Oblivious PAKE and Efficient Handling of Password Trials

Franziskus Kiefer and Mark Manulis

Department of Computing, University of Surrey, United Kingdom
f.kiefer@surrey.ac.uk, mark@manulis.eu

Abstract. An often neglected problem for potential practical adoption of Password-based Authenticated Key Exchange (PAKE) protocols on the Internet is the handling of failed password trials. Unlike the currently used approach, where a server-authenticated TLS channel (involving constant number of public key-based operations on both sides) is set up once and can then be used by the client to try a limited number of passwords essentially for free, any new password trial using PAKE would result in the repetition of the entire protocol. With existing PAKE protocols, the minimum number of public key-based operations on both sides is thus lower-bounded by $O(n)$, where n is the number of trials. This bound is optimal for the client (that tries n passwords in the worst case) but is clearly not optimal for the server, which uses the same reference password of the client in each trial. This paper presents a secure and practical approach for achieving the lower bound of $O(1)$ public key operations on the server side.

To this end, we introduce *Oblivious PAKE* (O-PAKE), a general compiler for a large class of PAKE protocols, allowing a client that shares one password with a server to use a *set of passwords* within one PAKE session, which succeeds if and only if one of those input passwords matches the one stored on the server side. The term "oblivious" is used to emphasize that no information about non-matching passwords input by the client is made available to the server, which contrasts for instance to the aforementioned TLS-based approach, where any tried password is disclosed to the server. The $O(1)$ bound on the server side is obtained in our O-PAKE compiler using special processing techniques for the messages of the input PAKE protocol. We prove security of the O-PAKE compiler under standard assumptions using the latest variant of the popular game-based PAKE model by Bellare, Rogaway, and Pointcheval (Eurocrypt 2000). We identify the requirements that PAKE protocols must satisfy in order to suit the compiler and give two concrete O-PAKE protocols based on existing PAKE schemes. Both protocols are implemented and the analysis of their performance attests to the practicality of the compiler.

The use of O-PAKE further eliminates another practical problem with password-based authentication on the Web in that users no longer need to remember the actual association between their frequently used passwords and corresponding servers and can try several of them in one execution without revealing the entire set to the server.

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 (typically TLS [23]). This approach requires PKI and its security relies on the client's ability to correctly verify server's certificate. Any impersonation of the certificate leads to password exposure and even if no impersonation takes place then any password input on the client side is still revealed to the server. This creates a different problem based on the existing statistics that many users operate with a small set of passwords but often do not remember their correct matching 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 study in [25] shows that on average every user has 6.5 passwords, which are used on 25 different websites. Another study in [26] shows that 2.4 failed login attempts are needed on average until the user types in the correct password. These numbers suggest that in the worst case roughly 2 passwords from the client's set could potentially be revealed through failed login attempts to the server within a single TLS session — a significant threat for the client. In order to minimize

the risk of online dictionary attacks aiming to impersonate the client, servers typically limit the number of failed trials. Yet, the amount of work for processing failed attempts on the server side is negligible since all tried passwords are transmitted through the same TLS channel.

The notion of *Password-based Authenticated Key Exchange (PAKE)*, introduced in [12], initially formalized in [10, 15], and later explored in numerous further works (cf. Section 1.2), is considered today as a more secure alternative to the above approach. The standard model of PAKE does not require any PKI and assumes that only one password is shared between the both parties. Considering that passwords have low entropy, PAKE protocols 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. The only way for current PAKE protocols to deal with failed sessions is to repeat the 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, i.e. $O(n)$ where n is the number of attempted PAKE sessions, which is a clear disadvantage in comparison to the TLS-based approach and could be seen as a reason for the limited progress on the adoption of PAKE on the Internet (in addition to some other unrelated issues such as browser incompatibility, patent considerations, and the lack of adopted standards).

Handling multiple password trials with PAKE, without experiencing linear increase in the amount of public key operations for the server, is a non-trivial problem. The problem seems to be avoidable if in a single PAKE execution the client can use several passwords, without exposing them to offline dictionary attacks and to the (possibly malicious) server, which in turn would use only one password that is 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 build out of n independent (possibly parallelized) runs of some secure PAKE protocol, where the client uses a different password in each session 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 to $O(1)$.

In addition, such PAKE protocol still needs to fulfill some basic requirements. For instance, it needs to address 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. Note that for the server there is no difference whether a client is given the opportunity to perform at most n independent PAKE sessions (password trials) or only one session but with verifiably at most n input passwords. Due to the asymmetry in the number of passwords used by the client and server, i.e. if $n > 1$, the protocol must also prevent a possibly malicious server from mounting an online dictionary attack against more than one password with respect to the client's input set of passwords, as this is the best what also traditional PAKE protocols can provide.

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 suitable PAKE protocols in a black-box way into what we call an *Oblivious PAKE* (O-PAKE). The functionality of O-PAKE protocols resembles that of PAKE except that the client inputs a set $\mathbf{pw}$ of n passwords from some dictionary $\mathcal{D}$ while the input of the server is limited to one password pw. Note that this does *not* increase the probability for offline dictionary attacks as the server still limits the number of trials and the maximum number of passwords input by the client. The protocol execution of O-PAKE succeeds if and only if $\mathrm{pw} \in \mathbf{pw}$. We utilise the well-known (game-based) PAKE model by Bellare, Pointcheval, and Rogaway [10] in its (stronger) Real-or-Random flavor from [6] and update it to match the O-PAKE setting by allowing $\mathbf{pw}$ as input on the client side. Thereby, we also assume uniformly at random distributed passwords. A client using passwords with different probabilities in one O-PAKE session obviously lowers the security of O-PAKE to the least secure password distribution as the server has to break only one of the client's passwords (cf. Section 2).

The crucial idea behind our transformation is to let each client execute n sessions of the input AKE-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 messages of the PAKE session that was computed by the client using

the correct password pw $\in$ **pw** while preserving security against offline dictionary attacks, including with respect to other passwords in the client's set. This is the most trickiest part of the compiler since intuitively if the server can recover the messages of the "right" PAKE session it can answer them according to the specification of the PAKE protocol and by repeating this for all PAKE rounds both parties will be able to successfully accomplish the protocol. If identification of the "right" 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 minimal runtime of the client on the other side is linear in the number 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.[1]

Our solution for the identification of the "right" PAKE session is a careful composition of two techniques that were introduced in a different context. Our first building block is a special message encoding technique that was suggested by Manulis, Pinkas, and Poettering [36, 37] in the context of affiliation-hiding authentication protocols (to solve there an open problem of group discovery from [30]). Their *Index-Hiding Message Encoding (IHME)* technique 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 at the encoding time and therefore all encoded messages must contain enough entropy to prevent dictionary attacks over the index space. The intuition behind our O-PAKE transformation is to encode messages of n PAKE sessions on the client side using corresponding passwords from **pw** as indices and let the server use its input password pw in the decoding procedure. However, adopting the IHME technique to the O-PAKE setting requires some work around, since the only known IHME construction treats index-message pairs as roots of a polynomial of a finite field $\mathbb{F}$ and the coefficients of the interpolated polynomial as the corresponding IHME structure. Therefore, in order to encode PAKE messages with IHME the former need to be first represented as elements of $\mathbb{F}$ in a way that would prevent offline dictionary attacks, i.e. the resulting field elements must remain indistinguishable from random elements, yet it should be possible to revert them into the original PAKE messages after the IHME decoding procedure. We show how to realise this intermediate step by adopting the generalised version of *admissible encodings* [14, 16, 24] that were initially introduced in the context of identity-based encryption as a method of hashing into groups of points $E(\mathbb{F})$ over pairing-friendly elliptic curves. By generalizing this concept to the message spaces of PAKE, we can plug in the corresponding admissible encoding for PAKE messages prior to encoding the resulting password-message pairs with IHME. Admissible encodings (cf. Section 3.2) exist for messages from rings $\mathbb{Z}_N$, groups of quadratic residues $QR(p) \subset \mathbb{Z}_p^\times$, prime-order subgroups of $\mathbb{G}_q \subset E(\mathbb{F})$ for certain elliptic curves E over a finite field $\mathbb{F}$, and one that can be applied to prime order groups $\mathbb{G}_q \subset \mathbb{Z}_p^\times$ for primes $p = \alpha q + 1$. In order to judge whether an AKE-secure PAKE protocol is suitable for our generic O-PAKE transformation it is sufficient to check the existence of an appropriate admissible encoding for its round messages.

To exemplify instantiations of O-PAKE we implement the generic compiler and use it on two suitable PAKE protocols, namely the AKE-secure (standard model, CRS) PAKE framework of Gennaro [27] and the (random-oracle based) SPAKE protocol from [7]. Based on real-world parameters from the aforementioned studies in [25, 26] we analyse the actual performance of our implementation and confront it with the naïve approach of repeating the PAKE protocol n times.

1.2 Related Work

The concept of PAKE was introduced by Bellovin [12] and corresponding security models for PAKE were initially developed by Bellare, Pointcheval, and Rogaway [10], Boyko, MacKenzie, and Patel [15], and Goldreich and Lindell [29]. The Find-then-Guess PAKE model from [10], strengthened and simplified in [6] towards the Real-or-Random PAKE model aims at Authenticated Key Exchange (AKE)-security, the main security property of PAKE protocols. The models in [6, 10] remain the most popular game-based PAKE models, adopted in the analysis of many protocols, including the random oracle-based protocols [2, 5] and protocols requiring a common reference string (CRS) [27, 28, 33]. The only (but fairly inefficient) protocol that is

[1] Note that the passwords are uniformly distributed in the dictionary by assumption.

built from general secure multi-party computation techniques but does not require any setup assumptions nor random oracles is due to Goldreich and Lindell [29], which was subsequently simplified at the cost of weakened security in [39]. A stronger PAKE model in the framework of Universal Composability (UC) [18] is due to Canetti, Halevi, Katz and Lindell [19]. UC-secure PAKE protocols require setup assumptions, with CRS being the most popular one [34], albeit ideal ciphers / random oracles [4] and stronger hardware-based assumptions [22] have also been used. In general, all PAKE models (see [40] for the recent overview) take into account unavoidable online dictionary attacks and aim to guarantee security against offline dictionary attacks.

Many PAKE constructions require a constant number of communication rounds [7,27,28,33,34]; a recent framework by Katz and Vaikuntanathan [34] offers an optimal one-round PAKE. We observe that CRS-based PAKE constructions from [28,33] make use of smooth projective hash function families and one-time signatures, whereas in [27] the latter were replaced with (more efficient) MACs. Interestingly, thanks to this replacement PAKE protocols from [27] can be transformed into O-PAKE protocols using our approach, whereas the protocols from [28,33] are not suitable for the transformation; the same applies to the framework in [34], which adopts (simulation-sound) NIZK proofs. We discuss these issues in Section 4.1 Remark 2.

There exist further PAKE flavors, including three-party protocols [3,6] and group protocols [1,8]. Additionally, there are several threshold-based PAKE protocols, e.g. [5,9], where the client's password is shared amongst two (or possibly more) servers that jointly authenticate the client. In addition to the aforementioned approaches that are tailored to the password-based setting there exist several more general authentication and key exchange frameworks such as [13,17] that also lend themselves to the constructions of (somewhat less practical) PAKE protocols. We observe that for many of these aforementioned password-based concepts efficient processing of failed password attempts (without repeating the execution) remains an open problem.

2 Oblivious PAKE Model

(Oblivious) PAKE protocols establish a session key $\mathtt{k}$ between two parties. The security model for O-PAKE protocols is in a multi-user setting and utilizes the Real-or-Random approach for AKE-security from [6,10]. Note that the AKE-security definition also suffices the aforementioned security against malicious servers. A server learning anything about the additional passwords in the client's password vector can easily break AKE-security by using his knowledge in a different session with the same client (and hence same password vector).

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. For each 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 $|\mathcal{D}|$. Let $\mathcal{X}_c(\mathcal{D})$ denote the subset of the power set of $\mathcal{D}$ with all elements of maximum size $c \in \mathbb{N}$. For a client P, let $\mathbf{pw}_P \in \mathcal{X}_c(\mathcal{D})$ denote a vector of passwords with $|\mathbf{pw}_P| = n$ and $1 \leq n \leq c$. We will sometimes write $\mathbf{pw}$ 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 $c = 1$, i.e. standard PAKE.

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 $\mathtt{null}$.
- $\mathtt{state}_P^i$ is the internal state of instance P_i.

– $\texttt{used}_P^i$ indicates whether P_i has already been used.
– $\texttt{role}_P^i$ 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) $\texttt{pid}_P^i = P'$ and $\texttt{pid}_{P'}^j = P$, and (iii) $\texttt{match}(\texttt{sid}_P^i, \texttt{sid}_{P'}^j) = 1$, where Boolean algorithm `match` is defined according to the matching conversations from [11], i.e. outputs 1 if and only if round messages (in temporal order) in $\texttt{sid}_P^i$ equal to the corresponding round messages in $\texttt{sid}_{P'}^j$ except for the final round, in which the incoming message of one instance may differ from the outgoing message of another instance.

Oblivious PAKE: Definition 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* $\texttt{O-PAKE} = (\texttt{init}, \texttt{next})$ *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:*

$P_i \leftarrow \texttt{init}(\mathbf{pw}, \texttt{role}, P', \texttt{par})$**:** *On input* $\mathbf{pw} \in \mathcal{X}_c(\mathcal{D})$*,* $\texttt{role} \in \{\texttt{client}, \texttt{server}\}$*,* $P' \in \Omega$ *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* $\texttt{pid}_P^i = P'$ *and session key* $\texttt{k}_P^i = \texttt{null}$*, and stores protocol parameters* `par`*. The* `role` *indicates whether the participant acts as* `client` *or* `server`*.*

$(m_{\text{out}}, \texttt{k}_P^i) \leftarrow \texttt{next}(m_{\text{in}})$**:** *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* $\texttt{k}_P^i$ *with* $\texttt{k}_P^i \in \mathcal{K} \cup \texttt{null} \cup \perp$*. As long as the instance has not terminated the key* $\texttt{k}_P^i$ *is* `null`*. If* m_{in} *leads to acceptance then* $\texttt{k}_P^i$ *is from* $\mathcal{K}$*, otherwise* $\texttt{k}_P^i = \perp$*. 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 $\texttt{init}(\mathbf{pw}_P, \texttt{client}, P', \texttt{par})$ and P'_j be an instance initialised through $\texttt{init}(\text{pw}_{P,P'}, \texttt{server}, P, \texttt{par})$ where $P \in \mathcal{C}$, $P' \in \mathcal{S}$, and $\text{pw}_{P,P'} \in \mathbf{pw}_P$. Assume that all outgoing messages, generated by `next` are faithfully transmitted between P_i and P'_j so that the instances become partnered. An $\texttt{O-PAKE} = (\texttt{init}, \texttt{next})$ is said to be *correct* if for all partnered P_i and P'_j it holds that $\texttt{k}_P^i \in \mathcal{K}$ and $\texttt{k}_P^i = \texttt{k}_{P'}^j$.

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 $m_{\text{in}} \in \mathcal{M}_{[C,S]}^r$ for the instance P_i and returns its outgoing message $m_{\text{out}} \in \mathcal{M}_{[C,S]}^{r+1} \cup \emptyset$. If P_i does not exist, the according session is created with P'_j as partner, where P'_j 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 [6], 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:
$\mathtt{k}_{\mathcal{A}} \leftarrow \mathsf{Test}(P,i)$, depending on the values of bit b and $\mathtt{k}_P^i$, this oracle responds with key $\mathtt{k}_{\mathcal{A}}$ defined as follows:

- If a pair $(P, \mathtt{pid}_P^i)$ or $(\mathtt{pid}_P^i, P)$ was corrupted (cf. definition of $\mathsf{Corrupt}$ oracle) while $\mathtt{k}_P^i = \mathtt{null}$ then abort. Note that this prevents $\mathcal{A}$ from obtaining $\mathrm{pw}_{P,P'}$ 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, and $b = 0$ then return the same response as to that query. Note that this guarantees consistency of oracle responses.
- If $\mathtt{k}_P^i \in \mathcal{K}$ then if $b = 1$, return $\mathtt{k}_P^i$, else if $b = 0$, return a randomly chosen element from $\mathcal{K}$ and store it for later use.
- Else return $\mathtt{k}_P^i$. Note that in this case $\mathtt{k}_P^i$ is either $\perp$ or $\mathtt{null}$.

According to [6] 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 session there exists a negligible function $\varepsilon(\cdot)$ such that:*

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

$\mathsf{Exp}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}}(\lambda)$: *Choose* $c \in \mathbb{N}, b \in_R \{0,1\}$ *for each pair* $(P,P') \in \mathcal{C} \times \mathcal{S}$, *pick* $\mathrm{pw}_{P,P'} \in_R \mathcal{D}$, *call* $b' \leftarrow \mathcal{A}^{\mathsf{Send},\mathsf{Execute},\mathsf{Corrupt},\mathsf{Test}}(\lambda, c)$ *and return* $b = b'$.

The above definition reverts to ROR AKE-security from [6] 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. To clarify that 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 is allowed to use c times as much online sessions with one password each. The typical advantage of a PAKE adversary $\mathcal{A}$ in an AKE-security experiment, e.g. [6,10], 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 formalize the relation between the two notions.

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

Proof (sketch). The lemma follows directly from the following observations. O-PAKE can be realized in the naïve way by running c separate PAKE sessions. That results in an advantage of at most $c \cdot \mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}} = c \cdot \mathcal{O}(t)/|\mathcal{D}| + \varepsilon'(\lambda)$. Information gathered from Send and $\mathsf{Execute}$ oracle invocations are the same for the O-PAKE and PAKE adversary. $\mathsf{Corrupt}$ and Test 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 from different dictionaries, i.e. weak passwords beside stronger ones. 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. Nonetheless, O-PAKE does not lower the security in contrast to common PAKE. In the case of diverse password probabilities the protocol security is limited by the weakest client password using PAKE or O-PAKE.

3 Transforming PAKE Protocols into O-PAKE

As discussed in the introduction it is easy to implement O-PAKE the naïve by running the input PAKE n times. That approach however is not efficient due to the linear increasing round complexity and the enormous server load. The idea of the O-PAKE compiler is to "encrypt" the n PAKE messages on client side such that the server can "decrypt" the "correct" one with the common password and answer only on that. However, using encryption does not work as the server does not know which PAKE message to decrypt, and when he knows (by some kind of verification mechanism) the protocol would become insecure as a malicious server could launch an offline dictionary attack.

Therefore, we present our compiler that follows the high-level intuition from Section 1.1 and uses mainly two building blocks — *Index-Hiding Message Encoding (IHME)* [36,37] and *admissible encoding* [14,16,24] — to generically construct AKE-secure O-PAKE protocols from (suitable) AKE-secure PAKE protocols, preserving constant round complexity and offering nearly constant server load. An IHME scheme consists of two algorithms `iEncode` and `iDecode`. The `iEncode` algorithm takes as input a set of index-message pairs $(i_1, m_1), \ldots, (i_n, m_n)$ and outputs a structure S whereas the `iDecode` algorithm can extract m_j, $j \in [1, n]$ from S using the corresponding index i_j. For formal definitions surrounding IHME we refer to Appendix A or the original work and only mention that the original IHME construction in [36] assumes $(i_j, m_j) \in \mathbb{F}$ 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 [37] for longer messages, which uses $(i_j, m_j) \in \mathbb{F} \times \mathbb{F}^\nu$ 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 i_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). This approach still allows the server to learn which of the n PAKE sessions is the "correct" one, however, it does not reveal any new password the server.

In order to enable encoding of PAKE messages using IHME with passwords as indices we first apply the inverse transformation $\mathcal{I}_F$ of an admissible encoding F (cf. Definition 3) to map PAKE messages into the IHME message space where necessary. In Section 3.2 we will discuss suitable PAKE message spaces for which there exist admissible encodings that offer compatibility with the message space $\mathbb{F}$ of the IHME schemes from [36,37].

Definition 3 (Admissible Encoding [16]). *Let S and R denote two finite sets such that $|S| > |R|$. A function $F : S \to R$ is called an* ϵ-admissible encoding *for (S, R) if it satisfies the following properties:*

1. Efficient*: F is computable in deterministic polynomial time.*
2. Invertible *: There exists a polynomial time algorithm $\mathcal{I}_F$ such that $\mathcal{I}_F(r) \in F^{-1}(r) \cup \{\perp\}$ for all $r \in R$*
3. Uniform*: For all r uniformly distributed in R the distribution of $\mathcal{I}_F(r)$ is ϵ-statistically indistinguishable from the uniform distribution over S.*

If ϵ is a negligible function of the security parameter then F is called an admissible encoding*.*

In the following we describe the compiler that allows to transform a suitable AKE-secure PAKE protocol into an AKE-secure O-PAKE protocol. The intuition behind the compiler is to let the client run n PAKE sessions for all input passwords and apply an index-hiding message encoding on each PAKE message, password pair such that the server is only able to recover the PAKE message he knows the password of. After recovering the "correct" PAKE message on the server side, the PAKE protocol with the according password is executed and the result returned to client just as in the original PAKE protocol. When the input protocol finished the server sends a confirmation message to the client so that he can identify the correct PAKE session and thus key.

3.1 The O-PAKE Compiler

Our compiler takes as input a PAKE protocol Π and outputs its O-PAKE version, denoted C_Π, which contains two algorithms `init` and `next`, as detailed in Figure 1. For each PAKE round r it uses a corresponding instance $\texttt{IHME}^r$ with message space $\mathcal{M}^{\texttt{IHME},r}$ and an compatible instance $F^r : \mathcal{M}^{\texttt{IHME},r} \rightarrow \mathcal{M}_C^r$ where $\mathcal{M}_C^r$ is the space of client's messages of Π in that round. In the following we will first assume that underlying $\Pi.\texttt{next}$ algorithm in each round outputs messages that can be seen as elements of one set and thus can be processed using one instance $(F^r, \texttt{IHME}^r)$. We will 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 separately at the end of this section.

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 pw as index) and replies using with the message output by $\Pi.\texttt{next}$. Note that 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.

Algorithm $C_\Pi.\texttt{init}$ As specified in Algorithm 1, $C_\Pi.\texttt{init}$ makes n calls to $\Pi.\texttt{init}$ in Line 3, one for each password in $\mathbf{pw}$, to generate corresponding `state` for each of the n PAKE sessions that are stored in $\mathbf{state}_P^i$. The partner id $\texttt{pid}_P^i$ is set to P' (Line 5) and the instance P_i with the given `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 equal, which is needed 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.

Algorithm $C_\Pi.\texttt{next}$ We separate between $C_\Pi.\texttt{next}$ specifications for clients (Algorithm 2) and servers (Algorithm 3) as they are significantly different. We write $\Pi_{\mathbf{pw}[i]}.\texttt{next}$ for the invocation of $\Pi.\texttt{next}$, where the protocol `state` is initialised with $\mathbf{pw}[i]$. On the client side $C_\Pi.\texttt{next}$ computes messages for all running PAKE sessions in Line 4 and encodes them (cf. Line 10). The server decodes the incoming message and computes its response using $\Pi.\texttt{next}$. (cf. Line 3-7). If on the client side, after processing the incoming round message $\texttt{k} \in \mathcal{K}_\Pi$ then the client expects server's confirmation message, which it then checks and eventually derives the final session key. Computation of this confirmation message with subsequent key derivation on the server side is specified in Line 10 and Line 11. Note that the use of the pseudorandom function `PRF` ensures the independence between the final session key and the confirmation message. If the confirmation message is not valid or some other failures occurred during the execution of the protocol then parties will end up with the session key being set to $\perp$ (cf. Line 13, resp. 12).

Remark 1 (Compiler Efficiency). The compiler offers several possibilities to save computation time. PAKE protocols with built-in key confirmation (e.g. [7]) for example can omit the additional key confirmation step from the compiler and use the given one to identify the correct PAKE session on the client side. Protocol specific encodings of PAKE messages to identify individual parts slow the compiler down as it has to decode messages before applying admissible and index-hiding encodings. This can be omitted by implementing the compiler protocol specific and therefore use the message elements directly.

The AKE-security of an O-PAKE protocol generated with the O-PAKE compiler is established in Theorem 1 and proven in Appendix B.

Theorem 1. *If Π is an AKE-secure PAKE protocol, for which admissible encodings for every output message of $\Pi.\texttt{next}$ of the client exist, and* `IHME` *is an index-hiding message encoding, then C_Π is an AKE-secure O-PAKE protocol.*

Algorithm 1 $C_{\Pi}.\texttt{init}(\mathbf{pw}, \texttt{role}, P', \texttt{par})$

Input: $\mathbf{pw}, \texttt{role}, P', \texttt{par}$
Output: P_i

initialise empty $\mathbf{state}_P^i$ vectors of length $n = |\mathbf{pw}|$ and $\texttt{k}_P^i = \texttt{null}$
for $l = 1 \dots n$ **do**
  $P_l \leftarrow \Pi.\texttt{init}(\mathbf{pw}[l], \texttt{role}, P', \texttt{par})$
  $\mathbf{state}_P^i[l] = \texttt{state}_P^l$
$\texttt{pid}_P^i = P'$
return P_i

Algorithm 2 $C_{\Pi}.\texttt{next}(m_{\text{in}})$ — Client

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

$P = \emptyset$
for $i = 1 \dots n$ **do**
  if $\Pi[i]$ has not finished **then**
    $(m'_{\text{out}}, \texttt{k}) \leftarrow \Pi_{\mathbf{pw}[i]}.\texttt{next}(m_{\text{in}})$
    $P = P \cup \{(\mathbf{pw}[i], \mathcal{I}_{F^r}(m'_{\text{out}}))\}$
    $\texttt{state}[i].\texttt{k} = \texttt{k}$
  else if $\Pi[i]$ has finished && $m_{\text{in}} = \texttt{PRF}_{\Pi[i].key}(\texttt{sid}_P^i, 0)$ **then**
    $\texttt{k} = \texttt{PRF}_{\Pi[i].\texttt{k}}(\texttt{sid}_P^i, 1)$
if $P \neq \emptyset$ **then**
  $m_{\text{out}} = \texttt{IHME}^r.\texttt{iEncode}(P)$
else if $\texttt{k} \notin \mathcal{K}_{C_\Pi}$ **then**
  $\texttt{k} = \perp$
return $(m_{\text{out}}, \texttt{k})$

Algorithm 3 $C_{\Pi}.\texttt{next}(m_{\text{in}})$ — Server

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

if $\texttt{k} = \texttt{null}$ **then**
  if $m_{\text{in}} \neq null$ **then**
    $m \leftarrow \texttt{IHME}^r.\texttt{iDecode}(\text{pw}, m_{\text{in}})$
    $m' = F^r(m)$
  else
    $m' = m_{\text{in}}$
  $(m_{\text{out}}, \texttt{k}) \leftarrow \Pi.\texttt{next}(m')$

else if $\texttt{k} \in \mathcal{K}_\Pi$ **then**
  $m_{\text{out}} = \texttt{PRF}_k(\texttt{sid}_{P'}^j, 0)$
  $\texttt{k} \leftarrow \texttt{PRF}_{\texttt{k}}(\texttt{sid}_{P'}^j, 1)$
else
  $(m_{\text{out}}, \texttt{k}) = (\emptyset, \perp)$
return $(m_{\text{out}}, \texttt{k})$

Fig. 1: O-PAKE Compiler

Processing Multi-Component Messages As mentioned before, message spaces of PAKE protocols mostly consist of Cartesian products of message spaces of the individual message components. We now briefly discuss how the compiler proceeds in the case of multi-component messages.

Message components can for instance be constants, group elements or integers. As constants are password independent they do not have to be processed by the compiler and can be sent directly. All other message components have to be encoded as described in the compiler. [37] introduces ν-fold IHME (cf. Appendix A), which allows to encode a list of ν message components from the same finite field. Therefore, we separate message components from different finite fields into message component classes. Message components from different classes (finite fields) have to be encoded in separate IHME structures over the corresponding finite field. This requires admissible encodings and index-hiding message encodings for each element class m_j of m. Thus, a loop over $m_1, \dots, m_l$ adds $(\mathbf{pw}[i], \mathcal{I}_{F^{r,j}}(m_j))$ to the input set of $\nu-\text{fold}-\text{IHME}_j^r.\texttt{iEncode}$ according to their message classes (finite fields). Likewise, the output message m_{out} of the `next` algorithm is the concatenation of the encoded message component classes. The server upon receiving a client message m_{in} 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 $F^{r,j}(m_j)$.

If we adopt the above approach for multi-component messages then the AKE-security remains preserved. This is due to the following observation on the proof of Theorem 1: in the game-hopping sequence the adversary will be provided with l IHME encoded messages (one for each message component class that requires encoding) and the corresponding index-hiding advantage will have to be multiplied by l.

3.2 Oblivious PAKE Instantiation

An AKE-secure PAKE protocol Π is suitable for our O-PAKE transformation if it outputs client messages that are uniformly distributed in respective round-dependent message spaces and there exist admissible encodings that can map those messages into the message space of IHME. We give a list of four sets R with suitable admissible encodings, meaning that any AKE-secure PAKE protocols whose client messages contain components from these four sets can be transformed into an O-PAKE protocol using our compiler.

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

(1) Set $R = \{0, \ldots, N-1\} = \mathbb{Z}_N$ of natural numbers, for arbitrary $N \in \mathbb{N}$. (cf. [24, Lemma 12])
(2) The set of quadratic residues modulo safe primes p, i.e. $R = QR(p) \subseteq \mathbb{Z}_p^\times$. (cf. [24, Lemma 12])
(3) Arbitrary subgroups $G_q \subseteq \mathbb{Z}_p^\times$ of prime order q. (cf. [24, Lemma 12])
(4) The set $R = E(\mathbb{F})$ of rational points on (certain) elliptic curves, defined over a finite field. (cf. [16])

4 Implementation & Performance

To show the practicality of the O-PAKE compiler we implement it in `C++` (using gcc 4.7.2), together with two concrete O-PAKE protocol implementations based on well-known PAKE protocols, namely the SPAKE protocol from [7] and the PAKE framework from [27].

4.1 Compiler Implementation

The O-PAKE compiler implementation consists basically of three packages: Admissible Encodings, IHME and the actual O-PAKE Compiler. The first package implements admissible encodings from Definition 3 and its references. The second package contains the implementation of IHME from [37]. The actual compiler functionality is implemented in the third package as an abstract class with virtual `init` and `next` functions. Despite the virtual functions the compiler offers all functions necessary to implement an O-PAKE protocol from a suitable PAKE protocol, namely it initializes and stores all c PAKE instances and offers `next` functions according to Algorithms 2 and 3. The compiler expects the server to initiate the O-PAKE protocol, e.g. after receiving the client's username and retrieving the corresponding password from the database.

The compiler's modular design makes it easy to implement O-PAKE protocols from suitable PAKE protocols. On high level, it is sufficient to derive a concrete instance of the abstract O-PAKE compiler class and implement the virtual `init` and `next` functions with protocol specific message handling and suitable admissible encodings to implement an instance of the O-PAKE compiler. Before we show how to use the compiler on the example of SPAKE and RG-PAKE we specify necessary admissible encodings and key derivation steps.

Admissible Encodings Using PAKE protocols on prime order groups, we briefly discuss the implementations of admissible encodings according to Definition 3 (1) and (3). To implement (3) we use (1) $\mathcal{I}_{F^{(1)}} : \mathbb{Z}_N \to \{0,1\}^{\ell(\lambda)}$ to get $\mathcal{I}_{F^{(3,1)}} : G_q \subseteq \mathbb{Z}_p^\times \to \mathbb{Z}_p^\times \to \{0,1\}^{\ell(\lambda)}$ with $\ell(\lambda) > 2|N|$ and $p = N$. Implementation of (1,3) is straight forward according to [24, Lemma 12] after generating appropriate $\ell(\lambda)$. To use the output of (1) as input to IHME $\ell(\lambda)$ has to be the next prime q' with $q' = \ell(\lambda) > 2|N|$ such that the implemented admissible encoding is defined as $\mathcal{I}_{F^{(3,1)}} : G_q \to \mathbb{Z}_{q'}$.

Algorithm 4 SPAKE.init(pw, role, $P', j, (\mathbb{G}, M, N, H)$)

Input: pw, role, P', j
Output: P_i
1: $i \in \mathbb{N}$
2: Create P_i
3: $\texttt{state}_P^i = \{\text{pw}, \texttt{role}, (\mathbb{G}, M, N, H)\}$
4: $\texttt{pid}_P^i = P'$
5: **return** P_i

Algorithm 5 RG.init(pw, role, P', j, pk)

Input: pw, role, P', j
Output: P_i
1: $i \in \mathbb{N}$
2: Create P_i
3: $\texttt{state}_P^i = \{\text{pw}, \texttt{role}, \texttt{pk}\}$
4: $\texttt{pid}_P^i = P'$
5: **return** P_i

Confirmation and Key Derivation The final key confirmation and derivation steps are implemented according to Algorithms 2 and 3. In particular, we use CBC-MAC [31, Sec. 4.5] with AES-256 [31, Sec. 5.5] to instantiate the pseudorandom function. The output of the confirmation value is sent to the client. Using this, the client can check the confirmation message, choose the correct PAKE session and compute the final key.

In the following we demonstrate how our compiler can be applied to implementations of two efficient PAKE protocols — the random-oracle-based SPAKE protocol from [7] and the standard-model RG-PAKE protocol framework from [27].

Oblivious SPAKE The SPAKE protocol from [7, Sec. 5], which is a secure variant of [12], is a PAKE protocol whose security has been proven in the random oracle model. SPAKE uses a prime-order cyclic group $\mathbb{G} = (G, g, q)$ for which the Computational Diffie-Hellman (CDH) is assumed to be hard. We recall the corresponding specification of SPAKE in Algorithms 4, 6 and 7, where $M, N \in \mathbb{G}$ and H are considered as random oracles. The shared SPAKE password pw is chosen from $\mathbb{Z}_q$. The protocol proceeds in one round, where client and server each send one message in arbitrary order (cf. Line 2 to 5 in Algorithms 6 and 7). Upon receiving the message the algorithm `next` starts performing the key derivation step (cf. Line 10).

The SPAKE protocol is a suitable input PAKE protocol for our O-PAKE compiler since it can instantiated using prime-order subgroups of $\mathbb{Z}_p^\times$ or prime order subgroups of $E(\mathbb{F})$ in which the CDH problem is believed to be hard and for its messages there exist a suitable admissible encoding. In particular, the admissible encodings in (3) and (4) are suitable due to the fact that SPAKE messages of the form $X^* = g^x \cdot M^{\text{pw}}$ are uniformly distributed in $\mathbb{G}$, given the uniformity of the exponent x. The implementation of Oblivious SPAKE (O-SPAKE) is straightforward as only one group element is exchanged in a single round per party. We use prime-order subgroups of $\mathbb{Z}_p^\times$ in which the CDH problem is assumed to be hard.[2]

Oblivious RG-PAKE As a second O-PAKE instantiation we present the protocol based on the efficient PAKE framework proposed by Gennaro [27] and proven AKE-secure under standard assumptions in the model from [10] assuming the existence of a Common Reference String (CRS). We refer to the PAKE protocols that can be obtained using the framework from [27] as RG-PAKE. Their construction relies on a CCA-secure labeled encryption algorithm `Enc`, a smooth projective hash function $(\alpha, \mathcal{H}_k)$. For an overview of these building blocks we refer to [27] and focus here on the protocol communication. The RG-PAKE protocol consists of three rounds and is depicted in Algorithms 5, 8 and 9. The client initially computes the ciphertext c (cf. Algorithm 8, Line 2 to 5)) and sends it to the server that in turn replies with the projected value s and another ciphertext c' (cf. Algorithm 9, Line 2 to 7). This message is then parsed by the client that first computes its final message, consisting of the projected value s' and a MAC value t and finally derives the session key (cf. Algorithm 8, Line 8 to 15). The server checks the received MAC value t and eventually computes the session key (cf. Algorithm 9, Line 9 to 16).

[2] We use the Botan cryptographic library (`http://botan.randombit.net`) and the prime-order groups specified in RFC 3526 [35].

Algorithm 6 SPAKE.$\texttt{next}(m_{\text{in}})$ — Client

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

1: **if** not $\texttt{used}$ **then**
2: $x \in_R \mathbb{Z}_p, X = g^x$
3: $X^* = X \cdot M^{\text{pw}}$
4: $\texttt{state}_P^i \leftarrow \texttt{state}_P^i \cup (X^*, x)$
5: $(m_{\text{out}}, \texttt{k}) = (X^*, \texttt{null})$
6: **if** $m_{\text{in}} \neq \emptyset$ **then**
7: goto 10
8: **else**
9: $m_{\text{out}} = \emptyset$
10: $K = (m_{\text{in}}/N^{\text{pw}})^x)$
11: $\texttt{k} = H(P, P', X^*, m_{\text{in}}, \text{pw}, K)$
12: **return** $(m_{\text{out}}, \texttt{k})$

Algorithm 7 SPAKE.$\texttt{next}(m_{\text{in}})$ — Server

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

1: **if** not $\texttt{used}$ **then**
2: $y \in_R \mathbb{Z}_p, Y = g^y$
3: $Y^* = Y \cdot M^{\text{pw}}$
4: $\texttt{state}_{P'}^j \leftarrow \texttt{state}_{P'}^j \cup (Y^*, y)$
5: $(m_{\text{out}}, \texttt{k}) = (Y^*, \texttt{null})$
6: **if** $m_{\text{in}} \neq \emptyset$ **then**
7: goto 10
8: **else**
9: $m_{\text{out}} = \emptyset$
10: $K = (m_{\text{in}}/N^{\text{pw}})^y)$
11: $\texttt{k} = H(P, P', m_{\text{in}}, Y^*, \text{pw}, K)$
12: **return** $(m_{\text{out}}, \texttt{k})$

Whether RG-PAKE is a suitable protocol for our O-PAKE compiler depends on the existence of appropriate admissible encodings for the sets of possible ciphertexts c, projected hash values s' and MAC values t and their distribution in the corresponding sets, which needs to be uniform. As discussed in [21, 27, 28], suitable encryption schemes and smooth projective hash functions for RG-PAKE are known to exist from various number-theoretic assumptions, including Decision Diffie-Hellman (DDH), quadratic residuosity, and N-residuosity assumptions. We observe that for client's ciphertexts c and projected values s' output by the those schemes an appropriate admissible encoding can be found amongst those mentioned in Section 3.2. That is, our compiler is applicable to a large class of PAKE protocols that can be explained through the framework from [27]. With regard to MAC values t we notice that the corresponding MAC algorithm must not only be unforgeable but also have pseudorandom outputs, in which case the admissible encoding can be realized as an identity function. Interestingly, if the RG-PAKE protocol is used in a way where the server performs Algorithm 8 and the client performs Algorithm 9 (which is possible since RG-PAKE is symmetric with no strict separation into the client/server roles), then no pseudorandomness of MAC values would be required, as they would become part of server's messages.

In our implementation of Oblivious RG-PAKE (O-RG-PAKE) we rely on the DDH-based instantiation, and realize the encryption algorithm $\texttt{Enc}$ using the CCA-secure labeled Cramer-Shoup encryption scheme [20, 43] and the smooth projective hash function $(\alpha, \mathcal{H}_k)$ using the scheme from [28, 32]. This leads to the CRS containing the public key $\texttt{pk} = (b, d, h, g_1, g_2, q, H) = (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^z, g_1, g_2, q, H)$ where $x_1, x_2, y_1, y_2, z \in_R \mathbb{Z}_q$, $g_1, g_2 \in_R G$ are generators, and H is a universal one-way hash function[3]. Given the above instantiations, the ciphertext c and projected hash value s' in the resulting RG-PAKE protocol have the following form:

$c \leftarrow \texttt{Enc}_{\texttt{pk}}(\text{pw}, \texttt{label})$: $c = (u_1, u_2, e, v)$ with $u_1 = g_1^r$, $u_2 = g_2^r$, $e = h^r g^{\text{pw}}$, $v = (bd^\theta)^r$, $\theta = H(u_1, u_2, e, \texttt{label})$ and $r \in_R \mathbb{Z}_q$.

$s' \leftarrow \alpha(k, c, \texttt{label})$: $s' = g_1^{k_1} g_2^{k_2} h^{k_3} (bd^\theta)^{k_4}$ for key $k \in \mathbb{Z}_q^4 = (k_1, k_2, k_3, k_4)$ with $k_1, k_2, k_3, k_4 \in_R \mathbb{Z}_q$ and $\theta = H(u_1, u_2, e, \texttt{label})$.

As a final remark we observe that in order to encrypt the password in Line 3 on client and Line 5 on server side, our implementation first hashes it into the group $\mathbb{G} = (G, g, p)$ within $\texttt{Enc}_{\texttt{pk}}$ by computing g^{pw}. The temporary session keys $\texttt{sk}$ and $\texttt{sk}'$ are computed using the smooth projective hash function $\mathcal{H}_k(c, \text{pw})$ in one of the following two ways: (1) as $\texttt{sk} = u_1^{k_1} u_2^{k_2} (e/(g^{\text{pw}}))^{k_3} v^{k_4}$ using the projection key $k = (k_1, k_2, k_3, k_4)$ and the ciphertext $c = (u_1, u_2, e, v)$, or (2) as $\texttt{sk} = s^r$ using the projection value s and the randomness r used in the generation of c.

[3] H is implemented using collision-resistant hash function SHA-256 [38].

Algorithm 8 RG.$\texttt{next}(m_{\text{in}})$ — Client

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

if $m_{\text{in}} = \emptyset$ **then**
  $\texttt{label} = (P, P')$
  $c \leftarrow \texttt{Enc}_{\texttt{pk}}(\text{pw}, \texttt{label})$
  $\texttt{state}_P^i \leftarrow \texttt{state}_P^i \cup (c)$
  $(m_{\text{out}}, \texttt{k}) = (c, \texttt{null})$

else
  Parse m_{in} as (s, c')
  $k \in_R K, s' = \alpha(k, c')$
  $\texttt{label}' = (c, s)$, $\texttt{label} = (P, P')$
  $\texttt{sk} = \mathcal{H}_k(c', \text{pw}_{P,P'}, \texttt{label}')$
  $\texttt{sk}' = \mathcal{H}_{k'}(c, \text{pw}_{P,P'}, \texttt{label})$
  $t = \texttt{MAC}_{\texttt{sk}'}(c, s, c', s') \oplus \texttt{sk}$
  $\texttt{k} = \texttt{sk} \oplus \texttt{sk}'$
  $(m_{\text{out}}, \texttt{k}) = ((s', t), \texttt{k})$

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

Algorithm 9 RG.$\texttt{next}(m_{\text{in}})$ — Server

Input: m_{in}
Output: $(m_{\text{out}}, \texttt{k})$

if not $\texttt{used}$ **then**
  $k \in_R K, s = \alpha(k, m_{\text{in}})$
  $\texttt{label} = (P, P')$, $\texttt{label}' = (m_{\text{in}}, s)$
  $\texttt{sk} = \mathcal{H}_k(m_{\text{in}}, \text{pw}_{P,P'}, \texttt{label})$
  $c' \leftarrow \texttt{Enc}_{\texttt{pk}}(\text{pw}, \texttt{label}')$
  $\texttt{state}_{P'}^j \leftarrow \texttt{state}_{P'}^j \cup (c, s, c')$
  $(m_{\text{out}}, \texttt{k}) = ((s, c'), \texttt{null})$
else
  Parse m_{in} as (s', t)
  $\texttt{label} = (c, s)$
  $\texttt{sk}' = \mathcal{H}_{k'}(c', \text{pw}_{P,P'}, \texttt{label})$
  if $t = \texttt{MAC}_{\texttt{sk}}(c, s, c', s') \oplus \texttt{sk}'$ **then**
    $\texttt{k} = \texttt{sk} \oplus \texttt{sk}'$
    $(m_{\text{out}}, \texttt{k}) = (\emptyset, \texttt{k})$
  else
    $(m_{\text{out}}, \texttt{k}) = (\emptyset, \perp)$

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

Remark 2 (MACs vs. One-Time Signatures and NIZKs). CRS-based PAKE frameworks from [28, 33] used one-time signatures, whereas in the RG-PAKE framework [27] those were essentially replaced by MACs for better efficiency. Interestingly, unlike the RG-PAKE protocols that can be directly made oblivious using our O-PAKE compiler, the protocols from [28, 33] cannot because the use of one-time signatures introduces for an adversary the possibility to publicly check the consistency of PAKE messages by performing signature verification. Note that existence of such public checks rules out any index-hiding encoding of those messages. That is, PAKE protocols from [28,33], if processed with our O-PAKE compiler, would become susceptible to offline dictionary attacks. The same reasoning applies to the PAKE framework in [34], which adopts publicly verifiable (simulation-sound) NIZK proofs [41,42]. In contrast, the more efficient MAC-based approach taken in RG-PAKE [27] doesn't admit public consistency checks of the resulting PAKE messages.

4.2 Performance Analysis

As discussed in the introduction the naïve solution of executing n PAKE protocols allows for several improvements, namely on the round complexity and the server's computation costs. We have shown how to improve those with the proposed compiler. In the following we substantiate this by conducting a performance analysis of our implementation. All calculations are done on the 2048 bits modular exponential Diffie-Hellman group from [35].

The proposed O-PAKE compiler is efficient in the sense that it achieves almost constant server load and preserves the constant round complexity of the original PAKE protocol. The O-PAKE compiler doesn't reduce the overall length of transmitted client messages compared with the naïve solution due to the overhead implied by the encoding process. In our implementation of the SPAKE protocol parties exchange two group elements in total with each element being 2048 bits long. In RG-PAKE client and server exchange 10 group elements altogether (five per party) and one MAC value. In our RG-PAKE implementation this corresponds to $10 \cdot 2048 + 128 = 20608$ bits from which 128 bits are the output of the MAC algorithm, implemented as CBC-MAC with AES.

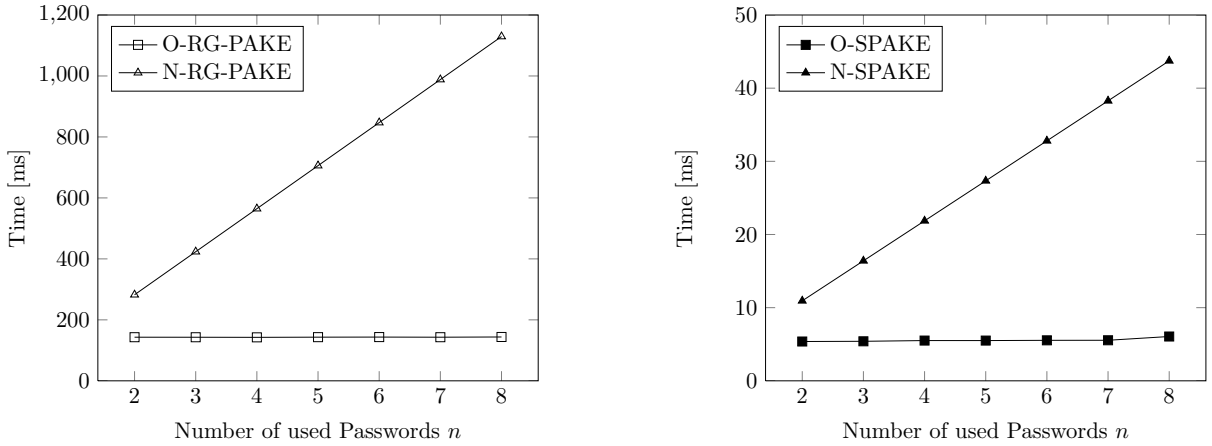

Fig. 2: Server-side Timings for Oblivious RG-PAKE (left) and SPAKE (right) Protocols

Considering the compiled versions of SPAKE and RG-PAKE, denoted O-SPAKE and O-RG-PAKE respectively, we observe that the encoding process applied to client messages increases the length of their group elements to 4096 bits due to the bound on function ℓ from Section 4.1. Note that this increase comes from the application of admissible encoding and not from IHME, which is length-preserving. In contrast server's messages are not encoded and thus do not experience any size increase at all. Only in the final compiler round where the server needs to send its confirmation message, computed using CBC-MAC with AES, additional 128 bits needs to be communicated.

We proceed with the experimental performance analysis of the implemented O-SPAKE and O-RG-PAKE protocols and their comparison with the naïve execution involving multiple independent PAKE sessions. An Intel(R) Core(TM)2 Duo P8600 @ 2.40GHz platform was used as a reference architecture. The measurements include the entire computation time (exclusive network delays) from the moment of initialization and until the derivation of the session key. The original SPAKE protocol requires roughly 5ms for the client and server and is significantly faster than the RG-PAKE protocol that requires 141ms. Figure 2 illustrates server-side timings measured for our O-SPAKE and O-RG-PAKE implementations and compares them with the performance of naïve protocols (N-SPAKE, N-RG-PAKE) with linear number of PAKE sessions on the server side. The exact measurements are provided in Table 1. The first thing to notice is that the server runtime for protocols obtained through our O-PAKE compiler is almost independent from the number of used client passwords. With roughly 5.5ms for O-SPAKE and 143ms for O-RG-PAKE, the compiled protocols are significantly faster than the corresponding naïve protocols. The latter experience linear increase in the server's runtime with the number of used passwords. For instance, if two client passwords are used then the running time

Table 1: Oblivious and Naïve PAKE Measurements [ms]

c	**O-SPAKE**		**N-SPAKE**		**O-RG-PAKE**		**N-RG-PAKE**	
	Client	Server	Client	Server	Client	Server	Client	Server
2	52.415	5.374	10.83	10.934	487.872	143.079	282.014	282.332
3	52.609	5.402	16.245	16.401	756.986	142.95	423.021	423.498
4	112.561	5.503	21.66	21.868	995.472	142.64	564.028	564.664
5	142.927	5.502	27.075	27.335	1274.418	143.226	705.035	705.83
6	176.513	5.535	32.49	32.802	1564.885	143.494	846.042	846.996
7	210.325	5.543	37.905	38.269	1838.485	143.078	987.049	988.162
8	245.79	6.057	43.32	43.736	2127.653	143.817	1128.056	1129.328

of N-SPAKE with roughly 10.8ms and of N-RG-PAKE with 282ms is almost double the time required by the compiled versions of those protocols. Due to the IHME decoding step a quadratic increase in server's runtime might be expected. However, this increase remains unnoticeable in practice where on average two to five passwords are used (cf. Section 1).

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 optimize 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. The proposed compiler can be used with arbitrary PAKE protocols as input as long as those messages are uniformly distributed within the corresponding message space and can be encoded through a suitable admissible encoding scheme. The security of the compiler has been proven under standard assumptions in a widely used PAKE model from [6] and its practicality has been demonstrated experimentally through the implementation and compilation of two efficient PAKE constructions from [7, 27].

References

1. M. Abdalla, J.-M. Bohli, M. I. G. Vasco, and R. Steinwandt. (Password) Authenticated Key Establishment: From 2-Party to Group. In *TCC'07*, volume 4392 of *LNCS*, pages 499–514. Springer, 2007.
2. M. Abdalla, E. Bresson, O. Chevassut, B. Möller, and D. Pointcheval. Provably secure password-based authentication in TLS. In *ASIACCS'06*, pages 35–45, New York, NY, USA, 2006. ACM.
3. M. Abdalla, E. Bresson, O. Chevassut, B. Moller, and D. Pointcheval. Strong password-based authentication in TLS using the three-party group Diffie Hellman protocol. *Int. J. Secur. Netw.*, 2(3/4):284–296, Apr. 2007.
4. M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In *CT-RSA'08*, pages 335–351, Berlin, Heidelberg, 2008. Springer-Verlag.
5. M. Abdalla, O. Chevassut, P.-A. Fouque, and D. Pointcheval. A simple threshold authenticated key exchange from short secrets. In *ASIACRYPT'05*, pages 566–584, Berlin, Heidelberg, 2005. Springer-Verlag.
6. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In *PKC'05*, pages 65–84, Berlin, Heidelberg, 2005. Springer-Verlag.
7. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA'05*, pages 191–208, Berlin, Heidelberg, 2005. Springer-Verlag.
8. M. Abdalla and D. Pointcheval. A Scalable Password-Based Group Key Exchange Protocol in the Standard Model. In *ASIACRYPT'06*, volume 4284 of *LNCS*, pages 332–347. Springer, 2006.
9. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *CCS'11*, pages 433–444, New York, NY, USA, 2011. ACM.
10. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, pages 139–155, Berlin, Heidelberg, 2000. Springer-Verlag.
11. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, pages 232–249, New York, NY, USA, 1994. Springer.
12. S. M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *IEEE Symposium on Research in Security and Privacy*, pages 72–84, 1992.
13. O. Blazy, D. Pointcheval, and D. Vergnaud. Round-Optimal privacy-preserving protocols with smooth projective hash functions. In *TCC'12*, pages 94–111, Berlin, Heidelberg, 2012. Springer-Verlag.
14. D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO'01*, volume 2139 of *LNCS*, pages 213–229. Springer, 2001.
15. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange using Diffie-Hellman. In *EUROCRYPT'00*, pages 156–171, Berlin, Heidelberg, 2000. Springer.
16. E. Brier, J.-S. Coron, T. Icart, D. Madore, H. Randriam, and M. Tibouchi. Efficient indifferentiable hashing into ordinary elliptic curves. In *CRYPTO'10*, pages 237–254, Berlin, Heidelberg, 2010. Springer-Verlag.
17. J. Camenisch, N. Casati, T. Gross, and V. Shoup. Credential authenticated identification and key exchange. In *CRYPTO'10*, pages 255–276, Berlin, Heidelberg, 2010. Springer-Verlag.

18. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS'01*, page 136, Washington, DC, USA, 2001. IEEE Computer Society.
19. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *EUROCRYPT'05*, pages 404–421, Berlin, Heidelberg, 2005. Springer-Verlag.
20. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In H. Krawczyk, editor, *CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer Berlin / Heidelberg, 1998. 10.1007/BFb0055717.
21. R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen Ciphertext Secure Public-Key Encryption. In *EUROCRYPT'02*, pages 45–64, London, UK, UK, 2002. Springer-Verlag.
22. Ö. Dagdelen and M. Fischlin. Unconditionally-Secure Universally Composable Password-Based Key-Exchange based on One-Time Memory Tokens. *IACR Cryptology ePrint Archive*, 2012, 2012. `http://eprint.iacr.org/`.
23. T. Dierks and E. Rescorla. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2, aug 2008. Updated by RFCs 5746, 5878, 6176.
24. N. Fleischhacker, F. Gnther, F. Kiefer, M. Manulis, and B. Poettering. Pseudorandom Signatures. *IACR Cryptology ePrint Archive*, 2011:673, 2011.
25. D. Florencio and C. Herley. A Large-Scale Study of Web Password Habits. In *16th international conference on World Wide Web*, WWW'07, pages 657–666, New York, NY, USA, 2007. ACM.
26. S. Gaw and E. W. Felten. Password Management Strategies for Online Accounts. In *Symposium on Usable privacy and security*, SOUPS'06, pages 44–55, New York, New York, USA, 2006. ACM Press.
27. R. Gennaro. Faster and Shorter Password-Authenticated Key Exchange. In *TCC'08*, pages 589–606. Springer-Verlag, Berlin, Heidelberg, 2008.
28. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Inf. Syst. Secur.*, 9(2):181–234, may 2006.
29. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *CRYPTO'01*, pages 408–432, London, UK, UK, 2001. Springer-Verlag.
30. S. Jarecki, J. Kim, and G. Tsudik. Beyond Secret Handshakes: Affiliation-Hiding Authenticated Key Exchange. In *CT-RSA'08*, pages 352–369, Berlin, Heidelberg, 2008. Springer.
31. J. Katz and Y. Lindell. *Introduction to modern cryptography*. Chapman & Hall/CRC cryptography and network security. Chapman & Hall/CRC, 2008.
32. J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. In *EUROCRYPT'01*, page 475494. Springer-Verlag, 2001.
33. J. Katz, R. Ostrovsky, and M. Yung. Efficient and Secure Authenticated Key Exchange Using Weak Passwords. *J. ACM*, 57(1):3–1, nov 2009.
34. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC'11*, pages 293–310, Berlin, Heidelberg, 2011. Springer-Verlag.
35. T. Kivinen and M. Kojo. RFC 3526 - More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE), May 2003.
36. M. Manulis, B. Pinkas, and B. Poettering. Privacy-preserving group discovery with linear complexity. In *ACNS'10*, pages 420–437, Berlin, Heidelberg, 2010. Springer-Verlag.
37. M. Manulis and B. Poettering. Practical affiliation-hiding authentication from improved polynomial interpolation. In *ASIACCS'11*, pages 286–295, New York, NY, USA, 2011. ACM.
38. National Institute of Standards and Technology. Federal Information Processing Standard 180-4 — Secure Hash Standard (SHS), 2012.
39. M.-H. Nguyen and S. P. Vadhan. Simpler Session-Key Generation from Short Random Passwords. In *TCC'04*, volume 2951 of *LNCS*, pages 428–445. Springer, 2004.
40. D. Pointcheval. Password-based authenticated key exchange. In *PKC'12*, pages 390–397, Berlin, Heidelberg, 2012. Springer-Verlag.
41. A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *FOCS'99*, page 543, Washington, DC, USA, 1999. IEEE Comput. Soc.
42. A. D. Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero Knowledge. In *CRYPTO'01*, volume 2139 of *LNCS*, pages 566–598. Springer, 2001.
43. V. Shoup. A Proposal for an ISO Standard for Public Key Encryption, 2001. sho@zurich.ibm.com 11676 received 20 Dec 2001.

A Index-Hiding Message Encoding (IHME)

The concept of an index-based message encoding scheme with the index-hiding property (IHME) was introduced in [36, 37]. Here we recall their definitions and constructions.

Definition 4 (Index-Based Message Encoding [36]). *An* index-based message encoding *scheme* ($\texttt{iEncode}$, $\texttt{iDecode}$) *over an index space* $\mathcal{I}$ *and a message space* $\mathcal{M}$ *consists of two efficient algorithms:*

$\mathcal{S} \leftarrow \texttt{iEncode}(\mathcal{P})$**:** *On input consisting of a tuple of index-message pairs* $\mathcal{P} = \{(i_1, m_1), \ldots, (i_n, m_n)\} \subseteq \mathcal{I} \times \mathcal{M}$ *with distinct indices* $i_1, \ldots, i_n$*, this algorithm outputs an encoding* $\mathcal{S}$*.*

$m \leftarrow \texttt{iDecode}(\mathcal{S}, i)$**:** *On input of an encoding* $\mathcal{S}$ *and an index* $i \in \mathcal{I}$*, this algorithm outputs a message* $m \in \mathcal{M}$*.*

An index-based message encoding scheme is correct *if* $\texttt{iDecode}(\texttt{iEncode}(\mathcal{P}), i_j) = m_j$ *for all* $j \in \{1, \ldots, n\}$ *and all tuples* $\mathcal{P} = \{(i_1, m_1), \ldots, (i_n, m_n)\} \subseteq \mathcal{I} \times \mathcal{M}$ *with distinct indices* i_j*.*

The index-hiding property of an index-based message encoding ensures that all indices are hidden from the receiver of the message, as long as he does not know which message to expect. Thus, a receiver of an index-hiding encoded message can only recover those messages, encoded with an index he knows while all other indices stay hidden. The index-hiding property of an index-based message encoding is given by the following definition.

Definition 5 (Index-Hiding Message Encoding (IHME) [36]). *Let* $\texttt{IHME} = (\texttt{iEncode}, \texttt{iDecode})$ *denote a correct index-based message encoding scheme over index space* $\mathcal{I}$ *and message space* $\mathcal{M}^{\texttt{IHME},r}$*. Let* $b \in_R \{0, 1\}$ *be a randomly chosen bit and let* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ *be a PPT adversary. We say that* $\texttt{IHME}$ *provides* index-hiding *if there exists a negligible function* $\varepsilon(\cdot)$ *such that:*

$$\mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME},\mathcal{A}}(\lambda) := |\Pr[\mathsf{Exp}^{\mathsf{ihide},0}_{\mathrm{IHME},\mathcal{A}}(\lambda) = 1] - \Pr[\mathsf{Exp}^{\mathsf{ihide},1}_{\mathrm{IHME},\mathcal{A}}(\lambda) = 1]| \leq \varepsilon(\lambda).$$

Moreover, if $\mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME},\mathcal{A}}(\lambda) = 0$ *for all* λ*, the* IHME*-scheme is called* perfect*.*

$\mathsf{Exp}^{\mathsf{ihide},b}_{\mathrm{IHME},\mathcal{A}}(\lambda)$ *: Let* $(I_0, I_1, M', St) \leftarrow \mathcal{A}_1(1^\lambda)$ *with* $I_0, I_1 \subseteq \mathcal{I}$*,* $|I_0| = |I_1| = n$*,* $M' = (m'_1, \ldots, m'_{|I_0 \cap I_1|})$*,* $m'_j \in \mathcal{M}^{\texttt{IHME},r}$ *and* $\{i_1, \ldots, i_r\} = I_b \setminus I_{1-b}$*. Choose* $m_1, \ldots, m_r$ *uniformly at random from* $\mathcal{M}^{\texttt{IHME},r}$*, execute* $\mathcal{S} \leftarrow \texttt{iEncode}(\{(i_j, m'_j) | i_j \in I_0 \cap I_1\} \cup \{(i_j, m_j) | i_j \in I_b \setminus I_{1-b}\})$ *and* $b' \leftarrow \mathcal{A}_2(St, \mathcal{S})$*, and return* $b' = b$*.*

ν-fold IHME There exists an optimized version of IHME for longer messages, denoted ν-fold IHME [37] which has mainly two advantages. The original instantiation of IHME requires that the index and message space correspond, i.e. $\mathcal{I} = \mathcal{M}^{\mathrm{IHME},r} = \mathbb{F}$. Regarding the fact that the password (even the hashed one) is most likely significantly shorter than a PAKE message, the constraint $\mathcal{I} = \mathcal{M}^{\mathrm{IHME},r}$ seems impractical. Furthermore, ν-fold IHME is significantly faster than the original IHME implementation, as shown in [37]. It also allows us to encode distinct components of the same protocol message (cf. Section 3.1).

Definition 6 (ν-fold IHME [37]). *For an arbitrary finite field* $\mathbb{F}$ *and* $\nu \in \mathbb{N}$*, after setting* $\mathcal{I} = \mathbb{F}$ *and* $\mathcal{M}^{\mathrm{IHME},r} = \mathbb{F}^\nu$*, an index-hiding message encoding scheme* $\texttt{IHME} = (\texttt{iEncode}, \texttt{iDecode})$ *with index space* $\mathcal{I}$ *and message space* $\mathcal{M}^{\mathrm{IHME},r}$ *is constructed from standard* $\texttt{IHME}' = (\texttt{iEncode}', \texttt{iDecode}')$ *over* $\mathbb{F}$ *as follows:*

$\mathcal{S} \leftarrow \texttt{iEncode}(\mathcal{P})$**:** *On input of* $\mathcal{P} = \{(i_1, (m_{1,1}, \ldots, m_{1,\nu})), \ldots, (i_n, (m_{n,1}, \ldots, m_{n,\nu}))\} \subseteq \mathcal{I} \times \mathcal{M}^{\mathrm{IHME},r} = \mathbb{F} \times \mathbb{F}^\nu$*, the encoding is defined as the list* $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\nu)$ *of* IHME'*-encodings* $\mathcal{S}_k = \texttt{iEncode}'(\{i_j, m_{j,k}\}_{1 \leq j \leq n})$*, for* $1 \leq k \leq \nu$*.*

$m \leftarrow \texttt{iDecode}(\mathcal{S}, i)$**:** *On input of an encoding* $\mathcal{S} = (\mathcal{S}_1, \ldots, \mathcal{S}_\nu)$ *and an index* $i \in \mathcal{I}$*, this algorithm outputs a message* $(m_1, \ldots, m_\nu) = m \in \mathcal{M}^{\mathrm{IHME},r}$ *where* $m_k = \texttt{iDecode}'(\mathcal{S}_k, i)$*, for* $1 \leq k \leq \nu$*.*

B Proof of Theorem 1

At first, we want to stress that the existence of the admissible encodings for client messages implies the uniform distribution of these messages, output by $\Pi.\texttt{next}$ over $\mathcal{M}_C^r$. Thus, the uniform distribution of the messages output by the clients $\Pi.\texttt{next}$ in the according message space is a compulsory condition. Let w.l.o.g. P denote a client and P' a server. We first specify the conditions under which an instance P_i is *qualified* to be used in a Test session. For an instance P_i with $\texttt{pid}_P^i = P'$ to be *qualified* for a Test query the following must hold: (i) (P, P') must not be marked as corrupt, (ii) $\mathsf{Test}(P', j)$ has not been asked before for the partnered instance P'_j. Furthermore, we distinguish between two kinds of messages: *adversarial generated messages* and *forwarded messages*. *Adversarial generated messages* are generated by the adversary himself, i.e. have never been output by an oracle. *Forwarded messages* in contrast are outputs of oracle queries. We define experiments $\mathsf{Exp}_0, \ldots, \mathsf{Exp}_4$, where the original experiment $\mathsf{Exp}^{\mathsf{AKE\text{-}Sec}}_{C_\Pi, \mathcal{A}}$ corresponds to Exp_0. Let $\mathsf{Adv}_{i,\mathcal{A}}$, $\mathsf{Succ}_{i,\mathcal{A}}$ denote the advantage resp. success of $\mathcal{A}$ in the i-th experiment Exp_i and Sim the simulator that simulates the oracles in the experiments. All changes we introduce in the experiments are done for qualified Test sessions. Every other session has to be simulated honestly or in accordance to previous Test queries as the adversary would recognise the experiment change due to his knowledge of the password or the key.

On high level the experiments first replace the key computation done by Π with choosing a random key. This first step is possible since the possibility of an adversary against the AKE-security of Π to distinguish between randomly chosen keys and correct computed keys is bounded according to the AKE-security of Π. Subsequently, we replace the client messages output by $\Pi.\texttt{next}$ with randomly chosen messages from the appropriate IHME message space. The second step relies on the existence of an appropriate admissible encoding. In the next experiment we replace certain passwords in the clients password vector with random ones. Based on the index-hiding of IHME this step introduces a gap according to the advantage of the index-hiding adversary.[4] Since all messages and keys are password independent now, we have to ensure that the key confirmation and derivation steps do not offer any attack possibilities. This is assured by the used pseudorandom function. In the following we describe the experiments and their impacts in detail.

The AKE-security of Π ensures that it is not possible to distinguish between randomly chosen keys $\texttt{k}' \in_R \mathcal{K}_\Pi$ and correct computed keys $\texttt{k}$ output by Test_Π oracles. Therefore, we change the experiment as follows.

Exp_1: The key $\texttt{k}_\mathcal{A} \in \mathcal{K}_{C_\Pi}$ returned by the Test oracle in a qualified test session is computed as follows. Instead of the correct key $\texttt{k}$, computed by Π, a randomly chosen key $\texttt{k}' \in_R \mathcal{K}_\Pi$ is used to compute $\texttt{k}_\mathcal{A}$ in the case of $b = 1$.

Claim (1). $|\mathsf{Succ}_{0,\mathcal{A}} - \mathsf{Succ}_{1,\mathcal{A}}| \leq c \cdot \mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}'}$

Proof. The claim follows directly from the AKE security of the used PAKE protocol Π and Lemma 1. Sim can use $\mathcal{A}'$ to distinguish between real keys and randomly chosen ones and thus break the AKE security of Π. Therefore, the simulator interacts with c AKE experiments of Π on c different instances of P. Each experiment has a secret bit b for Test queries. Sim also chooses a random bit b' to use in Test query generation and thus simulates Exp_0 and Exp_1. The $\mathsf{Corrupt}$ queries can be answered by calling $\mathsf{Corrupt}_\Pi$ on one of the c experiments. Sim answers Send oracle queries of $\mathcal{A}'$ by forwarding the queries to all c experiments and combining their answers according to the O-PAKE compiler except that for all but one message distinct random passwords are used to create the IHME structure. If a Send_Π query finishes Π, i.e. all r messages are sent, Sim additionally calls Test_Π queries on all c experiments if Test has not been asked to the respective partnered instance. Thus, Sim is able to answer the remaining Send_{C_Π} query as well as a potential Test_{C_Π} query. $\mathsf{Execute}_{C_\Pi}$ oracles can be simulated by using the $\mathsf{Execute}_\Pi$ and Test_Π oracles of the c experiments and the same construction as above. Test_{C_Π} queries are answered with a key according to the chosen bit b', i.e. with $\texttt{k}_f$ derived from a randomly chosen key $\texttt{k} \in_R \mathcal{K}_\Pi$ for $b' = 0$ or derived from the according $\texttt{k}$ returned by a Test_Π oracle answer to a random experiment for $b' = 1$. On $b'' \leftarrow \mathcal{A}'$, Sim returns 1 to all experiments

[4] Note that the used IHME implementation is perfectly hiding such that the gap is actually zero.

if $b'' = b'$, else 0. Here, the factor c comes into play as it is sufficient that the simulator wins in one of the c experiments. Let p_1 denote the probability that Sim outputs 1. We can see that p_1 can be reduced to the success of $\mathcal{A}'$ guessing the correct bit. For the Test_{C_Π} oracles that returned real keys, p_1 is equal to the success of $\mathcal{A}'$ in Exp_0. For the Test_{C_Π} oracles that returned random keys, p_1 is equal to the success of $\mathcal{A}'$ in Exp_1 that uses only random keys. □

Note that we assumed a perfect index-hiding so far. Thus, the adversary has no advantage through the message format in C_Π, i.e. he can not recover anything from the sent passwords. To formally proof that this assumption is justified we define the next two experiments. Let ϵ_F denote the advantage of an adversary against the used ϵ-admissible encoding.

Exp_2: We modify the $\mathsf{Execute}$ and Send oracle for qualified Test sessions such that $\mathsf{Execute}$ returns a transcript $\mathtt{trans} = [m^0_P, m^0_{P'}, m^1_P, m^1_{P'}, \dots]$ and Send a messages m_{out} generated as follows. Instead of the protocol-conform encoded message $\mathcal{I}_{F^r}(m)$, a randomly chosen message $m' \in_R \mathcal{M}^{\mathtt{IHME},r}$ is used for client messages. Server messages are computed honestly.

Claim (2). $|\mathsf{Adv}_{1,\mathcal{A}} - \mathsf{Adv}_{2,\mathcal{A}}| < c \cdot \epsilon_F$

Proof. The admissible encoding maps a uniformly distributed message $m \in \mathcal{M}^r_C$ into the IHME message space $\mathcal{M}^{\mathtt{IHME},r}$. According to our premises, F^r with its inverse $\mathcal{I}_{F^r}$ is an ϵ_F-admissible encoding such that $|\Pr[\mathsf{Exp}_1 = 1] - \Pr[\mathsf{Exp}_2 = 1]| \leq \epsilon_F(\lambda)$. In any other case Sim can use $\mathcal{A}$ to distinguish between an uniformly at random sampled $m' \in_R \mathcal{M}^{\mathtt{IHME},r}$ and $\mathcal{I}_{F^r}(m)$. □

Based on Exp_2 we show that the index-hiding property of $\mathtt{IHME}$ is necessary to prevent the adversary from obtaining the correct password $\text{pw}_{P,P'}$ and thus breaking the AKE security. This justifies the assumption above that $\mathcal{A}$ gains no advantage from the message format used in C_Π. We use the term of qualified passwords here similar to the qualified Test sessions. A password $\mathbf{pw}[i]$ from the clients password vector $\mathbf{pw}$ is qualified if no $\mathsf{Corrupt}(T, T')$ has been asked with $\text{pw}_{T,T'} = \mathbf{pw}[i]$.

Exp_3: All qualified client passwords from $\mathbf{pw}$ are replaced with randomly chosen passwords $\mathbf{pw}[i] \in_R \mathcal{D}$. Let $\mathbf{pw}_q \subseteq \mathbf{pw}$ with $|\mathbf{pw}_q| = x$ denote the vector of qualified passwords. Let $\mathbf{pw}[t] = \mathbf{pw}_q[t'] = \text{pw}_{P,P'}$ denote the correct password. Thus, the client messages returned by Send and $\mathsf{Execute}$ oracles are generated as

$$m_{\text{out}} = \mathtt{iEncode}(\{(\mathbf{pw}[1], m_1), \dots, (\mathbf{pw}[n-x], m_{n-x})\}$$

$$\cup\{(\mathbf{pw}'_q[1], m'_1), \dots, (\mathbf{pw}'_q[x], m'_x)\}),$$

with $\mathbf{pw}'_q[i] \in_R \mathcal{D}$ and $\mathbf{m}'_i \in_R \mathcal{M}^{\mathtt{IHME},r}$ for $i \in 1, \dots, x$ and $\mathbf{pw}[j], m_j$ for $j \in 1, \dots, n-x$ computed honestly. To ensure the correctness of the protocol, i.e. P and P' compute the same key, the server's password has to be changed according to the random passwords $\mathbf{pw}'_q[t']$. Computation of server messages, returned by the oracles is not changed. The final key computation in the case where only forwarded messages have been used is done protocol-conform. If adversarial generated messages have been injected, the final key is drawn uniformly at random from the key space as a correct key computation is not possible. Oracles in unqualified Test sessions answer according to their specification in Exp_2.

Claim (3). $|\mathsf{Adv}_{2,\mathcal{A}} - \mathsf{Adv}_{3,\mathcal{A}}| \leq c \cdot \sum_r \mathsf{Adv}^{\mathsf{ihide},r}_{\text{IHME},\mathcal{B}}$

Proof. The gap between Exp_2 and Exp_3 is bounded by the the sum over the advantage of the index-hiding adversary $\mathcal{B}$ of each round, multiplied by c, the maximum number of passwords in one O-PAKE session. Let $\mathcal{B} = (\mathcal{B}_1, \mathcal{B}_2)$ denote the adversary in the ihide experiment against the index-hiding of $\mathtt{IHME}^r$. We define $\mathcal{B}_1$ as follows: Choose $\text{pw}_1, \dots, \text{pw}_n, \text{pw}'_1, \dots, \text{pw}'_x \in_R \mathcal{D}$ and $m_1, \dots, m_{n-x} \in_R \mathcal{M}^{\text{IHME},r}$, and initialise P_i with $\mathtt{role} = \mathtt{client}$, $\mathbf{pw}_P = (\text{pw}_1, \dots, \text{pw}_n)$ and partner P' with $\text{pw}_{P',P} = \text{pw}_t$. The output is generated as $I_0 = \mathbf{pw}_P$, $I_1 = \mathbf{pw}'_P$ with $\mathbf{pw}'_P = \mathbf{pw}_P \setminus \mathbf{pw}_q \cup (\text{pw}'_1, \dots, \text{pw}'_x)$, $\mathtt{St} = (t, \mathbf{pw}'_P, \mathbf{pw}_P, P_i, P', M')$ and $M' = \{m_1, \dots, m_{n-x}\}$ where m_i for $i \in 1, \dots, n-x$ are computed honestly according to the protocol

specification of Π. The intersection $\omega = I_b \setminus I_{1-b}$ is either $\omega = \mathbf{pw}_q$ for $b = 0$, or $\omega = (\mathrm{pw}'_1, \ldots, \mathrm{pw}'_x)$ for $b = 1$. The message $m'_i \in_R \mathcal{M}^{\mathrm{IHME},r}$ for $i \in 1, \ldots, x$ are chosen and encoded as

$$S \leftarrow \mathtt{IHME}^r.\mathtt{iEncode}(\{(\mathrm{pw}_1, m_1), \ldots, (\mathrm{pw}_{n-x}, m_{n-x})\}$$

$$\cup\{(\omega[1], m'_1), \ldots, (\omega[x], m'_x)\}),$$

according to the index-hiding game. As $\mathtt{IHME}^r$ is index-hiding according to Definition 5 the advantage of $\mathcal{B}$ to distinguish between Exp_2 and Exp_3 in round r is given as $c \cdot \mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME},\mathcal{B}}$. □

The final key $\mathtt{k}$ output by the O-PAKE compiler is computed by applying an additional pseudorandom function $\mathtt{PRF}_{\mathtt{k}}$. To show the security of the additional confirmation and key derivation step we change the experiment as follows. Let $\mathsf{Adv}_{\mathtt{PRF}}$ denote the advantage of the adversary against the used PRF.

Exp_4: The key $\mathtt{k}_{\mathcal{A}}$ returned by the Test oracle and the confirmation message c is chosen uniformly at random from $\mathcal{K}_{C_\Pi}$ in qualified Test sessions.

Claim (4). $|\mathsf{Adv}_{3,\mathcal{A}} - \mathsf{Adv}_{4,\mathcal{A}}| \leq 2 \cdot \mathsf{Adv}_{\mathtt{PRF}}$

Proof. The key $\mathtt{k}'$ used for confirmation c and final key generation $\mathtt{k}$ is according to Exp_1 chosen uniformly at random and thus offers no attack possibilities. Furthermore, the probability to distinguish between randomly chosen $c, \mathtt{k}$ and $c \leftarrow \mathtt{PRF}_{k'}$ resp. $\mathtt{k} \leftarrow \mathtt{PRF}_{k'}$ is bounded by $\mathsf{Adv}_{\mathtt{PRF}}$ as long as the used $\mathtt{PRF}$ is a pseudorandom function. □

The client messages in Exp_4 are password independent and thus can not leak any information about the passwords. Furthermore, the key $\mathtt{k}_{\mathcal{A}}$ is chosen uniformly at random. Thus, the advantage of $\mathcal{A}$ is given as

$$\mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{C_\Pi,\mathcal{A}}(\lambda) \leq c \cdot (\mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}'}(\lambda) + \sum_r \mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME},\mathcal{B}} + \epsilon_F) + 2 \cdot \mathsf{Adv}_{\mathtt{PRF}},$$

and therefore

$$\mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{C_\Pi,\mathcal{A}}(\lambda) \leq \frac{c \cdot \mathcal{O}(t)}{|\mathcal{D}|} + \varepsilon(\lambda).$$

□