Oblivious PAKE: 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. In this work we introduce the notion of *Oblivious Password based Authenticated Key Exchange* (O-PAKE) and a general compiler to transform a large class of PAKE into O-PAKE protocols. O-PAKE allows a client that shares *one* password with a server to use a *set of passwords* within one PAKE session. It 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 any password, input by the client, is made available to the server. Using special processing techniques, our O-PAKE compiler reaches nearly constant runtime on the server side, independent of the size of the client's password set. 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 PAKE protocols must satisfy in order to suit the compiler and give a concrete O-PAKE instantiation. The compiled protocol is implemented and its performance analysis attests to the practicality of the compiler. Furthermore, we implement a browser plugin demonstrating how to use O-PAKE in practice.

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 [24]). 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 [26, 27] show that every user has 6.5 passwords on average, used on 25 different websites. Approximately 2.4 failed computational login attempts are performed 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 trials are performed through the same secure channel.

The notion of *Password-based Authenticated Key Exchange (PAKE)*, introduced in [12], initially formalized in [10, 16], and later explored in numerous further works (cf. Section

1.2), 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 parallelized) 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 suitable 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 $\mathbf{pw}$ of $n \in [1, c]$ passwords from some dictionary while the input on the server-side is limited to a single password pw. Note that this does *not* increase the probability for offline dictionary attacks because the server still limits the number of trials and the

maximum number of passwords c input by the client. Note that the client is still allowed to input less than c passwords, e.g., only one if the client is confident about its validity. The protocol execution of O-PAKE succeeds if and only if the server's password pw is part of the client's password set **pw**. We utilise the well-known (game-based) PAKE model by Bellare, Pointcheval, and Rogaway [10] in its (stronger) Real-or-Random flavour from [6] and update it to match the O-PAKE setting by considering **pw** as input on the client side. In this model passwords are assumed to be distributed uniformly at random. In practice, a client that uses passwords with different strengths in the same O-PAKE session would obviously lower the overall security to the least secure password.

The crucial idea behind our transformation is to let each client execute n sessions of some given 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.

To exemplify instantiations of O-PAKE we implement the generic compiler and apply it on the secure, random-oracle-based SPAKE protocol from [7].[1] Based on real-world parameters from the aforementioned studies in [26, 27] we analyse the actual performance of our implementation and compare it with the naïve approach of repeating the PAKE protocol n times. Another practical component of this work is the O-PAKE implementation in a browser plugin to demonstrate an example of use.

1.2 Related Work

The concept of PAKE was introduced by Bellovin [12] and corresponding security models were initially developed by Bellare, Pointcheval, and Rogaway [10], Boyko, MacKenzie, and Patel [16], and Goldreich and Lindell [30]. 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) [28, 29, 33]. The only (but fairly inefficient) protocol that is built from general secure multi-party computation techniques but does not require any setup assumptions nor random oracles is due to Goldreich and

[1] A second instantiation using a secure, standard model, CRS-based PAKE framework of [28] can be found in Appendix D.

Lindell [30], which was subsequently simplified at the cost of weakened security in [40]. A stronger PAKE model in the framework of Universal Composability (UC) [19] is due to Canetti, Halevi, Katz and Lindell [20]. 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 [23] have also been used. In general, all PAKE models (see [41] for a recent overview) take into account unavoidable online dictionary attacks and aim to guarantee security against offline dictionary attacks. While many PAKE constructions require a constant number of communication rounds [7,28,29,33,34]; recent frameworks by Katz and Vaikuntanathan [34] and Benhamouda et al. [13] offer optimal one-round PAKE.

There exist further PAKE flavours, including three-party protocols [3,6] and group protocols [1,8]. Furthermore, 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 [14,18] 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 and our technique could possibly be used in there contexts as well.

2 Oblivious PAKE Model

In this section we recall the PAKE security model from [6], tailored to the needs of O-PAKE. The security model for O-PAKE protocols is in the multi-user setting and utilizes the Real-or-Random approach for AKE-security from [6,10]. 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 $\mathbf{pw}$ 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. 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 class of PAKE protocols.

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:

- $\texttt{pid}_P^i$ is the partner id of P_i, defined upon initialisation, subject to following restriction: if $P_i \in \mathcal{C}$ then $\texttt{pid}_P^i \in \mathcal{S}$, and if $P_i \in \mathcal{S}$ then $\texttt{pid}_P^i \in \mathcal{C}$.
- $\texttt{sid}_P^i$ 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. $\texttt{sid}_P^i$ is thus updated on each sent or received protocol message.
- $\texttt{k}_P^i$ is the value of the session key of instance P_i, which is initialised to $\texttt{null}$.
- $\texttt{state}_P^i$ 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 $\texttt{client}$ or a $\texttt{server}$.

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 $\texttt{match}$ 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 We define O-PAKE using an initialisation algorithm $\texttt{init}$ and a stateful interactive algorithm $\texttt{next}$, 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* $\texttt{next}$*, 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* $\texttt{par}$*, the algorithm initialises a new instance* P_i *with the internal O-PAKE state information* $\texttt{state}$*, defines the intended partner id as* $\texttt{pid}_P^i = P'$ *and session key* $\texttt{k}_P^i = \texttt{null}$*, and stores protocol parameters* $\texttt{par}$*. The* $\texttt{role}$ *indicates whether the participant acts as* $\texttt{client}$ *or* $\texttt{server}$*.*

$(m_{\text{out}}, \texttt{k}_P^i) \leftarrow \texttt{next}(m_{\text{in}})$**:** *On input* $m_{\text{in}} \in \mathcal{M}_{[S,C]}^r \cup \emptyset$ *with implicit access to internal* $\texttt{state}$*, 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* $\texttt{null}$*. 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* $\texttt{next}$ *implicitly updates the internal* $\texttt{state}$ *prior to each output and sets* $\texttt{used}$ *to* $\texttt{true}$*.*

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 $\texttt{k}^i_P \in \mathcal{K}$ and $\texttt{k}^i_P = \texttt{k}^j_{P'}$.

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

- If a pair $(P, \texttt{pid}^i_P)$ or $(\texttt{pid}^i_P, P)$ was corrupted (cf. definition of $\mathsf{Corrupt}$ oracle) while $\texttt{k}^i_P = \texttt{null}$, then abort. Note that this prevents $\mathcal{A}$ from obtaining $\text{pw}_{P,P'}$ 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 $\texttt{k}^i_P \in \mathcal{K}$ then if $b = 1$, return $\texttt{k}^i_P$, else if $b = 0$, return a randomly chosen element from $\mathcal{K}$ and store it for later use.
- Else return $\texttt{k}^i_P$. Note that in this case $\texttt{k}^i_P$ is either $\perp$ or `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* $c \in \mathbb{N}, b \in_R \{0,1\}$, *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. 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. [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. 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 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 [6, 10]).

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.

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)* [36,37]. 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 $(i_1, m_1), \ldots, (i_n, m_n)$ 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 the original work or Appendix B 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). 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* [15, 17, 25]. 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 [17, 25] or Appendix A for more details. $\mathcal{I}_F$ enables us to map PAKE messages into the IHME message space where necessary. In Section 3.2 we will discuss suitable PAKE message spaces and their admissible encodings offering compatibility with the message space $\mathbb{F}$ of the IHME schemes from [36, 37].

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 $\mathbf{pw}$, 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 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 each PAKE round r it 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. 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 Appendix C. Further note that we consider PAKE protocols with implicit authentication as explicit authentication may break the O-PAKE compilers security.

The $C_\Pi.\texttt{next}$ algorithm on the client side computes corresponding PAKE round messages for all passwords in **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 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 $C_\Pi.\texttt{init}$ The algorithm makes n calls to $\Pi.\texttt{init}$, one for each password in **pw**, to generate corresponding $\texttt{state}$s 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' 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 **pw** are equivalent, 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.

Algorithm $C_\Pi.\texttt{next}$ We split between $C_\Pi.\texttt{next}$ specifications for clients (Algorithm 1a) and servers (Algorithm 1b) as they are significantly different. We write $\Pi_{\mathbf{pw}[i]}.\texttt{next}$ for the invocation of $\Pi.\texttt{next}$, where the protocol $\texttt{state}$ 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 $\texttt{k} \in \mathcal{K}_\Pi$ on the client side after processing the incoming round message, then the client expects the server to send a confirmation message in the next round. After checking the confirmation message, the final session key is derived. Computation of this confirmation message with subsequent key derivation on the server

side is specified in Lines 10 and 11. Note that the use of the pseudorandom function PRF ensures the independence of 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. Lines 12 and 13).

Remark 1 (Compiler Efficiency). The compiler offers several possibilities to save computation time. PAKE protocols with built-in key confirmation (e.g. [7]) 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. Further, 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. Those drawbacks can only be omitted by implementing a protocol specific version of the compiler. As we are initially interested in a general compiler for a large class of PAKE protocols we do not consider protocol-specific optimisations at this stage.

The overall AKE-security of the protocol generated with the O-PAKE compiler is established in Theorem 1 (with the formal proof given in Appendix E).

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

Proof (sketch). Note that existence of admissible encodings for client messages implies the uniform distribution over $\mathcal{M}_C^r$ of these messages, output by Π.next. Thus, the uniform distribution of the messages output by client's algorithm Π.next in the according message space is a necessary condition. In the following we sketch experiment hops from the AKE-security experiment to an experiment where the Test oracle always returns random keys such that the adversary can not do better than guessing, while losing essentially only the AKE-security of Π (we refer to Appendix E for a formal proof).

a C_Π.next(m_{in}) — Client

Input: m_{in}
Output: $(m_{\text{out}}, \mathtt{k})$
1: $P = \emptyset$
2: **for** $i = 1 \ldots n$ **do**
3: **if** $\Pi[i]$ has not finished **then**
4: $(m'_{\text{out}}, \mathtt{k}) \leftarrow \Pi_{\mathbf{pw}[i]}.\mathtt{next}(m_{\text{in}})$
5: $P = P \cup \{(\mathbf{pw}[i], \mathcal{I}_{F^r}(m'_{\text{out}}))\}$
6: $\mathtt{state}[i].\mathtt{k} = \mathtt{k}$
7: **else if** $\Pi[i]$ has finished && $m_{\text{in}} = \mathtt{PRF}_{\Pi[i].key}(\mathtt{sid}^i_P, 0)$ **then**
8: $\mathtt{k} = \mathtt{PRF}_{\Pi[i].\mathtt{k}}(\mathtt{sid}^i_P, 1)$
9: **if** $P \neq \emptyset$ **then**
10: $m_{\text{out}} = \mathtt{IHME}^r.\mathtt{iEncode}(P)$
11: **else if** $\mathtt{k} \notin \mathcal{K}_{C_\Pi}$ **then**
12: $\mathtt{k} = \perp$
13: **return** $(m_{\text{out}}, \mathtt{k})$

b C_Π.next(m_{in}) — Server

Input: m_{in}
Output: $(m_{\text{out}}, \mathtt{k})$
1: **if** $\mathtt{k} = \mathtt{null}$ **then**
2: **if** $m_{\text{in}} \neq null$ **then**
3: $m \leftarrow \mathtt{IHME}^r.\mathtt{iDecode}(\text{pw}, m_{\text{in}})$
4: $m' = F^r(m)$
5: **else**
6: $m' = m_{\text{in}}$
7: $(m_{\text{out}}, \mathtt{k}) \leftarrow \Pi.\mathtt{next}(m')$
8:
9: **else if** $\mathtt{k} \in \mathcal{K}_\Pi$ **then**
10: $m_{\text{out}} = \mathtt{PRF}_k(\mathtt{sid}^j_{P'}, 0)$
11: $\mathtt{k} \leftarrow \mathtt{PRF}_{\mathtt{k}}(\mathtt{sid}^j_{P'}, 1)$
12: **else**
13: $(m_{\text{out}}, \mathtt{k}) = (\emptyset, \perp)$
14: **return** $(m_{\text{out}}, \mathtt{k})$

Fig. 1: C_Π.next Algorithms

First, we replace the client messages output by Π.`next` with randomly chosen elements from the appropriate IHME message space. This is possible due to the existence of an appropriate admissible encoding. We further replace passwords from uncorrupted client-server pairs in the clients password vector **pw** with randomly chosen passwords. Based on the index-hiding property of IHME this step introduces a gap upper bounded by the advantage of the index-hiding adversary. Note that since we use an IHME implementation, which is perfectly hiding this gap is essentially zero. Since all messages become password-independent we can replace the key computation step in Π with a randomly chosen key. This step is bounded by the advantage of the AKE-adversary against Π. Finally, we have to ensure that the key confirmation and derivation steps do not offer any attack possibilities by making the confirmation message and final session key independent of bit b. This step is bounded by the advantage of an adversary against the PRF used to generate the confirmation message and final session key. Eventually, the information on the secret bit b, extractable by the adversary in the resulting experiment is zero, which means that the adversary can only win this final experiment by guessing bit b. □

3.2 Oblivious PAKE Instantiation

An AKE-secure PAKE protocol Π is suitable for our O-PAKE transformation if its client messages are uniformly distributed in respective round-dependent message spaces 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 3 (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. [25, Lemma 12])
(2) The set of quadratic residues modulo safe primes p, i.e. $R = QR(p) \subseteq \mathbb{Z}_p^\times$. (cf. [25, Lemma 12])
(3) Arbitrary subgroups $G \subseteq \mathbb{Z}_p^\times$ of prime order q. (cf. [25, Lemma 12])
(4) The set $R = E(\mathbb{F})$ of rational points on (certain) elliptic curves, defined over a finite field (cf. [17]).

4 Concrete Instantiation Example – Oblivious SPAKE

To show the practicality of the O-PAKE compiler we implement it in `C++`, and demonstrate how the compiler can be applied to implementations of the AKE-secure, random-oracle-based SPAKE protocol from [7]. In order to show potential use of the compiled O-PAKE protocol in practice we further implement a browser plugin for the Firefox browser and a server module for the Apache web server (cf. Section 4.1).

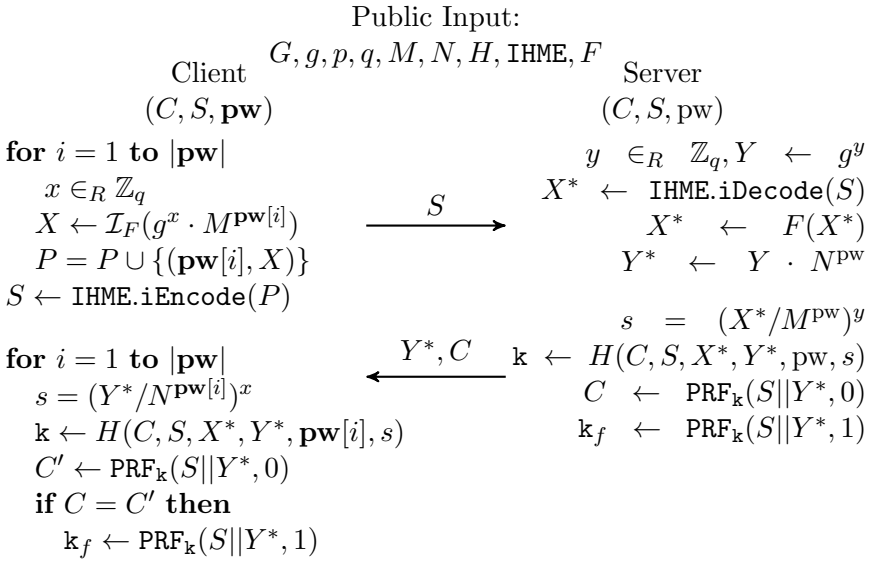

Fig. 2: Oblivious SPAKE (O-SPAKE)

Implementation of the O-PAKE Compiler The O-PAKE compiler is implemented as an abstract class with virtual `init` and `next` functions, using implementations of admissible encodings and IHME. Despite the virtual functions the compiler provides everything necessary to implement an O-PAKE protocol from a suitable PAKE protocol. It initializes and stores all c PAKE instances and offers `next` functions according to Figure 1. 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 compilers modular design makes it easy to instantiate new 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 get an instance of the O-PAKE compiler.

Implementation of the O-SPAKE Protocol The resulting O-SPAKE is specified in Figure 2 and involves steps of the original SPAKE protocol from [7, Sec. 5], which is a secure variant of [12], whose security has been proven in the random oracle model.[2] 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 $X^* \leftarrow g^x \cdot M^{\mathrm{pw}}, x \in_R \mathbb{Z}_q$ and the server responds with $Y^* \leftarrow g^y \cdot N^{\mathrm{pw}}, y \in_R \mathbb{Z}_q$.[3] The algorithm `next` computes an intermediate value s and derives the session key as $\mathtt{k} \leftarrow H(P, P', X^*, Y^*, \mathrm{pw}, s)$. 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 or prime-order subgroups of elliptic curve points $E(\mathbb{F})$ in which

[2] Note that the very similar SOKE protocol from [2] can also be used in the O-PAKE compiler following the here given description of O-SPAKE.

[3] The actual order of these messages does not matter since they are independent

the CDH problem is believed to be hard. We can apply the admissible encodings (3) and (4) from Definition 3 due to the fact that client's SPAKE message $X^* = g^x \cdot M^{\text{pw}}$ is uniformly distributed in G, given the uniformity of $x \in \mathbb{Z}_q$. We use the 2048 bit prime-order subgroup G from RFC 3526 [35] in which the CDH problem is assumed to be hard for SPAKE. We implemented the resulting SPAKE and oblivious SPAKE protocols using the Botan cryptographic library (`http://botan.randombit.net`).

Admissible Encodings for SPAKE We use admissible encodings (1) and (3) from Definition 3 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 [25, Lemma 12] with prime $|q'| = \ell(\lambda) > 2|N|$ to meet IHME requirements.

Confirmation and Key Derivation Steps The key confirmation and derivation steps from algorithm C_{Π}.`next` are implemented using CBC-MAC [31, Sec. 4.5] with AES-256 [31, Sec. 5.5] to instantiate the pseudorandom function `PRF`. The message C is sent to the client, such that he can identify the "right" PAKE session and derive the final session key $\mathtt{k}_f$.

Performance Analysis 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 does not 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.

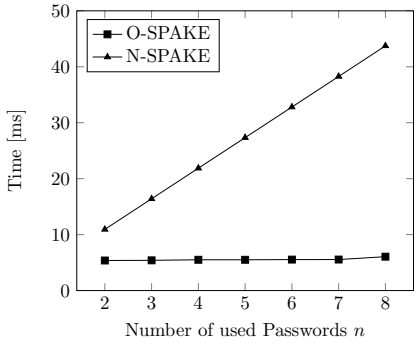

Fig. 3: Oblivious SPAKE – Performance

We observe that the encoding process applied to client messages increases the length of its group element to 4096 bits due to the bound on function ℓ in the description of the admissible encoding. Note that this increase comes from the application of admissible encoding and not from IHME, which is length-preserving according to [36]. The server messages in contrast, 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 protocol and its comparison with the naïve execution involving multiple independent SPAKE 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 until the derivation of the session key.

Fig. 4: Login Browser Bars

The original SPAKE protocol requires roughly 5ms for the client and server. Figure 3 illustrates server-side timings measured for our O-SPAKE implementation and compares them with the performance of the naïve protocol, denoted N-SPAKE, with linear number of PAKE sessions on the server side. The exact measurements are provided in Table 1 in Appendix F. Observe that the server runtime for protocols obtained through our O-PAKE compiler is almost independent from the number of used client passwords while the naïve implementation experiences linear increase in the servers runtime. With roughly 5.5ms for O-SPAKE with two passwords, the compiled protocol is significantly faster than the naïve protocol, which needs roughly 10.8ms. 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).

4.1 O-PAKE in Practice

Based on the Firefox extension and PHP server code developed in [38] we implement a Firefox plugin and Apache web server module that allows to use O-PAKE on a website as a login mechanism. While the actual user interface does not significantly differ from the implementation in [38], which is focused on the standard PAKE functionality, the underlying architecture of our O-PAKE plugin and user experience is new. We extend the architecture from [38] with a native Firefox plugin and Apache module (implemented in `C++`) that allows us to use well established cryptographic libraries and our previously described implementation of O-PAKE. This implementation may be of independent interest as it allows to use a wide range of authentication mechanisms for website logins. The first notification bar in Figure 4 displayed on top of the website is equivalent to the one in [38] and appears only if the website supports the O-PAKE mechanism. After clicking "Login", the actual login window for O-PAKE is prompted. Figure 5 shows its user interface. In contrast to the original implementation the user has the possibility to use the "plus"-button to create up to $c = 3$ password input fields for his passwords. When clicking the "Login"-button the O-PAKE protocol is executed using the entered passwords on the client side and the stored one on the server side. The user is successfully logged-in if one of the entered passwords is correct and the browser bar with the green lock is shown (cf. second bar in Figure 4). The source code for the Firefox plugin and extension as well as a demo can be found at `https://130.83.239.102/zaiTa6Ooso/`.

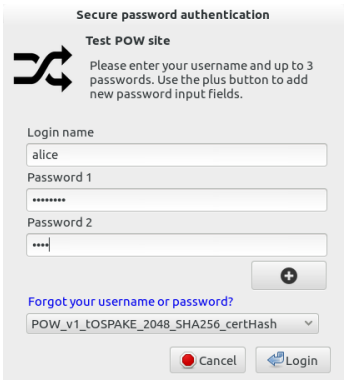

Fig. 5: Login Popup

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. It can be used with arbitrary PAKE protocols as input as long as their client 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 the widely used PAKE model from [6] and its practicality has been demonstrated experimentally through the implementation and compilation of an efficient PAKE constructions from [7] and the implementation in a browser plugin for practical use.

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*, pages 499–514. Springer-Verlag, 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. ACM, 2006.
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. Springer-Verlag, 2008.
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. Springer-Verlag, 2005.
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. Springer-Verlag, 2005.
7. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA'05*, pages 191–208. Springer-Verlag, 2005.
8. M. Abdalla and D. Pointcheval. A Scalable Password-Based Group Key Exchange Protocol in the Standard Model. In *ASIACRYPT'06*, pages 332–347. Springer-Verlag, 2006.
9. A. Bagherzandi, S. Jarecki, N. Saxena, and Y. Lu. Password-protected secret sharing. In *CCS'11*, pages 433–444. ACM, 2011.
10. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *EUROCRYPT'00*, pages 139–155. Springer-Verlag, 2000.
11. M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. In *CRYPTO'93*, pages 232–249. Springer-Verlag, 1994.
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. F. Benhamouda, O. Blazy, C. Chevalier, D. Pointcheval, and D. Vergnaud. New Techniques for SPHFs and Efficient One-Round PAKE Protocols. In *CRYPTO'13*, pages 449–475. Springer-Verlag, 2013.
14. O. Blazy, D. Pointcheval, and D. Vergnaud. Round-Optimal privacy-preserving protocols with smooth projective hash functions. In *TCC'12*, pages 94–111. Springer-Verlag, 2012.
15. D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. In *CRYPTO'01*, pages 213–229. Springer-Verlag, 2001.
16. V. Boyko, P. MacKenzie, and S. Patel. Provably Secure Password-Authenticated Key Exchange using Diffie-Hellman. In *EUROCRYPT'00*, pages 156–171. Springer-Verlag, 2000.
17. 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. Springer-Verlag, 2010.

18. J. Camenisch, N. Casati, T. Gross, and V. Shoup. Credential authenticated identification and key exchange. In *CRYPTO'10*, pages 255–276. Springer-Verlag, 2010.
19. R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *FOCS'01*, page 136. IEEE Computer Society, 2001.
20. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. MacKenzie. Universally Composable Password-Based Key Exchange. In *EUROCRYPT'05*, pages 404–421. Springer-Verlag, 2005.
21. 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, pages 13–25. Springer-Verlag, 1998.
22. 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. Springer-Verlag, 2002.
23. 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/.
24. T. Dierks and E. Rescorla. RFC 5246 - The Transport Layer Security (TLS) Protocol Version 1.2, aug 2008. Updated by RFCs 5746, 5878, 6176.
25. N. Fleischhacker, F. Günther, F. Kiefer, M. Manulis, and B. Poettering. Pseudorandom Signatures. In *ASIA CCS'13*, pages 107–118. ACM, 2013.
26. 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. ACM, 2007.
27. S. Gaw and E. W. Felten. Password Management Strategies for Online Accounts. In *Symposium on Usable privacy and security*, SOUPS'06, pages 44–55. ACM, 2006.
28. R. Gennaro. Faster and Shorter Password-Authenticated Key Exchange. In *TCC'08*, pages 589–606. Springer-Verlag, 2008.
29. R. Gennaro and Y. Lindell. A Framework for Password-Based Authenticated Key Exchange. *ACM Trans. Inf. Syst. Secur.*, 9(2):181–234, may 2006.
30. O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. In *CRYPTO'01*, pages 408–432. Springer-Verlag, 2001.
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*, pages 475–494. Springer-Verlag, 2001.
33. J. Katz, R. Ostrovsky, and M. Yung. Efficient and Secure Authenticated Key Exchange Using Weak Passwords. *J. ACM*, 57(1):3:1–3:39, nov 2009.
34. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC'11*, pages 293–310. Springer-Verlag, 2011.
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. Springer-Verlag, 2010.
37. M. Manulis and B. Poettering. Practical affiliation-hiding authentication from improved polynomial interpolation. In *ASIACCS'11*, pages 286–295. ACM, 2011.
38. M. Manulis, D. Stebila, and N. Denham. Secure modular password authentication for the web using channel bindings, August 2013. http://www.douglas.stebila.ca/research/papers/MSD13/.
39. National Institute of Standards and Technology. Federal Information Processing Standard 180-4 — Secure Hash Standard (SHS), 2012.
40. M.-H. Nguyen and S. P. Vadhan. Simpler Session-Key Generation from Short Random Passwords. In *TCC'04*, pages 428–445. Springer-Verlag, 2004.
41. D. Pointcheval. Password-based authenticated key exchange. In *PKC'12*, pages 390–397. Springer-Verlag, 2012.
42. A. Sahai. Non-Malleable Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. In *FOCS'99*, page 543. IEEE Comput. Soc, 1999.
43. A. D. Santis, G. D. Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-interactive Zero Knowledge. In *CRYPTO'01*, pages 566–598. Springer-Verlag, 2001.
44. V. Shoup. A Proposal for an ISO Standard for Public Key Encryption, 2001. sho@zurich.ibm.com 11676 received 20 Dec 2001.

A Admissible Encoding

Definition 4 (Admissible Encoding [17]). *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*.*

B 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.

Definition 5 (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 6 (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* 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* $I_0, I_1 \subseteq \mathcal{I}$*,* $|I_0| = |I_1| = n$*,* $M' = (m'_1, \ldots, m'_{|I_0 \cap I_1|})$*,* $m'_j \in \mathcal{M}^{\mathtt{IHME},r}$ *and* $\{i_1, \ldots, i_r\} = I_b \setminus I_{1-b}$*. Choose* $m_1, \ldots, m_r$ *uniformly at random from* $\mathcal{M}^{\mathtt{IHME},r}$*, execute* $\mathcal{S} \leftarrow \mathtt{iEncode}(\{(i_j, m'_j) | i_j \in I_0 \cap I_1\} \cup \{(i_j, m_j) | i_j \in I_b \setminus I_{1-b}\})$ *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] with two advantages. First, the original instantiation of IHME requires that the index and message space correspond, i.e. $\mathcal{I} = \mathcal{M}^{\mathrm{IHME},r} = \mathbb{F}$. Secondly, ν-fold IHME is significantly faster than the original IHME implementation, as shown in [37]. Furthermore, it provides us with an easy way to encode multiple elements of the same protocol message (cf. Appendix C).

Definition 7 (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* $\mathtt{IHME} = (\mathtt{iEncode}, \mathtt{iDecode})$ *with index space* $\mathcal{I}$ *and message space* $\mathcal{M}^{\mathrm{IHME},r}$ *is constructed from standard* $\mathtt{IHME}' = (\mathtt{iEncode}', \mathtt{iDecode}')$ *over* $\mathbb{F}$ *as follows:*

$\mathcal{S} \leftarrow \mathtt{iEncode}(\mathcal{P})$**:** *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* $\mathcal{S}_k = \mathtt{iEncode}'(\{i_j, m_{j,k}\}_{1 \le j \le n})$*, for* $1 \le k \le \nu$*.*

$m \leftarrow \mathtt{iDecode}(\mathcal{S}, i)$**:** *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* $m_k = \mathtt{iDecode}'(\mathcal{S}_k, i)$*, for* $1 \le k \le \nu$*.*

C Processing Multi-Component Messages

As mentioned before, message spaces of PAKE protocols often consist of Cartesian products of message spaces of the individual message components. In the following we discuss how the compiler proceeds in this case of multi-set messages.

First, we distinguish between two basic classes of message elements: constants and password depending elements (e.g. group elements or integers). Constants are password independent and thus do not have to be processed by the compiler. They can be sent directly. All other message components have to be encoded as described in the compiler. To encode messages of multiple elements we use ν-fold IHME. [37] introduces ν-fold IHME, which allows us to encode a list of ν message elements from the same finite field (cf. Appendix B). Therefore, we separate message elements from different finite fields into message element classes. Message elements 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. In order to process the elements, a loop over $m_1, \ldots, m_l$ adds $(\mathbf{pw}[i], \mathcal{I}_{F^{r,j}}(m_j))$ to the input set of $\nu-\mathrm{fold}-\mathrm{IHME}^r_j.\mathtt{iEncode}$ according to their message classes (finite fields). Likewise, the output message m_{out} of the `next` algorithm is the concatenation of the encoded message element classes. Upon receiving a client message m_{in}, the server has

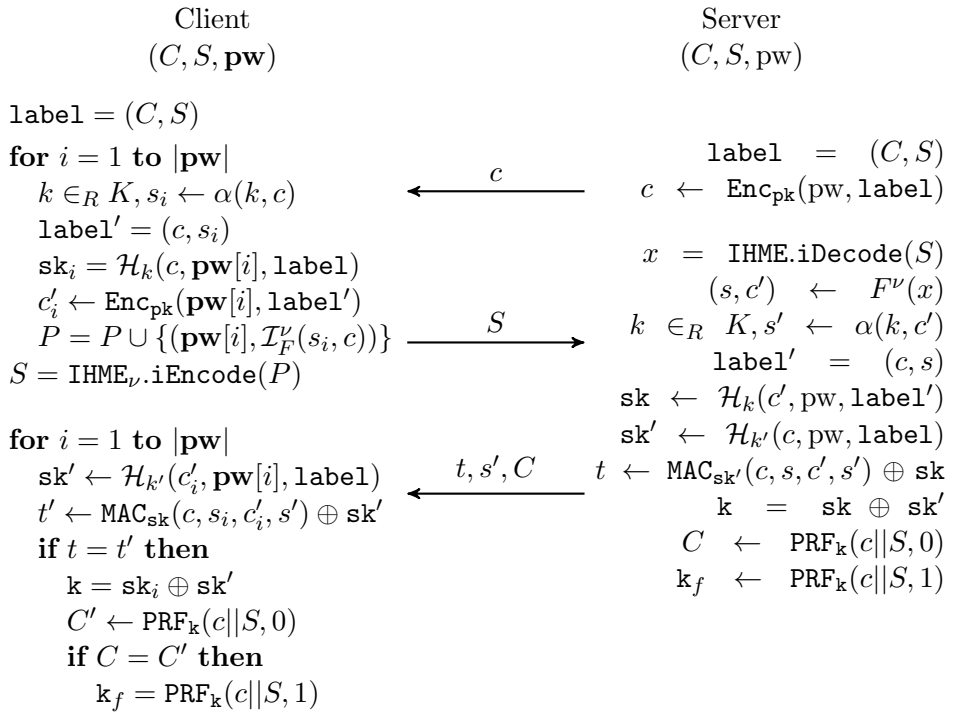

Fig. 6: Oblivious RG-PAKE

to decompose it to retrieve the IHME encoded messages. After decoding the message parts with $m_j \leftarrow \nu\text{−fold}-\mathtt{IHME}^r_j.\mathtt{iDecode}(\mathrm{pw}_{P,P'}, m^j_{\mathrm{in}})$ the original PAKE message of Π is reassembled by decoding messages $F^{r,j}(m_j)$.

Adopting this approach for multi-element messages, 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 element class that requires encoding). The corresponding index-hiding advantage will therefore be multiplied by l. The remaining proof stays unchanged.

D Instantiation – Oblivious RG-PAKE

In this section we present a second concrete instantiation of the O-PAKE compiler using the efficient PAKE framework proposed by Gennaro [28]. It is 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 [28] as RG-PAKE. The construction relies on a CCA-secure labelled encryption algorithm $\mathtt{Enc}_{\mathtt{pk}}(m, \mathtt{label})$ and a smooth projective hash function $(\alpha, \mathcal{H}_k)$. For an overview of these building blocks we refer to [28] and focus here on the protocol communication in the following.

The client initially computes the ciphertext $c \leftarrow \mathtt{Enc}_{\mathtt{pk}}(\mathrm{pw}, (P, P'))$ and sends it to the server that in turn replies with the projected value $s \leftarrow \alpha(k, c)$ and another ciphertext $c' \leftarrow \mathtt{Enc}_{\mathtt{pk}}(\mathrm{pw}, (c, s))$. This message is then parsed by the client that computes its final message,

consisting of the projected value $s' \leftarrow \alpha(k, c')$ and a MAC value $t \leftarrow \mathtt{MAC}_{\mathtt{sk}'}(c, s, c', s')$. Finally, the client derives the session key $\mathtt{k} = \mathtt{sk} \oplus \mathtt{sk}'$ with $\mathtt{sk} = \mathcal{H}_k(c', \mathrm{pw}_{P,P'}, (c, s))$ and $\mathtt{sk}' = \mathcal{H}_{k'}(c, \mathrm{pw}_{P,P'}, (P, P'))$. The server checks the received MAC value t and eventually computes the session key.

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. Additionally, all these message elements must have uniform distribution in the corresponding sets. As discussed in [22,28,29], 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 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 [28]. 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 and client swap their algorithms (which is possible since RG-PAKE has no strict separation into client/server roles), then no pseudorandomness of MAC values would be required, as they would become part of server's messages. Note that we use RG-PAKE this way in the following as it offers another useful property, we will see later.

In our implementation of Oblivious RG-PAKE (O-RG-PAKE) we rely on the DDH-based instantiation, and realize the encryption algorithm $\mathtt{Enc}$ using the CCA-secure labeled Cramer-Shoup encryption scheme [21, 44] and the smooth projective hash function $(\alpha, \mathcal{H}_k)$ using the scheme from [29, 32]. This leads to the CRS containing the public key $\mathtt{pk} = (b, d, h, g_1, g_2, q, H) = (g_1^{x_1} g_2^{x_2}, g_1^{y_1} g_2^{y_2}, g_1^z, g_1, g_2, q, H)$ 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.[4] Given the above instantiations, the ciphertext c and projected hash value s' in the resulting RG-PAKE protocol have the following form:

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

$s' \leftarrow \alpha(k, c, \mathtt{label})$: $s' = g_1^{k_1} g_2^{k_2} h^{k_3} (bd^\theta)^{k_4}$ 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 $\theta = H(u_1, u_2, e, \mathtt{label})$.

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

Figure 7 depicts the compiled RG-PAKE protocol O-RG-PAKE using $(\alpha, \mathcal{H}_k)$ and $\mathtt{Enc}_{\mathtt{pk}}$ as defined above. Note that we use $F^\nu, \mathcal{I}_F^\nu$ to denote the use of per-element admissible

[4] H is implemented using collision-resistant hash function SHA-256 [39].

encoding/decoding $F, \mathcal{I}_F$ on the ciphertext c and projection s. The admissible encoding F is realised according to the description in Definition 3 and Section 4. Likewise, we use $\texttt{IHME}_\nu$ from Definition 7 to encode the parts from c and s element by element. The implementation of O-RG-PAKE could be optimised by dropping the additional confirmation message C. The client could still compute the correct key $\texttt{k}_f$ using the MAC value t to identify the correct RG-PAKE session. However, this is not possible in the generic version.

Performance We highlight here only the protocol specific performance observations. For a more general discussion on the compiler efficiency and the used reference platform we refer to Section 4. We use prime-order groups specified in RFC 3526 [35] with 2048 bits. In our implementation of RG-PAKE client and server exchange 10 group elements altogether (five per party) and one MAC value. 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. Looking at the server side, the O-RG-PAKE protocol is with 143ms for two passwords significantly faster than the corresponding naïve protocol (282ms) and only slightly slower than the original RG-PAKE with one password that needs 141ms.

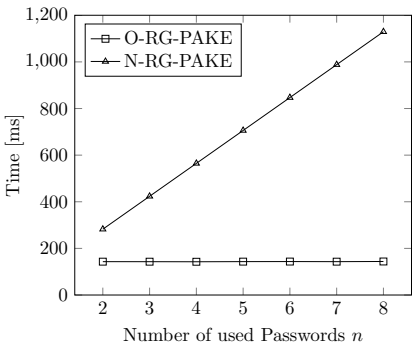

Fig. 7: O-RG-PAKE – Performance

Remark 2 (MACs vs. One-Time Signatures and NIZKs). PAKE frameworks from [29,33] used one-time signatures, whereas in the RG-PAKE framework [28] those were essentially replaced by MACs for better efficiency. Interestingly, unlike the RG-PAKE protocols that can be directly made oblivious using our compiler, the protocols from [29,33] 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 [29,33], if processed with our 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 [42,43]. In contrast, the more efficient MAC-based approach taken in RG-PAKE [28] doesn't admit public consistency checks of the resulting PAKE messages.

E Proof of Theorem 1

This is the formal proof of Theorem 1 explicitly defining the game hops, sketched below before. First note that the existence of the admissible encodings for client messages implies the uniform distribution of these messages over $\mathcal{M}_C^r$, output by $\Pi.\texttt{next}$. 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 $\mathtt{pid}_P^i = P'$ 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 oracle generated. We define experiments $\mathsf{Exp}_0, \ldots, \mathsf{Exp}_4$, where the original AKE-security experiment $\mathsf{Exp}^{\mathsf{AKE\text{-}Sec}}_{C_{\Pi},\mathcal{A}}$ corresponds to Exp_0 and Exp_4 contains only independent randomly distributed or adversarial known information. Let $\mathsf{Adv}_{i,\mathcal{A}}$ and $\mathsf{Succ}_{i,\mathcal{A}} = \Pr[\mathsf{Exp}_i = 1]$ denote the advantage resp. success of $\mathcal{A}$ in the i-th experiment Exp_i. All changes we introduce in the experiments are done for qualified Test sessions only. 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 key.

We first recall the high level intuition of the proof. The experiments start with replacing the client messages output by $\Pi.\mathtt{next}$ with randomly chosen messages from the appropriate IHME message space. This 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. Note that the used IHME implementation is perfectly hiding such that the gap is actually zero. Since all messages are password independent now we can replace the key computation done by Π with choosing a random key. This step introduces an adversarial advantage according to the AKE-security of Π. Finally, we have to ensure that the key confirmation and derivation steps do not offer any attack possibilities. However, this is assured by the used pseudorandom function. In the following we describe the experiments and the simulator more in detail.

Let ϵ_F denote the advantage of an adversary against the used ϵ-admissible encoding. We replace all admissible encoded messages by random elements from the according IHME message space.

Exp_1: We modify the $\mathsf{Execute}$ and Send oracle for qualified Test sessions such that $\mathsf{Execute}$ returns a transcript $\mathtt{trans} = [m^0_P, m^0_{P'}, m^1_P, m^1_{P'}, \ldots]$ 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 (1). $|\mathsf{Succ}_{0,\mathcal{A}} - \mathsf{Succ}_{1,\mathcal{A}}| < c \cdot \epsilon_F$

Proof. The claim follows from the definition of the used admissible encoding F^r. 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. It is easy to see that the difference of the success probability in Exp_0 and Exp_1 is bounded by the ϵ_F-admissible encoding property of F.

Assume an admissible encoding adversary $\mathcal{A}'$ that has an oracle to retrieve either random elements from $\mathcal{M}^{\mathtt{IHME},r}$ or $\mathcal{I}_{F^r}(m)$, depending on a secret bit b, $\mathcal{I}_{F^r}(m)$ for random $\mathbf{m} \in_R \mathcal{M}$ if $b = 0$ and random elements from $\mathcal{M}^{\mathtt{IHME},r}$ otherwise. The advantage of $\mathcal{A}'$ is given as $\mathsf{Adv}_{F,\mathcal{A}'} = |\Pr[\mathsf{Exp}_{0,F} = 1] - \Pr[\mathsf{Exp}_{1,F} = 1]|$, where $\Pr[\mathsf{Exp}_{b,F} = 1]$ denotes

the probability of $\mathcal{A}'$ to identify the given distribution b. By assumption $\mathsf{Adv}_{F,\mathcal{A}'} \leq \epsilon_F$ is negligible.

To answer the Test oracles, a bit $\hat{b}$ is randomly chosen by $\mathcal{A}'$ to either return random or real keys. $\mathcal{A}'$ simulates the oracles in the AKE-security according to the protocol specification, except the Send and $\mathsf{Execute}$ oracle, which are simulated using elements retrieved through his own oracle. Eventually, $\mathcal{A}$ outputs his guess for $\hat{b}$, $\hat{b}'$. $\mathcal{A}'$ just outputs $\hat{b}'$ as his guess b' for b if $\hat{b} = 1$ and chooses b' randomly otherwise. The success probability $\Pr[\mathsf{Exp}_{b,F} = 1]$ of $\mathcal{A}'$ in case b is given by $\Pr[\hat{b} = 0] \cdot \mathsf{Succ}_{b,\mathcal{A}} + 1/2 \cdot \Pr[\hat{b} = 1]$. This results in an advantage of

$$\begin{aligned}\mathsf{Adv}_{F,\mathcal{A}'} &= |\Pr[\mathsf{Exp}_{0,F} = 1] - \Pr[\mathsf{Exp}_{1,F} = 1]| \\ &= |\Pr[\hat{b} = 0] \cdot \mathsf{Succ}_{0,\mathcal{A}} + 1/2 \cdot \Pr[\hat{b} = 1] - \Pr[\hat{b} = 0] \cdot \mathsf{Succ}_{1,\mathcal{A}} - 1/2 \cdot \Pr[\hat{b} = 1]| \\ &= 1/2 \cdot |\mathsf{Succ}_{0,\mathcal{A}} - \mathsf{Succ}_{1,\mathcal{A}}|,\end{aligned}$$

and thus

$$\epsilon_F \geq 1/2 \cdot |\mathsf{Succ}_{0,\mathcal{A}} - \mathsf{Succ}_{1,\mathcal{A}}|.$$

Since we have c messages and therefore c times the possibility to distinguish real from encoded messages, we have to factor in c. Note that we omit the constant number of rounds in our advantage bounds. □

Based on Exp_1 we show now that the index-hiding property of $\mathtt{IHME}$ prevents the adversary from obtaining passwords pw from the exchanged messages and thus breaking the AKE security. 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 $\mathrm{pw}_{T,T'} = \mathbf{pw}[i]$.

Exp_2: All qualified client passwords from $\mathbf{pw}$ are replaced with randomly chosen passwords $\mathbf{pw}[i] \in_R \mathcal{D}$. To ensure consistency of oracle queries, the Test oracle returns the same key for Test queries on P and P' even though they may have calculated different keys due to the randomly chosen password used by one of them.

Claim (3). $|\mathsf{Succ}_{1,\mathcal{A}} - \mathsf{Succ}_{2,\mathcal{A}}| \leq \mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME},\mathcal{B}}$

Proof. To keep it simple we use a single protocol execution in the following proof. However, it can be easily extended to arbitrary number of oracle invocations of $\mathcal{A}$ by allowing the IHME adversary to play his game multiple times. The gap between Exp_1 and Exp_2 is bounded by the the sum over the advantage of the index-hiding adversary $\mathcal{B}$, multiplied by c, the maximum number of passwords in one O-PAKE session. Note that we omit the number of rounds again

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 $\mathrm{pw}_1, \ldots, \mathrm{pw}_n, \mathrm{pw}'_1, \ldots, \mathrm{pw}'_x \in_R \mathcal{D}$ and $m_1, \ldots, m_n \in_R \mathcal{M}^{\mathrm{IHME},r}$, and initialise P_i with $\mathtt{role} = \mathtt{client}$, $\mathbf{pw}_P = (\mathrm{pw}_1, \ldots, \mathrm{pw}_n)$ and partner P' with $\mathrm{pw}_{P',P} = \mathrm{pw}_t$. The output is generated as $I_0 = \mathbf{pw}_P$, $I_1 = \mathbf{pw}'_P$ with $\mathbf{pw}'_P = \mathbf{pw}_P \setminus \mathbf{pw}_q \cup (\mathrm{pw}'_1, \ldots, \mathrm{pw}'_x)$, $\mathtt{St} = (t, \mathbf{pw}'_P, \mathbf{pw}_P, P_i, P', M')$ and $M' = \{m_1, \ldots, m_{n-x}\}$. 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 messages $m'_i \in_R \mathcal{M}^{\mathrm{IHME},r}$ for $i \in 1, \ldots, x$ are chosen and encoded as

$$S \leftarrow \mathtt{IHME}^r.\mathtt{iEncode}(\{(\mathrm{pw}_1, m_1), \ldots, (\mathrm{pw}_{n-x}, m_{n-x})\} \cup \{(\omega[1], m'_1), \ldots, (\omega[x], m'_x)\})$$

by the index-hiding game. Thus, the index-hiding adversary interpolates between Exp_1 and Exp_2 such that $\Pr[\mathsf{Exp}_{b,\mathtt{IHME}} = 1] = \mathsf{Succ}_{b+1,\mathcal{A}}$, and therefore

$$\begin{aligned}\mathsf{Adv}_{\mathsf{ihide},\mathcal{B}} &= |\Pr[\mathsf{Exp}_{0,\mathtt{IHME}} = 1] - \Pr[\mathsf{Exp}_{1,\mathtt{IHME}} = 1]| \\ &= |\mathsf{Succ}_{1,\mathcal{A}} - \mathsf{Succ}_{2,\mathcal{A}}|.\end{aligned}$$

□

As all messages are random and password independent now we can replace the session key calculated in Π with a random one. The AKE-security of Π ensures that it is impossible to distinguish between randomly chosen keys $\mathtt{k}' \in_R \mathcal{K}_\Pi$ and correct computed keys $\mathtt{k}$ output by Test_Π oracles with adversarial advantage $\mathcal{O}(t)/|\mathcal{D}| + \varepsilon(\lambda)$. We change the experiment as follows.

Exp_3: The key $\mathtt{k}_\mathcal{A} \in \mathcal{K}_{C_\Pi}$ returned by the Test oracle in a qualified test session is computed as follows. Instead of the correct key $\mathtt{k}$, computed by Π, a randomly chosen key $\mathtt{k}' \in_R \mathcal{K}_\Pi$ is used to compute $\mathtt{k}_\mathcal{A}$ in the case of $b = 1$.

Claim (1). $|\mathsf{Succ}_{2,\mathcal{A}} - \mathsf{Succ}_{3,\mathcal{A}}| \leq c \cdot \mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}'}$

Proof. The claim follows directly from the AKE security of the used PAKE protocol Π and Lemma 1. Let $\mathcal{A}$ denote the adversary on the AKE-security of C_Π playing either in Exp_2 or in Exp_3. We show how to construct an adversary $\mathcal{A}'$ on the AKE-security of Π and thus bound the advantage of $\mathcal{A}$ distinguishing between Exp_2 and Exp_3 by $\mathsf{Adv}^{\mathsf{AKE\text{-}Sec}}_{\Pi,\mathcal{A}'}$ multiplied with the maximum number of passwords in one session c.

For each protocol instance between P and P, $\mathcal{A}$ interacts with c AKE experiments of Π between those parties. Note that each of those experiments uses the same secret bit b to answer Test queries. Furthermore, $\mathcal{A}$ chooses a random bit b' to use in Test query answers and thus simulate Exp_2 and Exp_3 depending on b' to $\mathcal{A}$. Oracle queries are simulated as follows. The $\mathsf{Corrupt}$ queries of $\mathcal{A}$ are answered by calling $\mathsf{Corrupt}_\Pi$ on one of the c experiments. $\mathcal{A}'$ 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 specification. Note that the passwords are random already, such that $\mathcal{A}'$ just picks random passwords to create the IHME structure for $\mathcal{A}$. If a Send_Π query finishes Π, $\mathcal{A}$ additionally calls Test_Π queries on all c experiments if Test has not been asked to the respective partnered instance. Thus, $\mathcal{A}$ is able to answer the remaining Send_{C_Π} query (O-PAKE confirmation) 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 $\mathtt{k}_f$ derived from a randomly chosen key $\mathtt{k} \in_R \mathcal{K}_\Pi$ for $b' = 0$ or derived from the according $\mathtt{k}$ returned by a Test_Π oracle

answer to a random experiment for $b' = 1$. On $b'' \leftarrow \mathcal{A}$, $\mathcal{A}'$ returns 1 to all experiments if $b'' = b'$, else 0. For $\mathsf{Succ}_{0,\mathcal{A}'} \geq \mathsf{Succ}_{2,\mathcal{A}}$ and $\mathsf{Succ}_{1,\mathcal{A}'} \geq \mathsf{Succ}_{3,\mathcal{A}}$ this results in

$$\mathsf{Adv}^{\mathsf{AKE}}_{\Pi,\mathcal{A}'} \geq |\mathsf{Succ}_{2,\mathcal{A}} - \mathsf{Succ}_{3,\mathcal{A}}|.$$

We can estimate the advantage of $\mathcal{A}$ over $\mathcal{A}'$ by the number of parallel sessions, $\mathcal{A}$ is using, i.e. the maximum number of passwords in the client's password vector c. □

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

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

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

Proof. First note that the key $\mathtt{k}'$ used for confirmation c and final key generation $\mathtt{k}$ is according to Exp_3. Thus, 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. Note that we omit the constant factor of 2 in the final estimation. □

The client messages in Exp_4 are password independent and thus can not leak any information about the passwords. Furthermore, the key $\mathtt{k}_f$ is chosen uniformly at random such that the success probability of the adversary to determine the bit b is the same as guessing b. Thus, the advantage of $\mathcal{A}$ is given by

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

and since $\mathsf{Adv}^{\mathsf{ihide}}_{\mathrm{IHME}} = 0$ and Adv_{PRF} is negligible this results in

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

□

F Performance Measurements

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