Owl: An Augmented Password-Authenticated Key Exchange Scheme

Feng Hao[1], Samiran Bag[1], Liqun Chen[2], and Paul C. van Oorschot[3]

[1] University of Warwick, UK
{feng.hao, samiran.bag}@warwick.ac.uk
[2] University of Surrey, UK
liqun.chen@surrey.ac.uk
[3] Carleton University, Canada
paulv@scs.carleton.ca

Abstract. We present Owl, an augmented password-authenticated key exchange (PAKE) protocol that is both efficient and supported by security proofs. Owl is motivated by recognized limitations in SRP-6a and OPAQUE. SRP-6a is the only augmented PAKE that has enjoyed wide use in practice to date, but it lacks the support of formal security proofs, and does not support elliptic curve settings. OPAQUE was proposed in 2018 as a provably secure and efficient alternative to SRP-6a, and was chosen by the IETF in 2020 for standardization, but open issues leave it unclear whether OPAQUE will replace SRP-6a in practice. Owl is obtained by efficiently adapting J-PAKE to an asymmetric setting, providing additional security against server compromise yet with lower computation than J-PAKE. Owl is provably secure, efficient and agile in supporting implementations in diverse multiplicative groups and elliptic curve settings. To the best of our knowledge, Owl is the first augmented PAKE solution that provides systematic advantages over SRP-6a in terms of security, computation, message sizes, and agility.[1]

1 Introduction

Password authenticated key exchange (PAKE) allows two parties to establish a high-entropy session key only based on a shared low-entropy secret (e.g., a memorable password) [3]. In general, there are two types of PAKE protocols: balanced and augmented (also resp. called symmetric and asymmetric PAKE). In the former, two parties share a common secret (e.g., a password or a hash of a password). In the latter, which is customized for a client-server setting, a client holds a password but the server stores only a one-way transformation of it. The idea is that reversing the transformation requires an offline search that proceeds by guessing through an enumeration of candidate passwords. Augmented PAKE improves security (vs. balanced PAKE) in case of *server compromise*: in a balanced PAKE, any plaintext credentials stolen from a server can be directly used to impersonate clients, but in an augmented PAKE, to recover plaintext

---

[1] A version of this paper will appear in the proceedings of FC 2024.

passwords requires an offline search. This is an easy way to make attacks more expensive, without involving tamper-resistant hardware or multiple servers [19].

Many augmented PAKE protocols are available in the literature, but only SRP-6a [28] has been widely implemented in practice, e.g., in iCloud, 1Password and ProtonMail [14]. (It is the latest version of SRP-3, following a series of revisions to patch weaknesses in the 1998 Secure Remote Password 3 protocol [30].) Security concerns [20] have been raised related to its heuristic design. The protocol also requires working over the whole range of a multiplicative group $\mathbb{Z}_p^*$ (where p is a safe prime, i.e., $p = 2q+1$ with q also prime). This makes modular exponentiation relatively expensive—e.g., given a 3072-bit p, the exponent for modular exponentiation is 3072 bits, and one exponentiation is $3072/256 = 12$ times as costly as an exponentiation in 3072-bit DSA with 256-bit exponents. Finally, SRP-6a does not support elliptic curve implementations.

OPAQUE is an augmented PAKE scheme proposed by Jarecki et al. in 2018 [20] and was selected by IETF for standardization in 2020. An advantage promoted in its favor is so-called *pre-computation security*: if a server is compromised, an attacker cannot use pre-computed tables to speed up offline search. In contrast, for some augmented schemes, a single pre-computed table can be used thereafter to efficiently recover many passwords. Compared to SRP-6a, however, the advantage is less: if a server is compromised, while it is possible for an attacker to speed up password guessing using a pre-computed table, a *unique* table must be pre-computed per user (because of SRP-6a's salt), significantly increasing attack costs in terms of memory used and pre-computation time. OPAQUE also has formal security proofs (SRP-6a does not), and seems to be much more efficient than others (for caveats, see §4).

However, OPAQUE has three limitations. First, it relies on a *constant-time* hash-to-curve (H2C) function, which is non-trivial to implement [5]. Second, the critical reliance on H2C renders the protocol unspecified in multiplicative groups in the original paper. When OPAQUE is instantiated in such a group, its performance advantages diminish (see §4). Third, in every login session of OPAQUE, the server sends a pre-computed ciphertext using authenticated encryption and a password-derived encryption key. When the password is changed, the pre-computed ciphertext will change, hence revealing whether the password has been changed from the last login [12]. Assume the credentials have been stolen from a server. If a user diligently changes their password, their account can remain safe. However, in OPAQUE, a passive attacker can monitor the login sessions to identify users who have not changed the password and then prioritize the brute-force attack against those. This attack does not apply to SRP-6a.

These above issues point to the need for an augmented PAKE scheme that is efficient across both multiplicative and elliptic curve settings, is supported by security proofs, and does not reveal information about password changes. We address this therein through an approach distinct from previous augmented PAKE designs. We efficiently adapt the symmetric J-PAKE scheme [13] to an

asymmetric setting, yielding a new scheme: Owl.[2] Compared with J-PAKE, Owl provides additional security against server compromise with even lower computation overall. We show that Owl is secure based on the Computational Diffie-Hellman (CDH) and Decision Diffie-Hellman (DDH) assumptions in the random oracle model (§3). We evaluate the performance of Owl and show that Owl has systematic advantages over SRP-6a in terms of security, computation, message sizes, and cryptographic agility in implementations (§4).

2   Protocol specification of Owl

Here we provide a specification of the Owl protocol. To ease comparison, we reuse J-PAKE notation where possible, including x_1, x_2, x_3 and x_4 for J-PAKE's four ephemeral private keys. We modify J-PAKE by fixing x_3 per user (thus leveraging the server storage available in the asymmetric setting) and furthermore combine the modified J-PAKE with a compiler due to Hwang et al. [16], achieving augmented security in an asymmetric setting. (By design a compiler *always* adds cost for converting a balanced PAKE, but our method saves cost.)

2.1   Setup

We describe the protocol in a DSA-like group G (the specification works the same in an elliptic curve setting). Let p and q be two large primes such that $q \mid p - 1$. The protocol operates in the subgroup of $\mathbb{Z}_p^*$ of prime order q. Any non-identity element in this subgroup can serve as a generator, denoted g. Unless specified otherwise, all modular operations are performed with reference to the modulus p. Let t be a (low-entropy) user-specific secret, obtained by $t = H(\texttt{username} \| \texttt{password}) \bmod q$, were H is a secure one-way function and $\|$ denotes concatenation. We define $\pi = H(t) \bmod q$ as a shared secret to be used for authenticated key exchange and $T = g^t \bmod p$ as a password verifier to be stored on the server. As in J-PAKE, we require $\pi \neq 0 \bmod q$. As summarized in Fig. 1, the protocol comprises two main phases: initial registration and login (authenticated key exchange); for completeness, we also cover password update.

2.2   Initial registration

To register an account on a server, a user computes $t = H(U \| w) \bmod q$. Here U denotes her $\texttt{username}$; w her (weak) $\texttt{password}$. She then computes $\pi = H(t) \bmod q$, $T = g^t \bmod p$, and sends (U, π, T) to the server through a secure channel (e.g., over TLS or out-of-band). The server chooses a random secret $x_3 \in_R [1, q - 1]$, and computes a public key $X_3 = g^{x_3} \bmod p$ together with a zero-knowledge proof $\Pi_3 = \text{ZKP}\{x_3 : g, X_3\}$ to prove the knowledge of the exponent x_3. When the context is clear, we shorten the notation to $\text{ZKP}\{x_3\}$. A

---

[2] Owls have asymmetric ears with one higher than the other. This asymmetry helps owls pinpoint the source of a sound in darkness.

Init registration		
User (U, w)		**Server** (S)
$t = H(U\|w) \bmod q$		$x_3 \in_R [0, q-1], X_3 = g^{x_3}$
$\pi = H(t) \bmod q$		$\Pi_3 = \text{ZKP}\{x_3\}$
$T = g^t \bmod p$	$\xrightarrow{\quad U, \pi, T \quad}$	Stores $\{U : X_3, \Pi_3, \pi, T\}$
	(Secure channel)	

Login

User (U, w)		**Server** $(S, \{U : X_3, \Pi_3, \pi, T\})$
$t = H(U\|w) \bmod q$		
$\pi = H(t) \bmod q$		
$x_1 \in_R [0, q-1], X_1 = g^{x_1}$		
$\Pi_1 = \text{ZKP}\{x_1\}$		
$x_2 \in_R [1, q-1], X_2 = g^{x_2}$		
$\Pi_2 = \text{ZKP}\{x_2\}$	$\xrightarrow{\quad U, X_1, X_2, \Pi_1, \Pi_2 \quad}$	Verify $\Pi_1, \Pi_2, X_2 \neq 1$
		$x_4 \in_R [1, q-1], X_4 = g^{x_4}$
		$\Pi_4 = \text{ZKP}\{x_4\}$
		$\beta = (X_1 X_2 X_3)^{x_4 \cdot \pi}$
Verify $\Pi_3, \Pi_4, \Pi_\beta, X_4 \neq 1$	$\xleftarrow{\quad S, X_3, X_4, \Pi_3, \Pi_4, \beta, \Pi_\beta \quad}$	$\Pi_\beta = \text{ZKP}\{x_4 \cdot \pi\}$
$\alpha = (X_1 X_3 X_4)^{x_2 \cdot \pi}$		
$\Pi_\alpha = \text{ZKP}\{x_2 \cdot \pi\}$		
$K = (\beta / X_4^{x_2 \cdot \pi})^{x_2}$		
$h = H(K\|\texttt{Transcript})$		
$r = x_1 - t \cdot h \bmod q$	$\xrightarrow{\quad \alpha, \Pi_\alpha, r \quad}$	Verify Π_α
		$K = (\alpha / X_2^{x_4 \cdot \pi})^{x_4}$
		$h = H(K\|\texttt{Transcript})$
		Verify $g^r \cdot T^h = X_1$
$k = H(K)$		$k = H(K)$

Password update

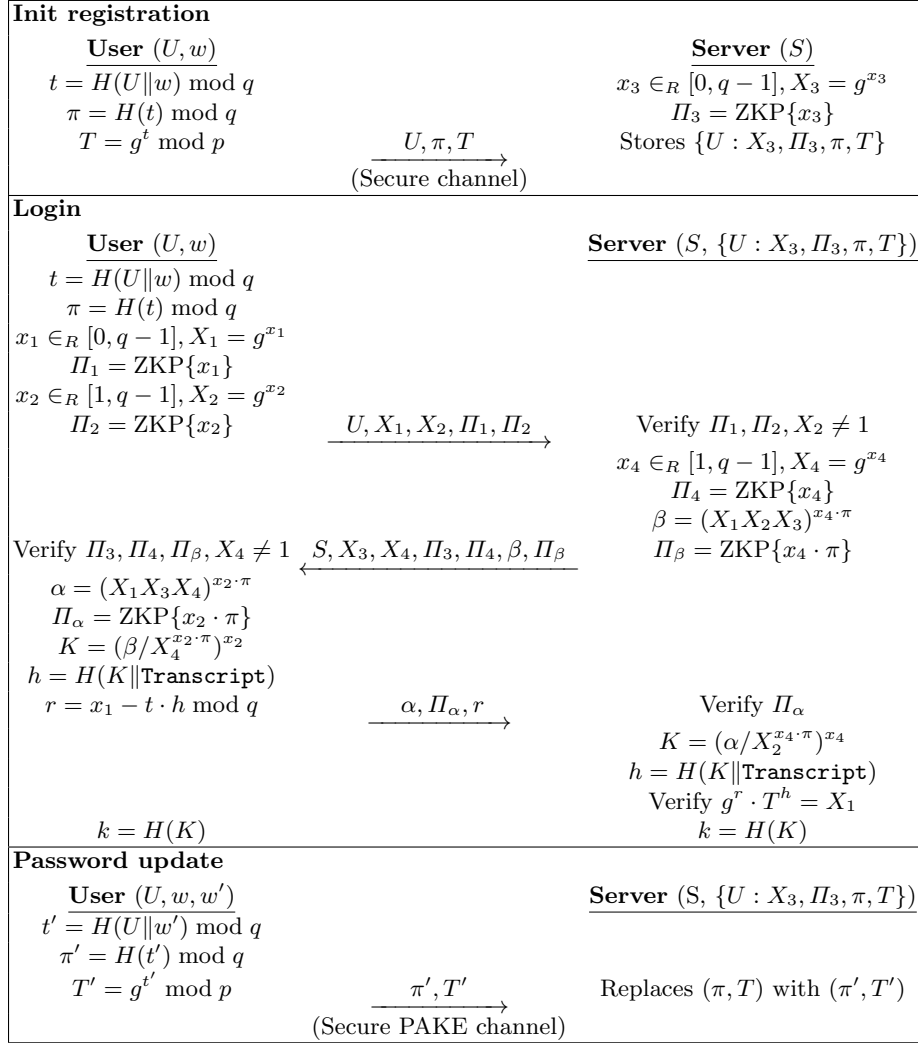| **User** (U, w, w')		**Server** $(S, \{U : X_3, \Pi_3, \pi, T\})$
$t' = H(U\|w') \bmod q$		
$\pi' = H(t') \bmod q$		
$T' = g^{t'} \bmod p$	$\xrightarrow{\quad \pi', T' \quad}$	Replaces (π, T) with (π', T')
	(Secure PAKE channel)	

Fig. 1. The Owl protocol. U is the user's identity, w her (weak) password.

concrete instantiation based on Schnorr [26] is given in Appendix A. The server stores $\{U : X_3, \Pi_3, \pi, T\}$ as the password verification file for U, and deletes x_3.

We will use π as a shared secret to run a modified J-PAKE protocol by using the pre-computed X_3 and Π_3 values instead of freshly generating them for each login session as in the original J-PAKE protocol. This modification does not affect the security of the session key, as we will show in §3. We define π as a secret salted by a unique username. This is to ensure that upon server compromise if an attacker wishes to use a pre-computed table to launch an offline dictionary attack, he needs to build a *unique* pre-computed table for each user. In addition to modifying J-PAKE, we adopt a method proposed by Hwang et al. to enable

a client to securely prove the knowledge of t for $T = g^t$ based on a variant of Schnorr non-interactive ZKP [16] with details below.

2.3 Login

The protocol runs between a client user (with unique identity U) and server (with identity S). The user initiates the communication. Both sides check: $U \neq S$. There are three flows in the login process to perform *authenticated key exchange* as presented in Figure 1. The first two flows are the same as in J-PAKE except that X_3 is fixed per user. In the third flow, the computation of r is based on a compiler proposed by Hwang et al. [16] to prove the knowledge of t for g^t in an asymmetric PAKE setting; here, we use X_1 as a commitment (which is included in `Transcript`). Our definition of `Transcript` herein is a record of the items exchanged between U and S for computing a common session key. `Transcript` $= U\|X_1\|X_2\|\Pi_1\|\Pi_2\|S\|X_3\|X_4\|\Pi_3\|\Pi_4\|\beta\|\Pi_\beta\|\alpha\|\Pi_\alpha$. If U and S have used the same passwords, they will derive a common session key (for simplicity, using a one-way hash H as a key derivation function): $k = H(K) = H(g^{(x_1+x_3)\cdot x_2\cdot x_4\cdot\pi})$. Optionally, they can perform explicit key confirmation by using standard methods (e.g., NIST SP 800-56A).

2.4 Password update

After the successful authenticated key exchange process, both parties use the session key k to create a secure channel. Through this secure channel, the user can update the password by sending π', T' to the server as shown in Fig. 1. The server updates the password verifier file for U accordingly. One reason for a user to update her password is suspicion that the old password was compromised. Owl's forward secrecy property (§3.2) ensures that a passive attacker who knows the old password cannot learn the session key k, and hence cannot learn the new password. Although OPAQUE and SRP-6a do not discuss this explicitly, they can use a similar method to update the password.

3 Security analysis

Hwang et al. [16] proposed a compiler to convert a symmetric PAKE to an asymmetric one based on a variant of Schnorr non-interactive ZKP and proved its security in a universally composable (UC) model. We adopt the method of Hwang et al. as part of our protocol (more specifically, proving the knowledge of t for $T = g^t$; see Fig. 1). As Hwang et al.'s compiler has been proven secure in the UC model on the assumption that the session key from the symmetric PAKE is secure, we focus on proving the security of the session key derived from the modified J-PAKE.

3.1 Security assumption

The security proof of Owl is based on the intractability of Multiple Decision Diffie-Hellman (MDDH) [23] and Square Computational Diffie-Hellman (SCDH) assumptions. We show that DDH and MDDH assumptions are equivalent. The SCDH and CDH assumptions are also equivalent, as shown by Bao et al. [1].

Assumption 1 (DDH): *For any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{DDH}(\lambda) \leq negl(\lambda)$, where,*

$$Adv_{\mathcal{A}}^{DDH}(\lambda) = \left| Pr[\mathcal{A}(g, g^a, g^b, g^{ab}) = 1 | a, b \xleftarrow{\$} \mathbb{Z}_q] - Pr[\mathcal{A}(g, g^a, g^b, g^c) | a, b, c \xleftarrow{\$} \mathbb{Z}_q] \right|$$

Assumption 2 (MDDH): *For $n \in poly(\lambda)$, any PPT adversary $\mathcal{A}$, the following two distributions are computationally indistinguishable.*

$$R_0 = \left((g, g^a, g^{b_i}, g^{ab_i}) : i \in [1, n] | a, b_1, b_2, \ldots, b_n \xleftarrow{\$} \mathbb{Z}_q \right)$$

$$R_1 = \left((g, g^a, g^{b_i}, g^{c_i}) : i \in [1, n] | a, b_1, b_2, \ldots, b_n, c_1, c_2, \ldots, c_n \xleftarrow{\$} \mathbb{Z}_q \right)$$

We define the advantage of $\mathcal{A}$ in distinguishing between R_0 and R_1 as,

$$Adv_{\mathcal{A}}^{MDDH}(\lambda) = \left| Pr[\mathcal{A}(R_0) = 1] - Pr[\mathcal{A}(R_1) = 1] \right|$$

Hence, $Adv_{\mathcal{A}}^{MDDH}(\lambda) \leq negl(\lambda)$, for a PPT adversary $\mathcal{A}$.

Lemma 1. *Assumption 1 and Assumption 2 are equivalent.*

Proof. ($\Rightarrow$): If there is an adversary $\mathcal{A}$ against Assumption 2, we could use it to construct another adversary $\mathcal{B}$ against Assumption 1. This has been proved in Lemma 4 by Kurosawa and Nojima [23]. ($\Leftarrow$): Let us assume that we have an adversary $\mathcal{B}$, against the DDH assumption. We use it to construct another adversary $\mathcal{A}$, against Assumption 2. $\mathcal{A}$ receives as input g, g^a, and b_i, $\Omega_{di} \in \{g^{a \cdot b_i}, R_i\}$, for $i \in [1, n]$, where $R_i \xleftarrow{\$} G$. It then invokes $\mathcal{B}$ with g, g^a, g^{b_k}, and Ω_{dk} for some $k \in [1, n]$. $\mathcal{B}$ has to check if $\Omega_{dk} = g^{a \cdot b_k}$ or a random element in G. If $\mathcal{B}$ is successful, so will be $\mathcal{A}$. Hence, the result holds.

Assumption 3 (CDH): *Given $x, y \xleftarrow{\$} \mathbb{Z}_q$, and $g \xleftarrow{\$} G$, define $CDH_g(g^x, g^y) = g^{xy}$. Consider the following security experiment $Exp_{\mathcal{A}}^{CDH}(\lambda)$. In this experiment, the challenger randomly chooses three elements g, $X = g^x$, $Y = g^y$ from G. Then the adversary $\mathcal{A}$ is invoked with these three as inputs. $\mathcal{A}$ has to compute $CDH_g(X, Y)$ from these parameters. The experiment is successful if $\mathcal{A}$ can compute $CDH_g(X, Y)$ correctly. The advantage of an adversary $\mathcal{A}$, against $Exp_{\mathcal{A}}^{CDH}(\lambda)$ is given by*

$$Adv_{\mathcal{A}}^{CDH}(\lambda) = Pr[Exp_{\mathcal{A}}^{CDH}(\lambda) = 1]$$

As such, for any PPT adversary $\mathcal{A}$ $Adv_{\mathcal{A}}^{CDH}(\lambda) \leq negl(\lambda)$.

$$
\begin{array}{|l|}
\hline
Exp_{\mathcal{A}}^{CDH}(\lambda) \\
\hline
g \xleftarrow{\$} G \\
X \xleftarrow{\$} G \\
Y \xleftarrow{\$} G \\
C \leftarrow \mathcal{A}(g, X, Y) \\
\text{Return } C \stackrel{?}{=} CDH_g(X, Y) \\
\hline
\end{array}
$$

Assumption 4 (SCDH): *Given $x \xleftarrow{\$} \mathbb{Z}_q$, and $g \xleftarrow{\$} G$, define $SCDH_g(g^x) = g^{x^2}$. Consider the following security experiment $Exp_{\mathcal{A}}^{SCDH}(\lambda)$. In this experiment, the challenger randomly chooses two elements g and $X = g^x$ from G. Then the adversary $\mathcal{A}$ is invoked with these two as inputs. $\mathcal{A}$ has to compute $SCDH_g(X)$ from these parameters. The experiment is successful if $\mathcal{A}$ can compute $SCDH_g(X)$ correctly. The advantage of an adversary $\mathcal{A}$, against $Exp_{\mathcal{A}}^{SCDH}(\lambda)$*

$$
\begin{array}{|l|}
\hline
Exp_{\mathcal{A}}^{SCDH}(\lambda) \\
\hline
g \xleftarrow{\$} G \\
X \xleftarrow{\$} G \\
C \leftarrow \mathcal{A}(g, X) \\
\text{Return } C \stackrel{?}{=} SCDH_g(X) \\
\hline
\end{array}
$$

is given by

$$
Adv_{\mathcal{A}}^{SCDH}(\lambda) = Pr[Exp_{\mathcal{A}}^{SCDH}(\lambda) = 1].
$$

As such, for any PPT adversary $\mathcal{A}$ $Adv_{\mathcal{A}}^{SCDH}(\lambda) \leq negl(\lambda)$.

Lemma 2. *Assumption 3 and Assumption 4 are equivalent [1].*

3.2 Session key indistinguishability

Impersonating the server First, we consider an active adversary impersonating the server without having the password verification file. We define a security experiment $Exp_{\mathcal{A}}^{RNDKey}(\lambda)$ to model the adversary impersonating the server.We introduce a simulator $\mathcal{SIM}$ to interact with an adversary $\mathcal{A}$. In this experiment, $\mathcal{SIM}$ can replace the session key with the hash of a random element of G, and it will not be possible for $\mathcal{A}$ to distinguish if the key it has received was the actual key or a randomly chosen key when the adversary had chosen an incorrect password. If the passwords used by the two parties do not match, both sides end up calculating different keys. In this attack scenario, the simulator responds to the queries of the attacker impersonating the server to a legitimate client without knowing the password verification file. The simulator follows the protocol specifications and sends appropriate information to the attacker against all its queries. In the end, the simulator calculates the session key. The simulator then flips a coin and returns either the actual session key or a random one depending

upon the outcome of the coin tossing. We show that the attacker will not be able to distinguish between the two keys if the password chosen by the attacker w' is not the one (w) used by the simulator. So, if the session key is compromised, the attacker will learn nothing about the actual password used by the simulator.

In the security experiment $Exp_{\mathcal{A}}^{RNDKey}(\lambda)$, we define g as a random generator of the mathematical group G. $\mathcal{D}$ is a dictionary of passwords. $|\mathcal{D}| \in poly(\lambda)$. H is a secure hash function (modelled as $\mathcal{F}_{RO}$). The adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$ is a three stage adversary. Here, $CDH_g(A, B)$ denotes the Computational Diffie-Hellman of two elements A, and B with respect to g. Also $SCDH_g(A)$ denotes the square Computational Diffie-Hellman of an element A with respect to g. First, the challenger generates the public parameters. Then she chooses a generator g from G, and a password w from $\mathcal{D}$. The challenger chooses $X_1, X_2 \xleftarrow{\$} G$, and invokes $\mathcal{A}_0$ with the specific parameters as shown in the experiment. $\mathcal{A}_0$ outputs $x_2, x_4 \in \mathbb{Z}_q^*$. Then the challenger computes R, and invokes $\mathcal{A}_1$. $\mathcal{A}_1$ outputs a password guess $w' \in \mathcal{D}$ and obtains $\pi' = H(H(U\|w'))$. Then the challenger calculates a raw key K_0, and randomly samples K_1 from G. The challenger randomly chooses one of K_0 and K_1, and invokes $\mathcal{A}_2$ with its hash value. $\mathcal{A}_2$ has to identify whether $H(K_0)$ or $H(K_1)$ was passed to her as the input. The experiment is successful if $\mathcal{A}_2$ can identify the correct challenge.

Note that if $w = w'$ (or $\pi = \pi'$), K_0 will be equal to $CDH_g(X_1, X_2)^{x_4\pi} * X_2^{x_3 x_4 \pi}$. As such, $\mathcal{A}_2$ can distinguish between $H(K_0)$ and $H(K_1)$ easily. So, we define the advantage of $\mathcal{A}$ as $\left| Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'] - \frac{1}{2} \right|$. Now, $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] = Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi = \pi'] * Pr[\pi = \pi'] + Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'] * Pr[\pi \neq \pi']$. If $\pi = \pi'$, $\mathcal{A}_2$ can easily win the experiment. Therefore, $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi = \pi'] = 1$. So, $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] = Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'] * (1 - Pr[\pi = \pi']) + Pr[\pi = \pi'] = Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'](1 - \frac{1}{|\mathcal{D}|}) + \frac{1}{|\mathcal{D}|}$. Now, $\mathcal{A}_2$ can always win with $\frac{1}{2}$ probability by making a random guess. Therefore, $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'] \geq \frac{1}{2}$. Let us assume that $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1|\pi \neq \pi'] = \frac{1}{2} + X$, where X is used to define the advantage of $\mathcal{A}$ against $Exp_{\mathcal{A}}^{RNDKey}(\lambda)$. So, $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] = (X + \frac{1}{2})(1 - \frac{1}{|\mathcal{D}|}) + \frac{1}{|\mathcal{D}|}$. Therefore, $X = \frac{|\mathcal{D}|}{|\mathcal{D}|-1} * (Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] - \frac{1}{|\mathcal{D}|}) - \frac{1}{2} = \frac{|\mathcal{D}|}{|\mathcal{D}|-1} * (Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] - \frac{1}{2|\mathcal{D}|} - \frac{1}{2})$. Since, $|\mathcal{D}|$ is $poly(\lambda)$, $\frac{|\mathcal{D}|}{|\mathcal{D}|-1}$ is a little higher than 1. So, we can eliminate the $\frac{|\mathcal{D}|}{|\mathcal{D}|-1}$ part in the above expression. Therefore, we define the advantage of an adversary $\mathcal{A}$ against the security experiment $Exp_{\mathcal{A}}^{RNDKey}(\lambda)$ as below:

$$Adv_{\mathcal{A}}^{RNDKey}(\lambda) = \left| Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] - \frac{1}{2|\mathcal{D}|} - \frac{1}{2} \right|$$

Theorem 1. *Under the SCDH and DDH assumptions with access to $\mathcal{F}_{RO}$, for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, $Adv_{\mathcal{A}}^{RNDKey}(\lambda) \leq negl(\lambda)$.*

Proof. We show that if there exists an adversary $\mathcal{A}$ against the security experiment $Adv_{\mathcal{A}}^{RNDKey}$, we can use it to construct another adversary $\mathcal{B}$, against the

```
Exp_A^{RNDKey}(λ)
─────────────────────────────────────────
(G, D, H) ← Setup(1^λ)
g ←$ G
w ← D, π = H(H(U‖w))
X_1, X_2 ←$ G
(x_3, x_4, state_0) ← A_0(g, G, D, X_1, X_2)
R ← (CDH_g(X_1, X_2) * X_2^{x_3+x_4})^π
(w', state_1) ← A_1(state_0, R)
π' = H(H(U‖w'))
K_0 ← CDH_g(X_1, X_2)^{x_4·π'} * X_2^{x_3·x_4·π'} * SCDH_g(X_2)^{x_4·(π'−π)}
K_1 ←$ G
d ←$ {0, 1}
Ω ← H(K_d)
d' ← A_2(state_1, Ω)
Return (d =? d')
```

security experiment $Exp_A^{SCDH}(\lambda)$. $\mathcal{B}$ receives as input $A \xleftarrow{\$} G$. Its aim is to compute $SCDH_g(A)$. It selects random $x_1 \xleftarrow{\$} \mathbb{Z}_q$, and assigns $X_1 = g^{x_1}$, and $X_2 = A$. It invokes $\mathcal{A}_0$ and receives x_3, and x_4. Then $\mathcal{B}$ computes $R = X_2^{(x_1+x_3+x_4)\pi}$. $\mathcal{B}$ invokes $\mathcal{A}_1$ and receives w', thus $\pi' = H(H(U\|w'))$. If $\pi' = \pi$, $\mathcal{B}$ aborts and outputs a random element from G. Else, $\mathcal{B}$ samples a random string from $\{0,1\}^\lambda$, and assigns this to Ω. When $\mathcal{A}_0, \mathcal{A}_1$ or $\mathcal{A}_2$ makes an oracle query to H, $\mathcal{B}$ answers them. For each such query, $\mathcal{B}$ samples a random string and returns it. If a query is repeated, $\mathcal{B}$ returns the same string it had returned earlier. $\mathcal{B}$ keeps a log of all query-response pairs. After $\mathcal{A}_2$ has returned, $\mathcal{B}$ checks all the queries made by $\mathcal{A}_1$ or $\mathcal{A}_2$. It randomly selects one query-response pair $\langle K_0, k_0 \rangle$ where $K_0 = \left((X_1 X_2 X_3)^{x_4 \cdot \pi'} / X_2^{x_4 \cdot \pi} \right)^{x_2} = CDH_g(X_1, X_2)^{x_4 \cdot \pi'} * X_2^{x_3 \cdot x_4 \cdot \pi'} * SCDH_g(X_2)^{x_4 \cdot (\pi' - \pi)}$ represents the raw keying material computed by $\mathcal{A}$ and k_0 the session key. $\mathcal{B}$ computes $C = (K_0/(CDH_g(X_1, X_2)^{x_4 \pi'} * X_2^{x_3 * x_4 \pi'}))^{1/(x_4(\pi' - \pi))} = SCDH_g(X_2)$ and outputs C. Let us now calculate the success probability of $\mathcal{B}$. $\mathcal{B}$ wins if the following events happen together:

- $\pi \neq \pi'$
- $\mathcal{A}$ queries $H(K_0)$.

Since H is modelled as a random oracle $\mathcal{F}_{RO}$, $\mathcal{A}_2$ cannot identify d without querying $H(K_0)$. Thus, the probability that $\mathcal{A}$ queries $H(K_0)$ is at least equal to $Pr[Exp_A^{RNDKey}(\lambda) = 1] - \frac{1}{2}$. Now, the probability that $H(K_0)$ will be queried and K_0 will have the term $SCDH_g(X_2)$ as a factor is $Pr[Exp_A^{RNDKey}(\lambda) = 1] - \frac{Pr[\pi = \pi']}{2} - \frac{1}{2}$. First, we calculate the value of $Pr[\pi = \pi']$. If $\mathcal{A}$ can guess the value of π from R, then the probability of this event happening could be as high as 1. However, the probability that $\mathcal{A}$ can guess the value of π from R is bounded by $Adv_A^{DDH}(\lambda)$ (i.e., distinguishing $CDH_g(X_1, X_2)$ from random). Thus, $Pr[\pi = \pi'] \leq \frac{1}{|\mathcal{D}|} + Adv_A^{DDH}(\lambda)$. Hence, the probability that $\mathcal{A}$

will query K_0 is at least $Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] - \frac{1}{2|\mathcal{D}|} - \frac{1}{2}Adv_{\mathcal{A}}^{DDH}(\lambda) - \frac{1}{2}$. Now, $\mathcal{B}$ randomly picks a query and computes $SCDH_g(A)$ on the basis of that. So if there are Q queries to H, the probability that $\mathcal{B}$ will pick the correct one is $1/Q$. Thus, $Adv_{\mathcal{B}}^{SCDH}(\lambda) \geq (1/Q) * (Pr[Exp_{\mathcal{A}}^{RNDKey}(\lambda) = 1] - \frac{1}{2|\mathcal{D}|} - \frac{1}{2}Adv_{\mathcal{A}}^{DDH}(\lambda) - \frac{1}{2}) = (1/Q)(Adv_{\mathcal{A}}^{RNDKey}(\lambda) - \frac{1}{2}Adv_{\mathcal{A}}^{DDH}(\lambda))$. Hence, $Adv_{\mathcal{A}}^{RNDKey}(\lambda) \leq Q * Adv_{\mathcal{B}}^{SCDH}(\lambda) + \frac{1}{2} * Adv_{\mathcal{A}}^{DDH}(\lambda)$.

Impersonating the client Next, we consider an active adversary impersonating the client without knowing the password. We consider a security experiment $Exp_{\mathcal{A}}^{RNDKey1}(\lambda)$, which emulates the event when the attacker tries to impersonate the user to a legitimate server without knowing the password. In this experiment, we introduce an oracle $\mathcal{O}$ that can be queried by the adversary. This oracle models the event that the adversary can exploit the fact the server uses the same value of X_3 across all the executions of the protocol. So, the adversary can try to impersonate the client and execute the protocol with the server multiple times and can receive the second-flow messages from the server for different values of X_4, but a single value of X_3. In the end, the goal of the adversary is to distinguish between the session key computed by an honest instance and a random string. Similar to the case of $Exp_{\mathcal{A}}^{RNDKey}(\lambda)$, the advantage of the adversary $\mathcal{A}$, against the security experiment $Exp_{\mathcal{A}}^{RNDKey1}(\lambda)$ (which models an attacker impersonating the user) is defined as

$$Adv_{\mathcal{A}}^{RNDKey1}(\lambda) = \left| Pr[Exp_{\mathcal{A}}^{RNDKey1}(\lambda) = 1] - \frac{1}{2|\mathcal{D}|} - \frac{1}{2} \right|$$

$Exp_{\mathcal{A}}^{RNDKey1}(\lambda)$
$(G, \mathcal{D}, H) \leftarrow Setup(1^\lambda)$
$g \xleftarrow{\$} G$
$w \leftarrow \mathcal{D}, \pi = H(H(U\|w))$
$X_3, X_4 \xleftarrow{\$} G$
$d \xleftarrow{\$} \{0,1\}$
$(x_1, x_2) \leftarrow \mathcal{A}^{\mathcal{O}}(g, G, D, X_3, X_4)$
$R \leftarrow (CDH_g(X_3, X_4) * X_4{}^{x_1+x_2})^\pi$
$w' \leftarrow \mathcal{A}(R)$
$\pi' = H(H(U\|w'))$
$B_0 \leftarrow CDH_g(X_3, X_4)^{x_2 \cdot \pi'} * X_4^{x_1 \cdot x_2 \cdot \pi'} * SCDH_g(X_4)^{x_2(\pi'-\pi)}$
$B_1 \xleftarrow{\$} G$
$d' \leftarrow \mathcal{A}(H(B_d))$
return $d \overset{?}{=} d'$

$\mathcal{O}()$
$X \xleftarrow{\$} G$
$(x_1, x_2) \leftarrow \mathcal{A}(X)$
$R \leftarrow (CDH_g(X_3, X) * X^{x_1+x_2})^\pi$
Return R

Theorem 2. *Under the SCDH and MDDH assumptions with access to $\mathcal{F}_{RO}$, for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{RNDKey1}(\lambda) \leq negl(\lambda)$.*

Proof. We can show that if there exists an adversary $\mathcal{A}$ against the security experiment $Exp_{\mathcal{A}}^{RNDKey1}(\lambda)$, we can use it in the construction of another adversary $\mathcal{B}$ again the experiment $Exp_{\mathcal{A}}^{SCDH}(\lambda)$. The proof is almost the same as the proof of Theorem 1. The only difference is that because of the use of a fixed X_3 value across the sessions, we use Assumption 2 instead of Assumption 1. All other arguments are the same. Thus, we substitute the DDH assumption of Theorem 1 with Assumption 2, and have $Adv_{\mathcal{A}}^{RNDKey1}(\lambda) \leq Q * Adv_{\mathcal{A}}^{MDDH}(\lambda) + \frac{1}{2} * Adv_{\mathcal{A}}^{SCDH}(\lambda)$. Hence, the result follows.

Session key exposure The following experiment $Exp_{\mathcal{A}}^{KeyExp}(\lambda)$ models the event where the attacker gets hold of an actual session key after the key is derived. We will show that the actual session key is indistinguishable from a random key in the view of the attacker even with the knowledge of the password (i.e., forward secrecy [13]). This can happen if $Adv_{\mathcal{A}}^{KeyExp}(\lambda)$ is negligible, where

$$Adv_{\mathcal{A}}^{KeyExp}(\lambda) = \left| Pr[Exp_{\mathcal{A}}^{KeyExp}(\lambda) = 1] - \frac{1}{2} \right|.$$

$Exp_{\mathcal{A}}^{KeyExp}(\lambda)$

$(G, \mathcal{D}, H) \leftarrow Setup(1^{\lambda})$

$g \xleftarrow{\$} G$

$X_4 \xleftarrow{\$} G$

$(\pi, state) \leftarrow \mathcal{A}_0(G, \mathcal{D}, H, g, X_4)$

$X_1, X_2, X_3 \xleftarrow{\$} G$

$\alpha \leftarrow CDH_g(X_1 * X_3 * X_4, X_2)^{\pi}$

$\beta \leftarrow CDH_g(X_1 * X_2 * X_3, X_4)^{\pi}$

$K_0 \leftarrow H(CDH_g(X_1 * X_3, X_2)^{\pi \cdot \log_g X_4})$

$K_1 \xleftarrow{\$} \{0, 1\}^*$

$d \xleftarrow{\$} \{0, 1\}$

$d' \leftarrow \mathcal{A}_{\infty}(state, X_1, X_2, X_3, \alpha, \beta, C_d)$

return $d \overset{?}{=} d'$

Theorem 3. *Under the DDH and SCDH assumptions with access to $\mathcal{F}_{RO}$, for any PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{KeyExp}(\lambda) \leq negl(\lambda)$.*

Proof. We show that if there exists an adversary $\mathcal{A}$ against the experiment $Exp_{\mathcal{A}}^{KeyExp}(\lambda)$, it could be used in the construction of another adversary $\mathcal{B}$ against the security experiment $Exp_{\mathcal{B}}^{SCDH}(\lambda)$. The adversary $\mathcal{B}$ works as follows: it receives as input an $X_4 \xleftarrow{\$} G$, and it needs to compute $SCDH_g(X_4)$. The adversary $\mathcal{B}$ proceeds as follows: It selects $x_1 \xleftarrow{\$} \mathbb{Z}_q$, $x_2, a \xleftarrow{\$} \mathbb{Z}_q^*$ and sets $X_1 = g^{x_1}$, $X_2 = g^{x_2}$, and $X_3 = X_4 * g^a$. That is, $\mathcal{B}$ implicitly sets $x_3 = \log_g X_3 = a + \log_g X_4$. Let $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ be a two-stage adversary. When

$\mathcal{A}_0$ is invoked with the corresponding parameters, it outputs the password π from the dictionary $\mathcal{D}$. $\mathcal{B}$ can compute $\alpha = (g^{x_1 x_2} * X_4^{2x_2} * g^{x_2 a})^\pi$. $\mathcal{B}$ samples random $\beta \stackrel{\$}{\leftarrow} G$. Now, $\mathcal{B}$ samples a string $\{0,1\}^*$, and assigns this to C_d. $\mathcal{A}_1$ is then invoked with all the necessary parameters. If $\mathcal{A}_0$ can identify d, it will have to query $CDH_g(X_1 * X_3, X_2)^{\pi \cdot \log_g X_4}$. $\mathcal{B}$ checks all the queries made by $\mathcal{A}_1$. From all the queries made by $\mathcal{A}_1$, $\mathcal{B}$ randomly picks one query Δ, and outputs $SCDH_g(X_4) = \frac{(\Delta^{1/x_2 \pi})}{X_4^{(x_1+a)}}$. Now, we calculate the advantage of the adversary $\mathcal{B}$. Note that in this experiment we replace β with a random element from G. According to the DDH assumption, $CDH_g(X_1 * X_2 * X_3, X_4)$ is indistinguishable from random. Therefore, replacing it with a random element will reduce the advantage of $\mathcal{A}$ by a factor of $Adv_{\mathcal{A}}^{DDH}(\lambda)$. Let us assume that $\mathcal{A}_1$ makes Q queries to H. $\mathcal{B}$ selects one query made by $\mathcal{A}_1$ to H, and calculates $SCDH_g(X_4)$ on the basis of this. Thus, $Adv_{\mathcal{B}}^{SCDH}(\lambda) \geq 1/Q * (Adv_{\mathcal{A}}^{KeyExp}(\lambda) - Adv_{\mathcal{A}}^{DDH}(\lambda))$. That is, $Adv_{\mathcal{A}}^{KeyExp}(\lambda) \leq Adv_{\mathcal{A}}^{DDH}(\lambda) + Q * Adv_{\mathcal{B}}^{SCDH}(\lambda)$. Hence, the result holds.

4 Related work and comparison

Many augmented PAKE schemes have been proposed in the past three decades. Hao and van Oorschot [14] categorize PAKE schemes according to whether they require an ideal cipher, a hash-to-group function, or a trusted setup (see Fig. 2). KHAPE [9],OKAPE [6] and aEKE [4] are examples that rely on *an ideal cipher*, an abstract building block that is assumed to not leak any information about the encryption content even when the encryption key has low entropy (e.g., using a password as the key) [2]. However, how to instantiate such an ideal cipher has remained an open problem [14]. VTBPEKE [25], KC-SPAKE2+ [27] and Bradley et al.'s scheme [5] rely on a *trusted setup* but a compromise in this setup will break key exchange sessions for all users [25]. PAK-Z+ [8], AuCPace [10] and OPAQUE [20] rely on a hash-to-group (also called hash-to-curve in an EC setting) function, which is assumed to map a password to a random generator of a designated group in *constant time*[3]. However, instantiating this function is not easy as we will explain later. SRP-6a does not depend on an ideal cipher, a hash-to-group function or a trusted setup, which has contributed to its wide adoption in real-world applications [14]. AMP and AugPAKE follow a similar design as SRP with the aim of better efficiency. AMP has been repeatedly revised to patch vulnerabilities [24]. AugPAKE has not been published in a peer-reviewed paper (it is described in an IETF RFC), and its security proof is questioned [20].

In this section, we focus on comparing Owl with SRP-6a and OPAQUE. We chose SRP-6a because it is the only augmented PAKE that has been widely used in practice. Although many provably secure augmented PAKE schemes have been proposed (see Figure 2), they generally rely on assumptions of an ideal cipher, a trusted setup or a hash-to-group function; however, difficulties in the realization of such assumptions have hindered the deployment of these

---

[3] A non-constant-time mapping could leak the password through timing attacks [29].
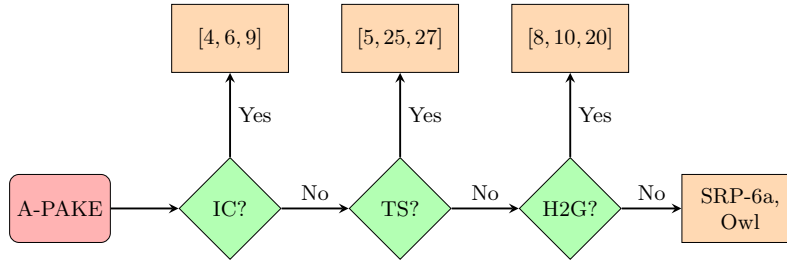
Fig. 2. Overview of selected Augmented PAKE (A-PAKE) schemes and dependence on various assumptions. IC: Ideal Cipher. TS: Trusted Setup. H2G: Hash-to-Group (also called Hash-to-Curve in an EC setting). SRP-6a and Owl do not require any ideal cipher, trusted setup or hash-to-group functions.

schemes [14]. We chose OPAQUE because it was selected by the IETF as a candidate for standardization and a possible replacement for SRP-6a.

Hash-to-group/hash-to-curve. In the original OPAQUE paper [20], the authors specify the protocol only in an EC setting, not in any multiplicative group (MODP). The rationale was however not explained. To provide insights into this design choice, we first need to clarify the hash-to-group (H2G) and hash-to-curve (H2C) functions. OPAQUE relies on H2C to map a password to a random prime-order generator on an elliptic curve in *constant time*. H2G is the equivalent function in a MODP group. The original idea of using H2G in PAKE is from Jablon's 1996 design of SPEKE [18]. More concretely, SPEKE defines a safe prime $p = 2q + 1$ as the modulus, i.e., q is also prime. The H2G function with input a is defined as $f(a) = a^2 \bmod p$, which forms the basis of SPEKE as standardized in IEEE 1363.2 [17]. However, using a safe prime as modulus makes modular exponentiations costly. Extending this construction to a DSA group (which uses short exponents) and EC proves harder than it seems.

- **DSA**. For a DSA group (p, q, g), IEEE 1363.2 defines the H2G function with input a as: $f(a) = a^{(p-1)/q}$ [17]. This function has two known issues: 1) it is costly due to the long exponent; 2) it may return an identity element (called an 'invalid' output). IEEE 1363.2 does not define any handling for 'invalid' outputs; handling such exceptions can forfeit the constant-time property.
- **EC**. To do the equivalent of H2G in EC, IEEE 1363.2 defined an H2C function based on a trial-and-increment method. The IEEE function guarantees the output is a prime-order generator (no 'invalid' output) and works with general curves, but is vulnerable to timing attacks [29]. IEEE 1363.2 was officially withdrawn in 2019. In 2023, IETF summarized custom-built H2C functions for selected curves [15], but existing functions do not exclude low-order (i.e., 'invalid') points by design and may not work with future curves. Excluding invalid points at the output can break the constant-time property.

Computational performance. SRP-6a mandates the use of a safe prime $p = 2q + 1$ where q is also a prime as the modulus. This leaves the protocol undefined for a multiplicative group with short exponents such as DSA, as well

Scheme	Flows	KC	MODP (safe prime)		MODP (DSA)		Elliptic Curve	
			Client	Server	Client	Server	Client	Server
SRP-6a	3/4	Exp	2(x12)+1	2(x12)+1	–	–	–	–
OPAQUE	3/2	Imp	4.5(x12)	3.5(x12)	1(x11)+6.5†	5.5†	4.5+H2C‡	3.5
Owl	1/3	Imp	–	–	14	13	11	10

Table 1. Comparison among selected PAKE schemes. The flows column gives the number of flows for registration/login respectively. Computational cost columns give the number of exponentiation in the MODP setting, assuming a 3072-bit modulus for concreteness. $2(\times12)$ denotes that the cost of one exponentiation (3071-bit exponent) is about the same as 12 typical 3072-bit DSA group exponentiations (256-bit exponent). The hash-to-group function in OPAQUE requires an exponentiation with a 2816-bit exponent (co-factor), which has cost equal to 11 exponentiations with 256-bit exponent in DSA. H2C denotes (yet unknown) cost of a hash-to-curve function. † denotes having concerns about 'invalid' output. ‡ denotes having concerns about computational cost, 'invalid' output and agility. KC: key confirmation. Exp: explicit. Imp: implicit.

as for an elliptic curve setting. OPAQUE is built on two building blocks: 1) authenticated key exchange (AKE) and 2) hash-to-curve. After OPAQUE was selected by IETF, its designers modified the protocol by using 3DH instead of HMQV to instantiate AKE in an IETF Internet draft [22]. Here, we will evaluate the performance of OPAQUE based on using HMQV as specified in the original paper [20]. (The performance will decrease when 3DH is used). The OPAQUE paper assumes a H2C function to map a password to a generator of the prime-order (sub)group on an elliptic curve, which leaves the protocol undefined for the MODP setting. We fill this gap by using two known H2G functions as defined for SPEKE and PAK in IEEE 1363.2 [17] to do the equivalent of hash-to-curve in two MODP settings respectively: 1) using a safe-prime modulus; 2) using DSA groups. Table 1 summarizes the comparison results (see Hao and van Oorschot [14] for a detailed breakdown of cost for SRP-6a and OPAQUE; also note that the elliptic curve implementation of Owl has a fewer number of scalar multiplications than modular exponentiations in the DSA implementation because the public key validation in the EC setting incurs a negligible cost whilst in the DSA setting, it requires one full modular exponentiation). Owl has clear advantages over SPR-6a in terms of flows, computation, and agility to work with both MODP and EC settings. Implementing OPAQUE in DSA gives a lower computation cost than using a safe-prime modulus (at the expense of having the 'invalid' output issue). In the DSA setting, the client requires more computation than Owl, but the server requires less. A direct comparison between the two in the EC setting is difficult due to the cost of H2C yet unclear (aside from the 'invalid' output and the agility issues that need to be addressed).

Round efficiency. Owl only requires one flow in the registration process, fewer than others. Between Owl and OPAQUE, OPAQUE seems to have the advantage of needing only two flows in the login process; Owl needs three. The (theoretical) advantage of a two-flow OPAQUE is that it allows the server to be authenticated in the second flow (based on explicit key confirmation) before

Schemes	MODP (safe prime)	MODP (DSA)	Ellipic Curve
SRP-6a	768 B	–	–
OPAQUE	2688 B	2336 B[†]	225 B[‡]
Owl	–	2720 B	609 B

Table 2. Comparison of message sizes in bytes (B) among selected PAKE schemes. The 'MODP (safe prime)' column gives the size in a MODP setting, assuming a 3072-bit safe-prime modulus. The 'MODP (DSA)' column gives the size in a MODP setting, assuming a 3072-bit DSA group with a 256-bit exponent. The "Elliptic Curve" column gives the size in an EC setting, assuming a 256-bit prime-order EC group. † denotes having concerns about 'invalid' output. ‡ denotes having concerns about computational cost, 'invalid' output and agility.

the client is authenticated. However, doing so is dangerous as it will expose the server to an *undetectable* online dictionary attack [21], in which a malicious client does not send the third flow and it is impossible for the server to distinguish the authentication failure from a drop-out. (In an asymmetric setting, the server is always online, and hence is more vulnerable to online guessing attacks than a client.) Preventing this attack requires the client to be authenticated first. Hence, OPAQUE actually requires the same three flows as Owl in the login.[4]

Message size. We now compare the message size among the three protocols. For simplicity, we only consider the login phase and focus on public-key cryptographic data (excluding auxiliary data such as user identities). Furthermore, we only consider implicit key confirmation, hence excluding explicit key confirmation strings that apply to all schemes. Table 2 summarizes the result. SRP-6a [28] involves exchanging two group elements in the login process (we omit the salt). Assume a 3072-bit safe-prime modulus is used. The total size is $3072 \times 2 = 6,144$ bits = 768 bytes. OPAQUE [20] involves exchanging four group elements and an encrypted string using an authenticated encryption scheme (containing two group elements and a private key generated from the registration phase). Assume OPAQUE uses the same 3072-bit safe-prime modulus as SRP-6a. The total size is approximately $3072 \times 6 + 3071 = 21,503$ bits = 2,688 bytes (we omit the size of the IV and the message authenticated code). If a 3072-bit DSA group is used instead, the total size is $3072 \times 6 + 256 = 18,688$ bits = 2,336 bytes. Assume an EC setting where the group has a prime order of 256 bits. We assume a group element requires 257 bits in a compressed form. Hence, the size of the messages is $257 \times 6 + 256 = 1,798$ bits = 225 bytes. Owl involves sending 6 group elements, 6 Schnorr ZKP and a short response in the key exchange process. In a 3072-bit DSA group, the size of each ZKP (h, r) is 512 bits (see App. §2.1). Hence, the total size is $3072 \times 6 + 512 \times 6 + 256 = 21,760$ bits = 2,720 bytes. In an EC setting, the size is $257 \times 6 + 512 \times 6 + 256 = 4,870$ bits = 609 bytes. Although Owl involves sending more group elements than SRP-6a, it can actually use less bandwidth because of its agile support for EC implementations. In the same EC

---

[4] The original conference paper describes OPAQUE as a two-flow scheme, but the full version has updated the specification requiring three flows in the login [20].

setting, the size of the messages in Owl is bigger than that in OPAQUE, but Owl is more flexible to work with any elliptic curve that is suitable for cryptography while OPAQUE is restricted by the availability of a *constant-time* and *correct* (no 'invalid' output) H2C function on that curve.

5 Conclusion

In this paper, we present a new augmented PAKE protocol called Owl. Our design strategy is to utilize Schnorr non-interactive zero-knowledge proof to enforce every party to follow the protocol specification honestly. We prove the security of Owl based on the standard DDH and CDH assumptions in the random oracle model. Through a thorough evaluation, we show Owl provides systematic advantages over SRP-6a in terms of provable security, computation, bandwidth and agility for implementation in versatile group settings. Owl's agility is attributed to the fact that it utilizes only the most basic addition/multiplication operations on an elliptic curve (equivalently, multiplication/exponentiation in MODP) plus a *standard* one-way hash function for implementation without depending on any ideal cipher, trusted setup, hash-to-group functions or special group characteristics (such as the mandated use of a safe-prime modulus in SRP-6a).

Acknowledgement

References

1. F. Bao, R. Deng, and H. Zhu. Variations of diffie-hellman problem. In *ICICS*, 2003.
2. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Eurocrypt*, 2000.
3. S. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *IEEE S&P*, 1992.
4. S. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *CCS*, 1993.
5. T. Bradley, S. Jarecki, and J. Xu. Strong asymmetric PAKE based on trapdoor CKEM. In *Crypto*. Springer, 2019.
6. B.F. Dos Santos, Y. Gu, S. Jarecki, and H. Krawczyk. Asymmetric pake with low computation and communication. In *Eurocrypt*. Springer, 2022.
7. A. Fiat and A. Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Crypto*, 1987.

8. C. Gentry et al. PAK-Z+. *Submission to IEEE P1363.2*, 2005.

9. Y. Gu, S. Jarecki, and H. Krawczyk. KHAPE: Asymmetric PAKE from key-hiding key exchange. In *Crypto*, 2021.

10. B. Haase and B. Labrique. AuCPace: Efficient verifier-based PAKE protocol tailored for the IIoT. *IACR Trans. CHES*, pages 1–48, 2019.

11. F. Hao. Schnorr non-interactive zero-knowledge proof. *RFC 8235*, 2017.

12. F. Hao. Prudent practices in security standardization. *IEEE Communications Standards Magazine*, 2021.

13. F. Hao and P. Ryan. Password authenticated key exchange by juggling. In *SPW*, 2008.

14. F. Hao and P.C. van Oorschot. SoK: Password-Authenticated Key Exchange–Theory, Practice, Standardization and Real-World Lessons. In *AsiaCCS*, 2022.

15. A. Hernández, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves. *IETF RFC 9380*, 2023.

16. J Hwang, S. Jarecki, T. Kwon, J. Lee, J. Shin, and J. Xu. Round-reduced modular construction of asymmetric password-authenticated key exchange. In *SCN*. Springer, 2018.

17. IEEE Standards Association. IEEE 1363.2-2008: IEEE Standard Specification for Password-Based Public-Key Cryptographic Techniques. `https://standards.ieee.org/standard/1363_2-2008.html`.

18. D. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Computer Commun. Review*, 26(5):5–26, 1996.

19. D. Jablon. Password authentication using multiple servers. In *CT-RSA*, 2001.

20. S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *Eurocrypt*, 2018. See also IACR e-print 2018/163.

21. Y. Kobayashi et al. Gateway threshold password-based authenticated key exchange secure against undetectable on-line dictionary attack. In *ICETE*. IEEE, 2015.

22. H. Krawczyk, D. Bourdrez, K. Lewi, and C. Wood. The OPAQUE asymmetric PAKE protocol. *IETF draft-irtf-cfrg-opaque-12*, 2023.

23. K. Kurosawa and R. Nojima. Simple adaptive oblivious transfer without random oracle. In *AsiaCrypt*, pages 334–346. Springer, 2009.

24. T. Kwon. Revision of AMP in IEEE P1363.2 and ISO/IEC 11770-4. *Submission to IEEE P1363*, 2005.

25. D. Pointcheval and G. Wang. VTBPEKE: Verifier-based two-basis password exponential key exchange. In *AsiaCCS*, 2017.

26. Claus-Peter Schnorr. Efficient signature generation by smart cards. *Journal of cryptology*, 4(3):161–174, 1991.

27. V. Shoup. Security Analysis of SPAKE2+. In *TCC*, 2020.

28. SRP Protocol Design. `http://srp.stanford.edu/design.html`. Includes description of SRP-6a. Accessed 10 February 2023.

29. M. Vanhoef and E. Ronen. Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd. In *IEEE S&P*, 2020.

30. T. Wu. The Secure Remote Password protocol. In *NDSS*, 1998.

Appendix

A Schnorr non-interactive zero-knowledge proof

Given a private key $x \in_R [0, q-1]$ and the corresponding public key $X = g^x \bmod p$, we use Schnorr non-interactive zero-knowledge (NIZK) proof to convey

the knowledge of the exponent x without revealing its value [11]. Specifically, to generate a zero-knowledge proof for $\text{ZKP}\{x : g, X\}$, the prover chooses a random secret $v \in_R [0, q-1]$, computes $V = g^v \bmod p$, and outputs (h, r) as the "proof", where $h = H(g\|V\|X\|\texttt{ProverID})$ and $r = v - x \cdot h \bmod q$. $\texttt{ProverID}$ represents the prover's unique identity. $H(\cdot)$ is a secure one-way hash function which serves to transform an interactive ZKP into a non-interactive one based on the Fiat-Shamir heuristics [7]. Verification of the ZKP requires that the verifier check:

1) X has the correct prime order; and

2) $h \stackrel{?}{=} H(g\|g^r \cdot X^h\|X\|\texttt{ProverID})$, where h is the received value.

Note: In the original J-PAKE protocol [13], the Schnorr ZKP contains (V, r) while we use (h, r). The two are equivalent [11, §4] with the same computation cost, but the latter has a more compact size in a MODP setting.