.
- $\mathtt{k}_P^i$ is the value of the session key of instance P_i, which is initialised to `null`.
- $\mathtt{state}^{i_P}$ is the internal state of instance P_i.
- $\mathtt{used}^{i_P}$ indicates whether P_i has already been used.
- $\mathtt{role}_P^i$ indicates whether P_i acts as a `client` or a `server`.

Partnered Instances. Two instances P_i and P'_j are *partnered* if all of the following holds: (i) $(P, P') \in \mathcal{C} \times \mathcal{S}$, (ii) $\mathtt{pid}_P^i = P'$ and $\mathtt{pid}_{P'}^j = P$, and (iii) $\mathtt{match}(\mathtt{sid}_P^i, \mathtt{sid}_{P'}^j) = 1$, where Boolean algorithm `match` is defined according to the matching conversations from [7], i.e. outputs 1 if and only if round

messages (in temporal order) in $\mathtt{sid}_P^i$ equal to the corresponding round messages in $\mathtt{sid}_{P'}^j$ except for the final round, in which the incoming message of one instance may differ from the outgoing message of another instance.

Oblivious PAKE. We define O-PAKE using an initialisation algorithm `init` and a stateful interactive algorithm `next`, which handles protocol messages and eventually outputs the session key.

Definition 1 (Oblivious PAKE). *An O-PAKE protocol* $\mathtt{O\text{-}PAKE} = (\mathtt{init}, \mathtt{next})$ *over a message space* $\boldsymbol{\mathcal{M}} = (\bigcup_r \mathcal{M}_C^r) \cup (\bigcup_r \mathcal{M}_S^r)$*, where* $\mathcal{M}_C^r$ *resp.* $\mathcal{M}_S^r$ *denotes the space of outgoing server's resp. client's messages in the* r*-th invocation of* `next`*, a dictionary* $\mathcal{D}$*, and a key space* $\mathcal{K}$ *consists of two polynomial-time algorithms:*

- $P_i \leftarrow \mathtt{init}(\mathbf{pw}, \mathtt{role}, P', \mathtt{par})$*: On input* $\mathbf{pw}$*,* $\mathtt{role} \in \{\mathtt{client}, \mathtt{server}\}$*,* $P' \in \Omega$ *and the public parameters* `par`*, the algorithm initialises a new instance* P_i *with the internal O-PAKE state information* `state`*, defines the intended partner id as* $\mathtt{pid}_P^i = P'$ *and session key* $\mathtt{k}_P^i = \mathtt{null}$*, and stores protocol parameters* `par`*. The* `role` *indicates whether the participant acts as* `client` *or* `server`*.*
- $(m_{\text{out}}, \mathtt{k}_P^i) \leftarrow \mathtt{next}(m_{\text{in}})$*: On input* $m_{\text{in}} \in \mathcal{M}_{[S,C]}^r \cup \emptyset$ *with implicit access to internal* `state`*, the algorithm outputs the next protocol message* $m_{\text{out}} \in \mathcal{M}_{[S,C]}^{r+1} \cup \emptyset$ *and updates* $\mathtt{k}_P^i$ *with* $\mathtt{k}_P^i \in \mathcal{K} \cup \mathtt{null} \cup \perp$*. As long as the instance has not terminated the key* $\mathtt{k}_P^i$ *is* `null`*. If* m_{in} *leads to acceptance then* $\mathtt{k}_P^i$ *is from* $\mathcal{K}$*, otherwise* $\mathtt{k}_P^i = \perp$*. We also assume that* `next` *implicitly updates the internal* `state` *prior to each output and sets* `used` *to* `true`*.*

Note that $\boldsymbol{\mathcal{M}} = (\bigcup_r \mathcal{M}_S^r) \cup (\bigcup_r \mathcal{M}_C^r)$ is the union of outgoing client's message spaces $\mathcal{M}_C^r$ and server's message spaces $\mathcal{M}_S^r$ over all protocol rounds r. We may further view each round's message space $\mathcal{M}_C^r$ as a Cartesian product $\mathcal{M}_C^{r,1} \times \cdots \times \mathcal{M}_C^{r,l}$ for up to l different classes of message components, e.g. to model labels, identities, group elements, etc. When clear from the context, we will write $\mathcal{M}_C^r$ instead of $\mathcal{M}_C^{r,1} \times \cdots \times \mathcal{M}_C^{r,l}$.

Correctness. Let P_i be an instance initialised through $\mathtt{init}(\mathbf{pw}_P, \mathtt{client}, P', \mathtt{par})$ and P'_j be an instance initialised through $\mathtt{init}(\mathrm{pw}_{P,P'}, \mathtt{server}, P, \mathtt{par})$ where $P \in \mathcal{C}$, $P' \in \mathcal{S}$, and $\mathrm{pw}_{P,P'} \in \mathbf{pw}_P$. Assume that all outgoing messages, generated by `next` are faithfully transmitted between P_i and P'_j so that the instances become partnered. An $\mathtt{O\text{-}PAKE} = (\mathtt{init}, \mathtt{next})$ is said to be *correct* if for all partnered P_i and P'_j it holds that $\mathtt{k}_P^i \in \mathcal{K}$ and $\mathtt{k}_P^i = \mathtt{k}_{P'}^j$.

Adversary Model. The adversary $\mathcal{A}$ is modelled as a probabilistic-polynomial time (PPT) algorithm, with access to the following oracles:

$m_{\text{out}} \leftarrow \mathsf{Send}(P, i, m_{\text{in}})$: the oracle processes the incoming message $m_{\text{in}} \in \mathcal{M}^{r}_{[C,S]}$ for the instance P_i and returns its outgoing message $m_{\text{out}} \in \mathcal{M}^{r+1}_{[C,S]} \cup \emptyset$. If P_i does not exist, a new session is created with P' as partner, where P' is given in m_{in}.

$\texttt{trans} \leftarrow \mathsf{Execute}(P, P')$: if $(P, P') \in \mathcal{C} \times \mathcal{S}$ the oracle creates two new instances P_i and P'_j via appropriate calls to `init` and returns the transcript `trans` of their protocol execution, obtained through invocations of corresponding `next` algorithms and faithful transmission of generated messages amongst the two instances.

$\text{pw} \leftarrow \mathsf{Corrupt}(P, P')$: if $P \in \mathcal{C}$ and $P' \in \mathcal{S}$ then return $\text{pw}_{P,P'}$ and mark (P, P') as a corrupted pair.

AKE-Security. The following definition of AKE-security follows the *Real-Or-Random* (ROR) approach from [4], which provides the adversary multiple access to the Test oracle for which the randomly chosen bit $b \in_R \{0, 1\}$ is fixed in the beginning of the experiment:

$\mathtt{k}_{\mathcal{A}} \leftarrow \mathsf{Test}_b(P, i)$, depending on the values of bit b and $\mathtt{k}^i_P$, this oracle responds with key $\mathtt{k}_{\mathcal{A}}$ defined as follows:

- If, while $\mathtt{k}^i_P = \texttt{null}$, either (P, P') or $(\mathtt{pid}^i_P, P)$ were queried to the $\mathsf{Corrupt}$ oracle for, w.l.o.g., any client-server pair (P, P') with $\text{pw}_{P,P'} \in \mathbf{pw}_P$, then abort. Note that this prevents $\mathcal{A}$ from obtaining any $\text{pw}_{P,P'} \in \mathbf{pw}$ and then testing new instances of P and P', or instances that were still in the process of establishing session keys when corruption took place.
- If some previous query $\mathsf{Test}(P', j)$ was asked for an instance P'_j, which is partnered with P_i, then return the same response as to that query. Note that this guarantees consistency of oracle responses.
- If $\mathtt{k}^i_P \in \mathcal{K}$ then if $b = 1$, return $\mathtt{k}^i_P$, else if $b = 0$, return a randomly chosen element from $\mathcal{K}$ and store it for later use.
- Else return $\mathtt{k}^i_P$. Note that in this case $\mathtt{k}^i_P$ is either $\perp$ or $\texttt{null}$.

According to [4] a session is an online session when $\mathcal{A}$ queried the Send oracle on one of the participants.

Definition 2 (AKE-Security). *An O-PAKE protocol Π with up to c passwords on client side is* AKE-secure *if for all dictionaries $\mathcal{D}$ with corresponding universe of participants Ω and for all PPT adversaries $\mathcal{A}$ using at most t online sessions there exists a negligible function $\varepsilon(\cdot)$ such that:*

$$\mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda) = \left| \Pr[\mathsf{Exp}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda) = 1] - \frac{1}{2} \right| \leq \frac{c \cdot \mathcal{O}(t)}{|\mathcal{D}|} + \varepsilon(\lambda).$$

$\mathsf{Exp}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}(\lambda)$: $c \in \mathbb{N}; b \in_R \{0,1\}; \forall (P, P') \in \mathcal{C} \times \mathcal{S}$ *choose* $\text{pw}_{P,P'} \in_R \mathcal{D}$; $b' \leftarrow \mathcal{A}^{\mathsf{Send},\mathsf{Execute},\mathsf{Corrupt},\mathsf{Test}_b}(\lambda, c)$; *return* $b = b'$.

The above definition (without $\mathsf{Corrupt}$) reverts to RoR AKE-security from [4] for $c = 1$. We have to factor in the maximal size of $|\mathbf{pw}| = n \leq c$ into the original adversarial advantage bound $\mathcal{O}(t)/|\mathcal{D}|$ to account for the adversarial possibility of testing up to c passwords per session in the role of the client.

PAKE vs. O-PAKE. The actual relation between common PAKE and O-PAKE security may not be immediately evident. For clarification, we discuss the relation between O-PAKE and the simple repetition of a PAKE protocol c times, and the implication of user's password choice.

The advantage of an adversary that is allowed to query up to c passwords in one session is not greater than the advantage of an adversary that runs c online sessions using one password in each of them. The typical advantage of a PAKE adversary $\mathcal{A}$ in an AKE-security experiment, e.g. [4,6], is bounded by $\mathcal{O}(t)/|\mathcal{D}| + \varepsilon(\lambda)$. In contrast, we limit the advantage of an O-PAKE adversary to $c \cdot \mathcal{O}(t)/|\mathcal{D}| + \varepsilon'(\lambda)$. We give the following lemma to formalise the relation between the two notions.

Lemma 1. $\mathsf{Adv}^{\mathsf{AKE}}_{\Pi_c,\mathcal{A}} \leq c \cdot \mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{A}}$ *for O-PAKE protocol* Π_c *allowing up to* c *passwords in one session, built from PAKE protocol* Π*.*

Proof. The lemma follows directly from the following observations. O-PAKE can be realised in the naïve way by running c separate PAKE sessions. That results in an advantage of at most $c \cdot \mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{A}} = c \cdot \mathcal{O}(t)/|\mathcal{D}| + \varepsilon'(\lambda)$. Information gathered from Send and $\mathsf{Execute}$ oracle invocations are the same for the O-PAKE and PAKE adversary. $\mathsf{Corrupt}$ and Test_b queries of the O-PAKE adversary return one password, respectively key, independent from c, while the PAKE adversary gets c passwords, respectively keys. Thus, the resulting advantage of the O-PAKE adversary is at most $c \cdot \mathcal{O}(t)/|\mathcal{D}| + \varepsilon'(\lambda)$, but depending on the implementation most probably lower. □

Assuming malicious servers one may also be concerned about the client's password choice considering a client entering passwords with different levels of entropy. Similar to the standard PAKE case the weakest password from **pw** would determine the security of O-PAKE. However, the used model considers uniformly at random chosen passwords from one dictionary such that the case of varying password probabilities can not be adequately addressed in this model (as is also the case for the models in [4,6]).

3 Transforming PAKE Protocols into O-PAKE

Recall that one may realise O-PAKE in a naïve way by running the input PAKE protocol n times, which is not efficient on the server side due to the linearly increasing round complexity. The idea of the O-PAKE compiler is to mix the n PAKE messages on client side such that the server can extract the "right" message using the shared password and reply only to that. This, however, is a non-trivial problem because PAKE messages do not provide information that would allow the server to check locally whether a given password was used in their computation; as this would offer the possibility of offline dictionary attacks. Note that we assume throughout this section that $n \geq 2$ and $\mathrm{pw}_{P,P'} \in \mathbf{pw}_P$. Our solution for the identification of the "right" PAKE session is a careful composition of two encoding techniques that were introduced in a different context

yet allow us to generically construct AKE-secure O-PAKE protocols from (suitable) AKE-secure PAKE protocols, preserving constant round complexity and offering nearly constant server load.

Our first building block is *Index-Hiding Message Encoding (IHME)* [30,31]. An IHME scheme assigns a different index to each given message and encodes the resulting index-message pairs into a single structure from which messages can be recovered on the receiver side using the corresponding indices. The IHME structure hides indices that were used for encoding and therefore all encoded messages must contain enough entropy to prevent dictionary attacks over the index space. An IHME scheme consists of two algorithms `iEncode` and `iDecode`. The `iEncode` algorithm takes as input a set of index-message pairs $(\mathrm{ix}_1, m_1), \ldots, (\mathrm{ix}_n, m_n)$ and outputs a structure S whereas the `iDecode` algorithm can extract m_j, $j \in [1, n]$ from S using the corresponding index ix_j. For formal definitions surrounding IHME we refer to the original work and only mention that the original IHME construction in [30] assumes $(\mathrm{ix}_j, m_j) \in \mathbb{F}$ for a prime-order finite field $\mathbb{F}$ and defines the IHME structure S through coefficients of the interpolated polynomial by treating index-message pairs as its points. There exists a more efficient IHME version from [31] for longer messages, which uses $(\mathrm{ix}_j, m_j) \in \mathbb{F} \times \mathbb{F}^\nu$ and thus splits m_j into ν components each being an element of $\mathbb{F}$. The corresponding index-hiding property demands that no information about indices ix_j is leaked to the adversary that doesn't know the corresponding messages m_j and is defined for messages that are chosen uniformly from the IHME message space. For the aforementioned IHME schemes the message space is given by $\mathbb{F}$ (or $\mathbb{F}^\nu$) and their index-hiding property is perfect (in the information-theoretic sense). Note that this approach still allows the server to learn which of the n PAKE sessions is the correct one without revealing any password to the server.

In order to enable encoding of PAKE messages using IHME with passwords as indices we apply our second building block, namely *admissible encoding* [13,15,19]. Briefly, a function $F : S \to R$ is an *ϵ-admissible encoding* for (S, R) with $|S| > |R|$ when for all uniformly distributed $r \in R$, the distribution of the inverse transformation $\mathcal{I}_F(r)$ is ϵ-statistically indistinguishable from the uniform distribution over S. We refer to [15,19] for more details. $\mathcal{I}_F$ enables us to map PAKE messages into the IHME message space where necessary. In Sect. 3.5 we will discuss suitable PAKE message spaces and their admissible encodings offering compatibility with the message space $\mathbb{F}$ of the IHME schemes from [30,31].

In the following we describe our compiler that transforms suitable AKE-secure PAKE protocols into AKE-secure O-PAKE protocols. The intuition behind the compiler is to let the client run n PAKE sessions, one session for each of the n input passwords **pw**, and apply an index-hiding message encoding on each message-password pair. The server can apply the shared password pw as index to IHME to extract the "right" PAKE message. For this message the server executes the algorithm `next` of the given PAKE protocol and returns the resulting PAKE message to the client. As soon as the algorithm `next` terminates, the server generates a confirmation message, which is then used by the client to derive the final session key.

3.1 Requirements on PAKE

Our O-PAKE compiler can be used to convert any AKE-secure R-round PAKE protocol Π where in each round $r \in [1, \ldots, R]$ the client sends messages from $\mathcal{M}_C^r$ that can be processed using a compatible admissible encoding $F^r : \mathcal{M}^{\mathtt{IHME},r} \rightarrow \mathcal{M}_C^r$. In order to guarantee that client messages from $\mathcal{M}_C^r$, when mapped into $\mathcal{M}^{\mathtt{IHME},r}$ using the inverse transformation $\mathcal{I}_F$, are uniformly distributed over $\mathcal{M}^{\mathtt{IHME},r}$, the underlying Π itself must output client messages whose joint distribution over all R rounds remains indistinguishable from a distribution where for each round r the output client message is chosen uniformly at random from $\mathcal{M}_C^r$. For this purpose Π must satisfy a stronger notion of AKE security that in addition to the indistinguishability of session keys requires indistinguishability of client messages. This requirement is formalised in Definition 3 that extends the AKE-security experiment for PAKE from Definition 2, using $\mathsf{Execute}, \mathsf{Send}, \mathsf{Corrupt}$ and Test_b definitions from there. We assume that $c = 1$ and define two oracles Send_b and $\mathsf{Execute}_b$ that are parameterised with the bit b as used in the Test_b oracle. Any query $\mathsf{Send}_b(P, i, m_{\text{in}})$ for a client $P \in \mathcal{C}$ made by the adversary $\mathcal{A}$ first triggers the invocation of $m_{\text{out}} \leftarrow \mathsf{Send}(P, i, m_{\text{in}})$. If $\mathcal{A}$ queried $\mathsf{Corrupt}(P, \mathtt{pid}_P^i)$ or $\mathsf{Corrupt}(\mathtt{pid}_P^i, P)$ while $\mathtt{k}_P^i = \mathtt{null}$ or if $b = 1$ then m_{out} is returned to $\mathcal{A}$ without any modification. The additional condition on the $\mathsf{Corrupt}$ queries prevents $\mathcal{A}$ from trivially distinguishing the client messages by corrupting passwords and then communicating with client instances that were still in the process of establishing the session keys. If $b = 0$ then m_{out} is set to a random message from $\mathcal{M}_C^r$ and returned to $\mathcal{A}$. Any $\mathsf{Execute}_b(P, P')$ query first triggers the invocation of $\mathtt{trans} \leftarrow \mathsf{Execute}(P, P')$. If $b = 0$ then for each round r the corresponding client's message in $\mathtt{trans}$ is replaced with an independently at random chosen message from $\mathcal{M}_C^r$, else if $b = 1$ then $\mathtt{trans}$ is forwarded without any modification. Note that if $\mathcal{A}$ mounts an online attack with a correct password then it can easily distinguish so that the lower bound of $\frac{\mathcal{O}(t)}{|\mathcal{D}|}$ that accounts for online dictionary attacks still applies in the definition.

Definition 3 (AKE-Security with Indistinguishable Client Messages). *A PAKE protocol Π is* AKE-secure with indistinguishable client messages *if for all dictionaries $\mathcal{D}$ with corresponding universe of participants Ω and for all PPT adversaries $\mathcal{A}$ using at most t online sessions there exists a negligible function $\varepsilon(\cdot)$ such that:*

$$\mathsf{Adv}_{\Pi,\mathcal{A}}^{\mathsf{AKE\text{-}ICM}}(\lambda) = \left| \Pr[\mathsf{Exp}_{\Pi,\mathcal{A}}^{\mathsf{AKE\text{-}ICM}}(\lambda) = 1] - \frac{1}{2} \right| \leq \frac{\mathcal{O}(t)}{|\mathcal{D}|} + \varepsilon(\lambda).$$

$\mathsf{Exp}_{\Pi,\mathcal{A}}^{\mathsf{AKE}}(\lambda) : c = 1; b \in_R \{0,1\}; \forall (P, P') \in \mathcal{C} \times \mathcal{S}$ *choose* $\mathrm{pw}_{P,P'}$;
$b' \leftarrow \mathcal{A}^{\mathsf{Send}_b, \mathsf{Execute}_b, \mathsf{Corrupt}, \mathsf{Test}_b}(\lambda, c)$; *return* $b = b'$.

The above requirement is stronger than AKE-security. In particular, it cannot be satisfied by PAKE protocols where client messages depend on those of the server or where client messages sent in later rounds depend on client messages that were sent in previous rounds. Nonetheless, there exist efficient AKE-secure

PAKE protocols with indistinguishable client messages as discussed in Sect. 3.5. In particular, for AKE-secure *one-round* PAKE protocols, where the client can send its message independently of the server's message the indistinguishability property can be argued based on the uniformity of the client's message in the message space.

3.2 The O-PAKE Compiler

Our compiler takes as input a PAKE protocol Π and outputs its O-PAKE version, denoted C_Π. The compiled protocol C_Π follows Definition 1 and consists of the two algorithms $C_\Pi.\texttt{init}$ and $C_\Pi.\texttt{next}$. For the passwords in $\mathbf{pw}$ used as input to $C_\Pi.\texttt{init}$ we assume that each $\mathbf{pw}[i] = (\text{ix}, \pi) \in \mathbb{F} \times \mathcal{D}_\Pi$, where ix denotes an *index* and π the corresponding *password* for the underlying PAKE protocol Π, whereby the distributions of ix and π are independent and no two pairs $(\text{ix}_1, \pi_1), (\text{ix}_2, \pi_2) \in \mathbf{pw}$ have $\text{ix}_1 = \text{ix}_2$. For each PAKE round r the compiler uses a corresponding instance $\texttt{IHME}^r$ with message space $\mathcal{M}^{\texttt{IHME}^r}$ and a compatible admissible encoding $F^r : \mathcal{M}^{\texttt{IHME}^r} \rightarrow \mathcal{M}_C^r$ where $\mathcal{M}_C^r$ is the space of clients messages of Π in that round. In the following we assume that the underlying $\Pi.\texttt{next}$ algorithm outputs messages that can be seen as one element and thus can be processed using one instance $(F^r, \texttt{IHME}^r)$ in each round. Note that this allows for a more comprehensible description and is not a restriction of the O-PAKE compiler. We discuss the case of multi-set messages $\mathcal{M}_C^r = \mathcal{M}_C^{r,1} \times \cdots \times \mathcal{M}_C^{r,l}$ that will require composition of up to l instances of encoding schemes per round in Sect. 3.6.

The $C_\Pi.\texttt{next}$ algorithm on the client side computes corresponding PAKE round messages for all passwords in $\mathbf{pw}$ using the original $\Pi.\texttt{next}$ algorithm and encodes them with $\mathcal{I}_{F^r}$ and $\texttt{IHME}^r.\texttt{iEncode}$ prior to transmission to the server. On the server side $C_\Pi.\texttt{next}$ decodes the incoming PAKE message using F^r and $\texttt{IHME}^r.\texttt{iDecode}$ (using its input $\mathbf{pw}[i].\text{ix}$ as index) and replies with the message output by $\Pi.\texttt{next}$. Note that the server only decodes messages but never encodes them. If $pw \in \mathbf{pw}$ then at the end of its n PAKE sessions the client will hold n intermediate PAKE keys, whereas the server holds only one such key. The additional key confirmation and key derivation steps allow the client to determine which of its n PAKE session keys matches the one held by the server, in which case both participants will derive the same session key. In the following we describe the two algorithms $C_\Pi.\texttt{init}$ and $C_\Pi.\texttt{next}$ more in detail.

Algorithm $\mathbf{C_\Pi}.\texttt{init}$ The algorithm makes n calls to $\Pi.\texttt{init}$, one for each password $\mathbf{pw}[i].\pi$, to generate corresponding $\texttt{state}$ for each of the n PAKE sessions that are stored in $\mathbf{state}_P^i$. An ith session of Π run by the client using the corresponding password $\mathbf{pw}[i].\pi$ is denoted by $\Pi[i]$. The partner id $\texttt{pid}_P^i$ is set to P' and the instance P_i with the given $\texttt{role}$ and a vector of n local states in $\mathbf{state}_P^i$ is established. We require that no two passwords in $\mathbf{pw}$ are identical, which is necessary to ensure the correctness of the IHME step. Note that if $\texttt{role} = \texttt{server}$ then $n = 1$, i.e. servers run only one PAKE session (Fig.1).

a $C_{\Pi}.\texttt{next}(m_{\text{in}})$ — Client

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$
  $E = \emptyset$; $m_{\text{out}} = \emptyset$
  for $i = 1 \ldots n$ **do**
    if $\Pi[i]$ has not finished **then**
      $(m'_{\text{out}}, \Pi[i].\texttt{k}) \leftarrow \Pi[i].\texttt{next}(m_{\text{in}})$
      if $m'_{\text{out}} \neq \emptyset$ **then**
        $E = E \cup \{(\mathbf{pw}[i].\text{ix}, \mathcal{I}_{F^r}(m'_{\text{out}}))\}$
    else if $\Pi[i].\texttt{k} \in \mathcal{K}_{\Pi}$ and
        $m_{\text{in}} = \texttt{PRF}_{\Pi[i].\texttt{k}}(\texttt{sid}^i_P || P_i || \texttt{pid}^i_P || 0)$ **then**
      $\texttt{k} = \texttt{PRF}_{\Pi[i].\texttt{k}}(\texttt{sid}^i_P || P_i || \texttt{pid}^i_P || 1)$
    else
      $\texttt{k} = \perp$
  if $E \neq \emptyset$ **then**
    $m_{\text{out}} = \texttt{IHME}^r.\texttt{iEncode}(E)$
  return $(m_{\text{out}}, \texttt{k})$

b $C_{\Pi}.\texttt{next}(m_{\text{in}})$ — Server

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$
  $m_{\text{out}} = \emptyset$
  if Π has not finished **then**
    $m \leftarrow \texttt{IHME}^r.\texttt{iDecode}(\text{pw.ix}, m_{\text{in}})$
    $m' = F^r(m)$
    $(m_{\text{out}}, \Pi.\texttt{k}) \leftarrow \Pi.\texttt{next}(m')$

  if $\Pi.\texttt{k} \in \mathcal{K}_{\Pi}$ **then**
    $m_{\text{out}} = \texttt{PRF}_{\Pi.\texttt{k}}(\texttt{sid}^j_{P'} || \texttt{pid}^j_{P'} || P'_j || 0)$
    $\texttt{k} = \texttt{PRF}_{\Pi.\texttt{k}}(\texttt{sid}^j_{P'} || \texttt{pid}^j_{P'} || P'_j || 1)$
  else
    $\texttt{k} = \perp$

  return $(m_{\text{out}}, \texttt{k})$

Fig. 1. $C_{\Pi}.\texttt{next}$ algorithms

Algorithm $\mathbf{C}_{\Pi}.\texttt{next}$ We distinguish between $C_{\Pi}.\texttt{next}$ specifications for clients (Algorithm 1a) and servers (Algorithm 1b) as they are significantly different. We write $\Pi[i].\texttt{next}$ for the invocation of $\Pi.\texttt{next}$ for the ith session of Π run by the client using $\mathbf{pw}[i].\pi$. On the client side $C_{\Pi}.\texttt{next}$ computes messages m'_{out} for all running PAKE sessions and encodes them. The server decodes the incoming IHME structure and computes its response using $\Pi.\texttt{next}$. If any PAKE session $\Pi[i]$ at the client has finished with $\Pi[i].\texttt{k} \in \mathcal{K}_{\Pi}$ then the client expects a valid confirmation message from the server prior to derivation of the resulting session key $\texttt{k}$ with $\texttt{PRF}$ using $\Pi[i].\texttt{k}$. An invalid confirmation message implies that $\texttt{k}$ is set to $\perp$. This confirmation message is generated on the server side using $\texttt{PRF}$ only if and immediately after $\Pi.\texttt{next}$ outputs $\Pi[i].\texttt{k} \in \mathcal{K}_{\Pi}$; in which case a valid resulting session key $\texttt{k}$ is also derived. If, however, Π finishes with $\Pi[i].\texttt{k} = \perp$ then $\texttt{k}$ will also be set to $\perp$.

3.3 Relation to LAKE

A Language Authenticated Key Exchange (LAKE) protocol, proposed by Benhamouda et al. in [10], authenticates two parties, client C and server S holding each a word in an algebraic languages. In particular, let $R : \{0,1\}^* \times P \times W \rightarrow \{0,1\}$ denote a relation and $L_R(\mathsf{pub}, \mathsf{priv}) \subseteq W$ a language with $\mathsf{pub} \in \{0,1\}^*$ and $\mathsf{priv} \in P$. A word $w \in W$ is in the language L_R iff $R(\mathsf{pub}, \mathsf{priv}, w) = 1$. The client holds a word w_c for relation R_C and the server holds a word w_s for relation R_S. They agree on public parameters pub, exchange ephemeral public keys, and *think* of a value priv'_C, resp. priv'_S, they expect to be used by the other party. To instantiate the LAKE framework it is necessary to specify client and server languages and according commitments with associated smooth projective hash functions (SPHF) [17]. We briefly recall how to instantiate LAKE with passwords from [10, Sect. 6.2], i.e. how to build PAKE protocols in the LAKE framework. The languages are defined as $L_C = \{w_c\}$ for the client and $L_S = \{w_s\}$ for the

server, such that $\mathsf{priv}'_C = \mathsf{priv}'_S = w_c = w_s$ is the password and the relations are $R_C = R_S = (\emptyset, \mathsf{priv}, w) = 1 \iff \mathsf{priv} = w$, i.e. equality test for the password.

To instantiate O-PAKE in LAKE we define client relation $R_C(\emptyset,\mathrm{pw}', \mathbf{pw}) = 1 \iff \mathrm{pw}' \in \mathbf{pw}$ and server relation $R_S(\emptyset, \mathrm{pw}', \mathrm{pw}) = 1 \iff \mathrm{pw}' = \mathrm{pw}$. While the server relation stays the same as in PAKE, the client language $L_{R_C}(\emptyset, \mathrm{pw}') \subseteq \{\mathrm{pw}_1, \ldots, \mathrm{pw}_n\} = \mathbf{pw}$ uses a relation that takes a set of passwords $\mathbf{pw}$ and an *expected* password pw' as input, and is fulfilled iff $\mathrm{pw}' \in \mathbf{pw}$. Following [10, Fig. 4] we realise O-PAKE in the LAKE framework as follows: First, the client (initiator) generates a multiDLCSCom′ commitment (C_C, C'_C) on word w_c, i.e. a multi-commitment to all passwords $\mathrm{pw} \in (\mathrm{pw}_1, \ldots, \mathrm{pw}_n)$, as well as a Pedersen commitment C''_C on C'_C, and sends (C_C, C''_C) to the server S. The server replies with $(C_S, \boldsymbol{\varepsilon}, \mathtt{k}_{\mathtt{p}S}, \sigma_S)$, computed as follows: C_S is a multi-LCS commitment on $w_s = \mathrm{pw}_S$; $\boldsymbol{\varepsilon}$ is a challenge vector on C_C of length n; $\mathtt{k}_{\mathtt{p}S}$ is a projection key for a suitable SPHF for C_C; and σ_S is a signature on all flows. In the final round, the client checks σ_S before returning $(C'_C, t, \mathtt{k}_{\mathtt{p}C}, \sigma_C)$ to the server, which is computed as follows: (C'_C, t) is the decommitment to C''_C, where t is the used randomness; $\mathtt{k}_{\mathtt{p}C}$ is a projection key for a suitable SPHF for C_S; and σ_C is a signature on all flows. After checking all signatures and commitments, session keys are computed as multiplication of projection and hash function on $\mathsf{Com}_C = C_C \cdot C'^{\boldsymbol{\varepsilon}}_C$ and $\mathsf{Com}_S = C_S$.

So while it seems possible to instantiate O-PAKE in the LAKE framework (after specifying necessary primitives), the construction is rather inefficient. In particular, an instantiation of O-PAKE in LAKE needs four rounds, our O-PAKE compiler adds only one round to the round-complexity of the underlying PAKE, i.e. can be instantiated with three rounds. Further, server-complexity is linear in the number of client-passwords n. This stems from the observation that the projection key $\mathtt{k}_{\mathtt{p}S}$, as well as the computation of the hash function, requires a linear number of public key operations, e.g., exponentiations, in n. Performance of O-PAKE instantiated in the LAKE framework is therefore not more efficient than the naïve construction, and in particular does not fulfil our requirement of nearly constant server performance.

3.4 Security Analysis

AKE-security of the protocol generated with the O-PAKE compiler is established in Theorem 1.

Theorem 1. *If Π is an R-round AKE-secure PAKE protocol with indistinguishable client messages in $\mathcal{M}^r_C$ for $r \in [1, \ldots, R]$, $F^r : \mathcal{M}^{\mathtt{IHME}^r} \to \mathcal{M}^r_C$ is an ϵ-admissible encoding, and $\mathtt{IHME}^r$ is an index-hiding message encoding, then C_Π is an $R+1$-round AKE-secure O-PAKE protocol.*

Proof (sketch). The proof uses a sequence of experiments Exp_i, $i = 1, \ldots 4$, where each Exp_i is based on a small modification of Exp_{i-1}. At a high level we first replace real client messages and session keys of underlying PAKE sessions $\Pi[i]$ with random messages and keys while ensuring the consistency against

an adversary that corrupted passwords and then mounts online attacks. This modification remains unnoticeable to $\mathcal{A}$ if the underlying PAKE protocol Π is AKE-secure with indistinguishable client messages. Then, we replace the outputs of the inverse transformations $\mathcal{I}_{F^r}$ applied in each round r by choosing random elements from the corresponding round's $\texttt{IHME}^r$ message space and show that this remains unnoticeable assuming that F^r is an ϵ_{F^r}-admissible encoding F^r. Then, we replace each real password index in the $\texttt{IHME}^r$ encoding process with a random password and show that this remains unnoticeable due to the index-hiding property of each $\texttt{IHME}^r$ scheme. Finally, we modify the computation of the server's confirmation message and of the session keys that are returned in Test_b queries by using random elements from the corresponding spaces. This remains unnoticeable due to the pseudorandomness of the $\texttt{PRF}$ function that is used to derive their values. We refer to the full version [29] of this work for the full proof due to space limitations.

3.5 Oblivious PAKE Instantiation

An AKE-secure PAKE protocol Π is suitable for our O-PAKE transformation if it is also AKE-ICM-secure and there exist admissible encodings to map those messages into the message space of the IHME scheme. In the following we list four sets R with suitable admissible encodings. Thus, any AKE-secure PAKE protocol whose client messages contain components from these four sets can be transformed into an O-PAKE protocol using our compiler.

Definition 4 (Admissible Encodings for Client Messages). *An admissible encoding $F : \{0,1\}^{\ell(\lambda)} \to R$ with polynomial $\ell(\lambda)$ exists for any of the following four sets:*

(1) Set $R = \{0, \ldots, N-1\} = \mathbb{Z}_N$ of natural numbers, for arbitrary $N \in \mathbb{N}$. (cf. [19, Lemma 13])
(2) The set of quadratic residues modulo safe primes p, i.e. $R = QR(p) \subseteq \mathbb{Z}_p^\times$. (cf. [19, Lemma 13])
(3) Arbitrary subgroups $G \subseteq \mathbb{Z}_p^\times$ of prime order q. (cf. [19, Lemma 13])
(4) The set $R = E(\mathbb{F})$ of rational points on (certain) elliptic curves, defined over a finite field (cf. [15]).

Computing Indices. We require that password pw used in O-PAKE consists of two independent components ix and π. For instance, it is sufficient for the user to choose $\pi \in_R \mathcal{D}$ and compute the index $\text{pw.ix} = f(\rho, \text{pw}.\pi)$ using some fresh randomness ρ and a function f with output independent from π, i.e. the probability that π was used as input to f to produce ix must remain $1/|\mathcal{D}|$. Note that this approach requires a pre-flow to the protocol to exchange randomness ρ, which can however be easily integrated into the overall login process. Furthermore, it is crucial that randomness ρ is fresh for every execution of the protocol as any reuse of ρ would offers an attacker the possibility to distinguish between real and simulated O-PAKE messages.

Remark 1. Verifier-based PAKE (VPAKE) protocols, such as [9,12,24,25], where only some password-dependent verification information (e.g. a randomised password hash with a random salt) is stored on the server side are not formally considered in this work. Nonetheless, the techniques underlying our O-PAKE compiler seem also applicable to VPAKE protocols as long as their messages satisfy the identified AKE-ICM requirement.

3.6 Processing Multi-Component Messages

In the following we describe how the compiler can handle PAKE protocol messages consisting of multiple elements, possibly from different sets. We observe that any such PAKE message can be seen as an element of a combined message space that is formed through a Cartesian product of those sets and distinguish between two types of message components, namely components that represent constants and components that depend on passwords, including integer values and group elements. Since constants are password-independent they do not need to be processed by the compiler and can be communicated directly. All other message components have to be encoded according to the compiler specification. In order to encode those components we use ν-fold IHME introduced in [31], which allows to encode a list of ν message components from the same finite field. The compiler splits message components from different finite fields into corresponding classes and applies appropriate IHME encoding to each class separately in order to compute the corresponding IHME structure. The IHME structures for all message components are then concatenated and treated as a single compiler message. This processing of multi-component messages requires existence of admissible encodings and index-hiding message encodings for each component class m_j of m. In order to process the components, a loop over $m_1, \ldots, m_l$ adds $(\mathbf{pw}[i], \mathcal{I}_{F^{r,j}}(m_j))$ to the input set of $\nu\text{−fold−IHME}_j^r.\texttt{iEncode}$ according to their classes (e.g. finite fields). Likewise, the output message m_{out} of the `next` algorithm is the concatenation of the encoded component classes. Upon receiving a client message m_{in}, the server has to decompose it to retrieve the IHME encoded messages. After decoding the message parts with $m_j \leftarrow \nu\text{−fold−}\texttt{IHME}_j^r.\texttt{iDecode}(\text{pw}_{P,P'}, m_{\text{in}}^j)$ the original PAKE message of Π is reassembled by decoding messages $F^{r,j}(m_j)$.

Adopting this approach for multi-component messages, the AKE-security remains preserved. This is due to the following observation about the proof of Theorem 1: in the game-hopping sequence the adversary will be provided with l IHME encoded messages (one for each message element class that requires encoding). The corresponding index-hiding advantage will therefore be multiplied by l. The remaining parts of the proof remain as is.

4 Concrete Instantiation Examples

In this section we give concrete instantiation of the O-PAKE compiler, using the random-oracle based SPAKE protocol by Abdalla and Pointcheval [5]. A second

instantiation using the common-reference-string-model protocol from Katz and Vaikuntanathan [28] can be found in the full version [29].

4.1 Oblivious SPAKE

We demonstrate how the compiler can be applied to PAKE protocols using the AKE-secure, random-oracle-based SPAKE protocol from [5]. The resulting O-SPAKE is specified in Fig. 2 and involves steps of the original SPAKE protocol from [5, Sect. 5], which is a secure variant of [8], whose security has been proven in the random oracle model.[1] SPAKE uses a prime-order cyclic group G for which the Computational Diffie-Hellman (CDH) problem is assumed to be hard. The shared SPAKE password pw is chosen from $\mathbb{Z}_q$. Let $M, N \in G$ denote two public group elements. The protocol proceeds in one round, where the client sends $X^* \leftarrow g^x \cdot M^{\mathrm{pw}\cdot\pi}, x \in_R \mathbb{Z}_q$ and the server responds with $Y^* \leftarrow g^y \cdot N^{\mathrm{pw}\cdot\pi}, y \in_R \mathbb{Z}_q$. The actual order of these messages does not matter since they are independent. The algorithm `next` computes an intermediate value s and derives the session key as $\Pi.\mathtt{k} \leftarrow H(P, P', X^*, Y^*, \mathrm{pw}, s)$. We refer to the original work [5, Sect. 5] for more details on SPAKE. The SPAKE protocol is a suitable input PAKE protocol for our O-PAKE compiler since it can be instantiated using subgroups $G \subseteq \mathbb{Z}_p^\times$ of prime order q in which the CDH problem is believed to be hard. We can apply the admissible encodings (3) from [19, Lemma 13] due to the fact that client's SPAKE message $X^* = g^x \cdot M^{\mathrm{pw}\cdot\pi}$ is uniformly distributed in G, given the uniformity of $x \in \mathbb{Z}_q$. We formalise this by showing that SPAKE fulfils our definition of AKE-ICM, before defining suitable admissible encodings, which concludes the instantiation of O-SPAKE.

Lemma 2 (SPAKE is AKE-ICM Secure). *The SPAKE protocol from [5, Sect. 5] is AKE-ICM secure.*

Proof. The initial experiment in the proof for SPAKE security in [5] corresponds to the AKE-ICM experiment with $b = 1$. In the following we show that the proof in [5, Appendix C] can be modified without changing the adversaries advantage such that the final experiment is equal to the AKE-ICM experiment with $b = 0$, which concludes the proof. We first change experiment one by additionally simulating the Corrupt oracle and using a global bit b in simulating the Test oracle. This does not change the adversary's success probability. The second experiment, aborting on hash collisions, stays unchanged. In the following two experiments we have to make sure that the adversary does not win trivially by returning the correct key to Test queries on corrupted sessions and only modify oracle replies to uncorrupted sessions. While the original proof only changes the calculation of the session key in passive sessions to a random element in experiment three, we also change client messages produced in Execute queries to random elements. Note that this is implicitly already done in the original proof. However, we formalise it here again and change experiment three as follows: Invocations of the

[1] Note that the very similar SOKE protocol from [1] can also be used in the O-PAKE compiler following the here given description of O-SPAKE.

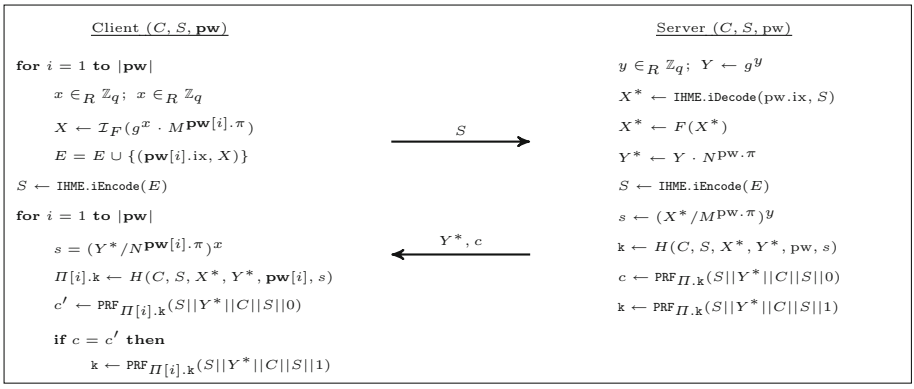

Fig. 2. Oblivious SPAKE (O-SPAKE) public input: $G, g, p, q, M, N, H, \mathtt{IHME}, F$

Execute oracle on uncorrupted parties are answered with uniformly at random chosen messages, i.e. $X^* = Ag^x$ and $Y^* = Bg^y$ with $x, y \in_R \mathbb{Z}_p$, for some DH instance (A, B). Experiment three corresponds now to the AKE-ICM experiment with Send_1, Test_b and $\mathsf{Execute}_0$. The lemma follows by noting that after our modifications of experiment three the last experiment of the AKE-security proof of SPAKE in [5, Appendix C] is equivalent to the AKE-ICM experiment with $\mathsf{Execute}_0$, Send_0 and Test_0, i.e. the adversary only wins by guessing the correct password. □

Admissible Encodings for SPAKE We use admissible encodings (1) and (3) from [19, Lemma 13] to encode SPAKE client messages. To implement the inverse encoding of (1) $:= \mathcal{I}_{F^{(1)}} : \mathbb{Z}_N \to \{0,1\}^{\ell(\lambda)}$ we use the inverse of encoding (3) $:= \mathcal{I}_{F^{(1)}} : G \to \mathbb{Z}_p^\times$. This results in a combined inverse encoding of $\mathcal{I}_{F^{(3,1)}} : G \to \mathbb{Z}_p^\times \to \{0,1\}^{\ell(\lambda)}$ with $\ell(\lambda) > 2|N|$ and $p = N$. Implementation of $F^{(3,1)} : \mathbb{Z}_{q'} \to G$ and $\mathcal{I}_{F^{(3,1)}}$ follows the specification from [19, Lemma 12] with prime $|q'| = \ell(\lambda) > 2|N|$ to meet IHME requirements.

5 Conclusion

In this paper we addressed the problem of handling multiple password trials efficiently within the execution of PAKE protocols; in particular, aiming to optimise the amount of work on the server side. The proposed O-PAKE compiler results in almost constant computational complexity for the server without significantly increasing the computation costs on the client side, yet preserving all security guarantees offered by standard PAKE protocols. It can be used with PAKE protocols that fulfil our new definition of AKE-ICM security and whose client messages can be encoded through a suitable admissible encoding scheme. The security of the compiler has been proven under standard assumptions in an extension of the widely used PAKE model from [4] and exemplified on the PAKE protocol and [28].

References

1. Abdalla, M., Bresson, E., Chevassut, O., Möller, B., Pointcheval, D.: Provably secure password-based authentication in TLS. In: ASIACCS 2006, pp. 35–45. ACM (2006)
2. Abdalla, M., Catalano, D., Chevalier, C., Pointcheval, D.: Efficient two-party password-based key exchange protocols in the UC framework. In: Malkin, T. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 335–351. Springer, Heidelberg (2008)
3. Abdalla, M., Chevassut, O., Fouque, P.-A., Pointcheval, D.: A simple threshold authenticated key exchange from short secrets. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 566–584. Springer, Heidelberg (2005)
4. Abdalla, M., Fouque, P.-A., Pointcheval, D.: Password-based authenticated key exchange in the three-party setting. In: Vaudenay, S. (ed.) PKC 2005. LNCS, vol. 3386, pp. 65–84. Springer, Heidelberg (2005)
5. Abdalla, M., Pointcheval, D.: Simple password-based encrypted key exchange protocols. In: Menezes, A. (ed.) CT-RSA 2005. LNCS, vol. 3376, pp. 191–208. Springer, Heidelberg (2005)
6. Bellare, M., Pointcheval, D., Rogaway, P.: Authenticated key exchange secure against dictionary attacks. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 139–155. Springer, Heidelberg (2000)
7. Bellare, M., Rogaway, P.: Entity authentication and key distribution. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773, pp. 232–249. Springer, Heidelberg (1994)
8. 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)
9. Bellovin, S.M., Merritt, M.: Augmented encrypted key exchange: a password-based protocol secure against dictionary attacks and password file compromise. In: CCS 1993, pp. 244–250. ACM (1993)
10. Ben Hamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: Efficient UC-secure authenticated key-exchange for algebraic languages. In: Kurosawa, K., Hanaoka, G. (eds.) PKC 2013. LNCS, vol. 7778, pp. 272–291. Springer, Heidelberg (2013)
11. Benhamouda, F., Blazy, O., Chevalier, C., Pointcheval, D., Vergnaud, D.: New techniques for SPHFs and efficient one-round PAKE protocols. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 449–475. Springer, Heidelberg (2013)
12. Benhamouda, F., Pointcheval, D.: Verifier-based password-authenticated key exchange: new models and constructions. Cryptology ePrint Archive, report 2013/833 (2013). http://eprint.iacr.org/2013/833
13. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 213–229. Springer, Heidelberg (2001)
14. Boyko, V., MacKenzie, P.D., Patel, S.: Provably secure password-authenticated key exchange using Diffie-Hellman. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 156–171. Springer, Heidelberg (2000)
15. Brier, E., Coron, J.-S., Icart, T., Madore, D., Randriam, H., Tibouchi, M.: Efficient indifferentiable hashing into ordinary elliptic curves. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 237–254. Springer, Heidelberg (2010)
16. Canetti, R., Halevi, S., Katz, J., Lindell, Y., MacKenzie, P.: Universally composable password-based key exchange. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 404–421. Springer, Heidelberg (2005)

17. Cramer, R., Shoup, V.: Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 45–64. Springer, Heidelberg (2002)
18. Dierks, T., Rescorla, E.: RFC 5246 - the transport layer security (TLS) protocol version 1.2, August 2008. Updated by RFCs 5746, 5878, 6176
19. Fleischhacker, N., Günther, F., Kiefer, F., Manulis, M., Poettering, B.: Pseudorandom signatures. In: ASIA CCS 2013, pp. 107–118. ACM (2013)
20. Florencio, D., Herley, C.: A large-scale study of web password habits. In: 16th International Conference on World Wide Web, WWW 2007, pp. 657–666. ACM (2007)
21. Gaw, S., Felten, E.W.: Password management strategies for online accounts. In: Symposium on Usable Privacy and Security, SOUPS 2006, pp. 44–55. ACM (2006)
22. Gennaro, R.: Faster and shorter password-authenticated key exchange. In: Canetti, R. (ed.) TCC 2008. LNCS, vol. 4948, pp. 589–606. Springer, Heidelberg (2008)
23. Gennaro, R., Lindell, Y.: A framework for password-based authenticated key exchange. ACM Trans. Inf. Syst. Secur. **9**(2), 181–234 (2006)
24. Gentry, C., MacKenzie, P.D., Ramzan, Z.: A method for making password-based key exchange resilient to server compromise. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 142–159. Springer, Heidelberg (2006)
25. Jablon, D.P.: Extended password key exchange protocols immune to dictionary attacks. In: WETICE, pp. 248–255. IEEE Computer Society (1997)
26. Katz, J., Ostrovsky, R., Yung, M.: Efficient and secure authenticated key exchange using weak passwords. J. ACM **57**(1), 3:1–3:39 (2009)
27. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 293–310. Springer, Heidelberg (2011)
28. Katz, J., Vaikuntanathan, V.: Round-optimal password-based authenticated key exchange. J. Cryptology **26**(4), 714–743 (2013)
29. Kiefer, F., Manulis, M.: Oblivious pake: efficient handling of password trials. Cryptology ePrint Archive, report 2013/127 (2013). http://eprint.iacr.org/2013/127
30. Manulis, M., Pinkas, B., Poettering, B.: Privacy-preserving group discovery with linear complexity. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 420–437. Springer, Heidelberg (2010)
31. Manulis, M., Poettering, B.: Practical affiliation-hiding authentication from improved polynomial interpolation. In: ASIACCS 2011, pp. 286–295. ACM (2011)