This extended abstract appeared in
Proceedings of the 2017 ACM Symposium on Information, computer and communications security (AsiaCCS '17)
(April 4–6, 2017, Abu Dhabi, United Arab Emirates), ACM Press. DOI: 10.1145/3052973.3053026

VTBPEKE: Verifier-based Two-Basis Password Exponential Key Exchange

David Pointcheval[1] and Guilin Wang[2]

[1] CNRS, INRIA, and ENS/PSL Research University, Paris, France
[2] Shield Lab, Huawei, #03-31/32, 20 Science Park Road, Singapore 117674

Abstract PAKE protocols, for *Password-Authenticated Key Exchange*, enable two parties to establish a shared cryptographically strong key over an insecure network using a short common secret as authentication means. After the seminal work by Bellovin and Merritt, with the famous EKE, for *Encrypted Key Exchange*, various settings and security notions have been defined, and many protocols have been proposed.
In this paper, we revisit the promising SPEKE, for *Simple Password Exponential Key Exchange*, proposed by Jablon. The only known security analysis works in the random oracle model under the CDH assumption, but in the multiplicative groups of finite fields only (subgroups of $\mathbb{Z}_p^*$), which means the use of large elements and so huge communications and computations. Our new instantiation (TBPEKE, for *Two-Basis Password Exponential Key Exchange*) applies to any group, and our security analysis requires a DLin-like assumption to hold. In particular, one can use elliptic curves, which leads to a better efficiency, at both the communication and computation levels. We additionally consider server corruptions, which immediately leak all the passwords to the adversary with symmetric PAKE. We thus study an asymmetric variant, also known as VPAKE, for *Verifier-based Password Authenticated Key Exchange*. We then propose a verifier-based variant of TBPEKE, the so-called VTBPEKE, which is also quite efficient, and resistant to server-compromise.

Keywords: Password-authenticated key exchange; server compromise; dictionary attacks

1 Introduction . . . 2
1.1 Dictionary Attacks . . . 2
1.2 Previous Constructions . . . 3
1.3 Objectives and Contributions . . . 5
2 Security Model . . . 5
3 Variants of SPEKE . . . 6
3.1 A Naive Construction . . . 6
3.2 A Secure Construction: TBPEKE . . . 6
4 Security Analysis of TBPEKE . . . 7
4.1 Assumptions . . . 7
4.2 Security Results for TBPEKE . . . 9
5 VTBPEKE: Verifier-based TBPEKE Protocol . . . 9
5.1 Password Hashing Scheme . . . 9
5.2 Our Password Hashing Scheme . . . 10
5.3 Description of VTBPEKE . . . 10
6 Security Analysis of VTBPEKE . . . 10
6.1 Discussions . . . 10
6.2 Forward-Secrecy & Verifier-Based . . . 11
6.3 Security Proof . . . 11
7 Parameters and Efficiency . . . 15
8 Conclusion . . . 16

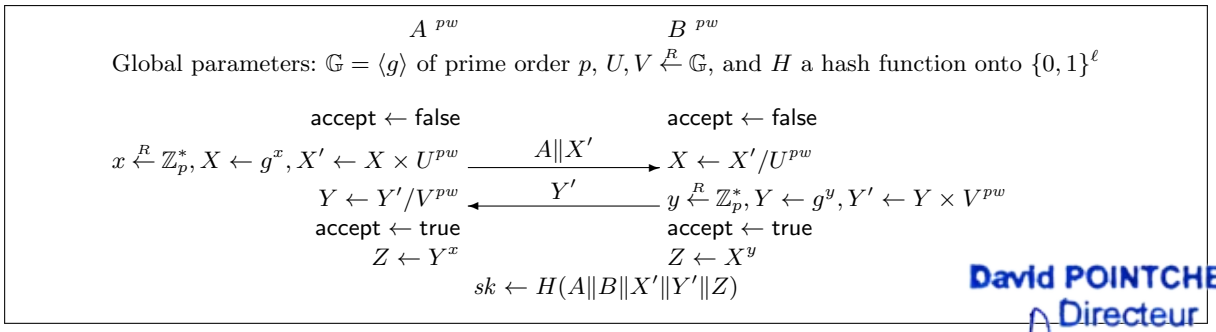

Figure 1. Simple Password Authenticated Key Exchange

1 Introduction

1.1 Dictionary Attacks

Key exchange protocols are cryptographic primitives used to provide several users (two or more), communicating over a public unreliable channel, with a secure session key. This allows establishment of virtual secure channels (*i.e.*, private and authenticated) in any network, even under the control of adversaries. This is thus one of the main practical applications of cryptography: it can guarantee privacy to any user, whatever the architectures are and the providers do.

Bellare and Rogaway gave the first foundations of authenticated key exchange in [10,11], but password-based authentication requires more work: in this setting, where the authentication means is a short secret chosen from a small set of possible values (*e.g.*, a four-digit PIN), the brute-force method, which consists in trying all the possible values in the dictionary, succeeds after a rather small number of attempts. This attack is called *on-line dictionary attack* and is unavoidable, since the adversary just has to check whether the authentication with a tentative password succeeds or not. But its damages can be limited by a policy that invalidates or blocks an account, and thus the use of a password, after a fixed number of failures (which is always possible in the two-party setting, but not necessarily in some other settings where such failures can be undetectable to the authenticator [26]). The security goal when studying a PAKE protocol is to show that this on-line dictionary attack is the best one can do, and namely that no one can guess the correct password without a linear number of interactions.

On the other hand, the classical *symmetric* PAKE setting, as introduced by Bellovin and Merritt [12] in 1992, with the famous *Encrypted Key Exchange* protocol (EKE), requires the server to know all the passwords in clear, which can be dramatic in case of intrusion into the server. With the increase of password-controlled accesses, users often use related passwords (if not the same) for many providers, which amplifies the damages of a hack of a single server to many services. To overcome this issue, Bellovin and Merritt proposed the *Augmented EKE* protocol [13], where the server just stores a means, called a *verifier*, to verify that the client used the correct password, but not the password itself. In concrete systems, the verifier is a hash of the password with a salt. This temporarily limits the impact of leakage of information on the server side, since it forces the adversary to spend a lot of time to learn many passwords. This should give enough time for letting the users renew (all) their passwords.

Such an *asymmetric* PAKE is also known as VPAKE, for *Verifiable Password-Authenticated Key Exchange*. In addition to the basic security when there is no server compromise, extracting the password from the verifier in case of database corruption should take a computation time linear in the number of possible passwords, for each user, when passwords are uniformly distributed, without any speed-up taking advantage of the large number of verifiers. This corresponds to the time of the trivial *off-line dictionary attack*, which we cannot avoid but which should be the best: for each verifier, the adversary tries all the possible passwords. Of course, in

such a case, the number of possible passwords should not be too small, and the time to check a possible candidate to a verifier should not be too short either (hence the use of slow hash functions).

Indeed, the same way as the *on-line dictionary attack* cannot be avoided but is proven to be the best possible attack against a PAKE (or a VPAKE, before a server compromise), an *off-line dictionary attack* cannot be avoided in case of server compromise, and one has to prove any new password-recovery is linear in the size of the dictionary after a server compromise.

As already mentioned, verifiers are usually hash values or transformations $V = \mathcal{H}(s, pw)$ of the private passwords pw with public extra data s, called *salt*. For each user, the server thus stores the pair (s, V). The salt aims at binding an exhaustive search on the passwords to a specific user, or at least to a small fraction with the same salt. Indeed, after the compromise of a server, or even several servers, the adversary gets access to many verifiers for many users, together with any information useful to verify the passwords. If there are many collisions among the salt values, the adversary can focus on the salt value s^* that corresponds to the highest number of verifiers $V_1^*, \ldots, V_k^*$. Then, the exhaustive search on the password, which consists in computing $H(pw, s^*)$ for all the passwords, will fall on one of the V_i^*'s with probability k/N for each test, where N is the size of the dictionary and k the number of verifiers for the tested salt value s^*. The smaller k is, the safer is the system. Hence, one has to limit the collusions on the salt values: in practice, this is either a random value, but then it must be large enough to avoid collisions (birthday paradox), or server-user identities, which exclude collisions, have compact representations, and can even be known in advance by the user. This might depend on the practical scenario.

1.2 Previous Constructions

Bellovin and Merritt [12] proposed the first scheme, the so-called Encrypted Key Exchange (EKE), which is essentially a Diffie-Hellman key exchange, where the two flows are encrypted with a symmetric encryption scheme under the password as the symmetric key. A first security analysis has been provided in the indistinguishability-based (or BPR) framework, in the ideal-cipher model [8], followed by several proofs of variants [18,19,6], trying to reduce the need of ideal models but still keeping the initial efficiency of EKE. EKE has also been studied in the simulation-based framework, in the random-oracle model [17], followed by studies in the UC framework [3] with security against adaptive corruptions, but still in ideal models.

The "Simple Password-Authenticated Key Exchange" protocols (SPAKE), proposed by Abdalla and Pointcheval [6], are definitely the most efficient variants, with a security analysis in the BPR framework. And it works in any group where the CDH assumption holds: two full exponentiations and one small exponentiation, plus a multiplication and a division for each user, and just one group element to send in each direction, for the basic SPAKE presented on Figure 1. However, the above simple construction is secure for non-concurrent executions only. The password has to be added to the last key derivation step to handle concurrent executions. More importantly, forward-secrecy has never been proven for these constructions, while it is essential to guarantee the privacy of a past communication even if the password is later leaked.

Katz, Ostrovsky and Yung [34] proposed the first practical scheme, but still less efficient than above schemes, in the standard model with a common reference string, followed by a generalization from Gennaro and Lindell [28,27], using the power of smooth-projective hash functions [25], in the BPR framework. Many variations [22,4,37,31,38] have thereafter been proposed, to achieve security in the UC framework, to improve round efficiency, or to rely on new assumptions. But the most efficient, in the standard model, is definitely the recent "Simple Password-Only Key Exchange" protocol (SPOKE) [2,1], where the players have globally to compute less than 17 exponentiations and to send 6 group elements. The security just relies on the DDH assumption. Even if this is quite efficient in the standard model, this is still too

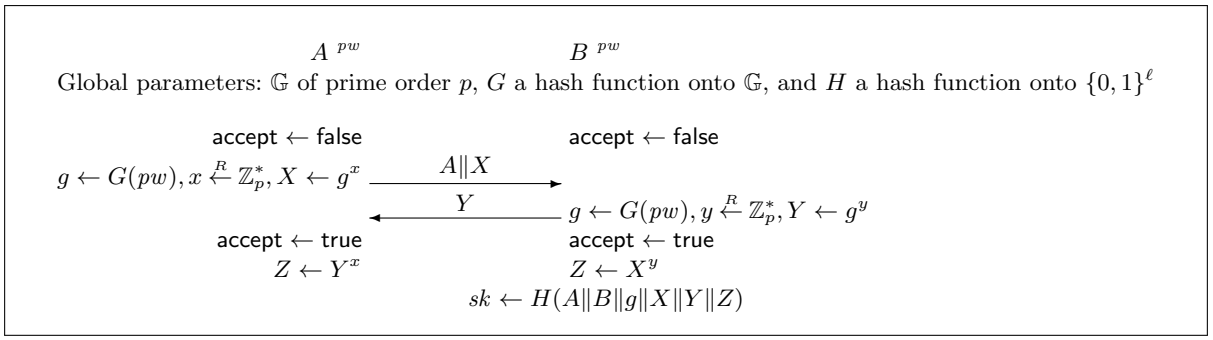

Figure 2. Simple Password Exponential Key Exchange

costly in constrained environments, where constructions proven secure in ideal models are still preferable.

Whereas the huge majority of the protocols rely on a Diffie-Hellman-like assumption, some schemes have also been proposed on factoring-related assumptions [41,43,24,30]. However, because of the huge parameters of factoring-based protocols, efficiency is far from SPAKE and even SPOKE.

Other alternatives are the *Secure Remote Password* protocol (SRP) [50] and the *Simple Password Exponential Key Exchange* protocol (SPEKE) [33], presented in Figure 2, that have been standardized, and the latter has even been analyzed in [42] in the BPR model, under the CDH assumption in the random oracle model. A specific instantiation has also been proposed with SAE [32].

While the security analysis [42] of SPEKE relies on the CDH assumption in the random oracle, the instantiation targets the specific prime-order subgroup of the squares in the finite field $\mathbb{Z}_p^*$, where $p = 2q+1$, with both p and q being large primes. There is no easy way to extend this instantiation to more efficient groups, such as elliptic curves. The main issue is to map the passwords into the group (modeled by G on Figure 2). As a consequence, this requires huge parameters, similar to factoring-based protocols.

While several constructions of VPAKE protocols were proposed in the early 2000, the first formal security model has been described in [29], with an ideal functionality in the UC framework [20,21] using a generic conversion (the so-called Ω-method). Unfortunately, the definition makes sense in ideal models only, such as the random oracle model [9]. More recently, the notion of password hashing [14] formalizes the way in which a verifier can be computed from a password, and the strong security it should guarantee in case of server compromise. This defines the notion of *tight one-wayness* which says that extracting a password for one verifier chosen among a set of verifiers should take nearly as long as hashing 2^β passwords, where β is the min-entropy of the password distribution $\mathcal{D}$. We recall that the min-entropy of a distribution is a very conservative way of measuring the unpredictability, or the number of random bits: it is defined as the negative 2-logarithm of the probability of the most likely element. The password hashing method is quite strict to allow the user to run the first flow of the protocol without knowing the salt, which is later sent by the server. Kiefer and Manulis [39] later extended this definition requiring the salt to be known by the user from the beginning. Depending on the salt, this might imply a pre-flow from the server, in order to send the salt to the user, or the server plays first, which is not always optimal in practice since the user is the one who wants to initiate a communication.

Cash *et al.* [23] proposed a new computational hard problem, the twin Diffie-Hellman problem, which is at least as hard as the ordinary Diffie-Hellman problem, but that additionally allows a trapdoor test to efficiently decide on twin Diffie-Hellman pairs. Based on this problem, they presented a verifier-based version of the SPAKE protocol [6], actually the SPAKE2 variant,

they thus called SPAKE2+. While it has been proven secure against server compromise, as for SPAKE, forward-secrecy has never been considered. Note that both SPAKE2 and SPAKE2+ have been considered under IEFT standardization [40]. Another related work is AugPAKE (for Augmented Password-Authenticated Key Exchange), proposed by Shin and Kobara [45,**?**,**?**]. According to the security proof [48], this is a secure PAKE protocol that resists to server compromise, under the Strong-Diffie-Hellman assumption [15]. None of them make an explicit use to a secure password hashing method to compute the verifier from a salt. This could be done, but at the cost of an additional flow.

1.3 Objectives and Contributions

As said above, EKE-like schemes in the random oracle model and constructions based on smooth-projective hash functions in the standard model are the two main streams in the literature. In the following, we revisit the third family of SPEKE-like protocols, in order to make it work in any group where a CDH-like assumption holds, such as elliptic curves. We expect it to lead to new efficient instantiations, at least as good as SPAKE.

Actually, our new instantiation of the random oracle, in order to map a password into the group, is inspired from SPAKE recalled on Figure 1, with two independent bases (hence the name TBPEKE, for *Two-Basis Password Exponential Key Exchange*). Its security will rely on a DLin-like assumption, which can hold in many kinds of groups, such as elliptic curves. There is no need of random oracle that maps onto group elements. It can thus be instantiated in any group, in an efficient way.

In a second step, we make it verifier-based at a quite low additional cost, and name it VTBPEKE, for *Verifier-based Two-Basis Password Exponential Key Exchange*). To this aim, we use the revised version of password-hashing from [39]. Its tight one-wayness has been proven in [14]. It requires the knowledge of the salt to start the key exchange protocol. As said above, this is not always optimal since this might make the server start the protocol, or one has to use a deterministic salt (such as server-user identities) and one has to assume the user can perfectly remember it. However, we additionally provide explicit user-authentication to the server, which anyway requires 3 flows when the user initiates the protocol and thereafter confirms his knowledge of the session key.

As a consequence, our VTBPEKE protocol is quite efficient from the communication point of view (number of flows), whatever kind of salt is used, and just requires 4 exponentiations on the user side (less than twice as much as the TBPEKE protocol).

2 Security Model

At the same time, Bellare, Pointcheval and Rogaway [8], and Boyko, MacKenzie and Patel [17] first formalized security of Password-Authenticated Key Exchange, in two different frameworks. Later, Canetti, Halevi, Katz, Lindell and MacKenzie [22] provided an ideal functionality in the Universally Composable (UC) security framework [21].

For the sake of efficiency, we focus on the weaker BPR security model, instead of UC. This is a *Find-then-Guess* game, in the indistinguishability-based framework where an adversary should not be able to get an advantage significantly greater than q_S/N in distinguishing a random session key from a real session key, if q_S is the number of active sessions against an honest user and N the size of the dictionary. It has thereafter been improved to the *Real-or-Random* scenario [5]. More precisely, the adversary is given access to oracles:

- Execute-queries model passive attacks of execution between honest players;
- Send-queries model active attacks against honest players;
- Corrupt-queries model corruptions with the leakage of long-term secrets, in order to study forward-secrecy;

– Reveal-queries model bad uses of session keys and thus the leakage of ephemeral secrets;
– and Test-queries model the semantic security of the session key, with a real or random answer.

In the Find-then-Guess scenario, only one Test-query can be asked, whereas in the Real-or-Random scenario many Test-queries can be asked with either always-real or always-random answers. The latter is clearly at least as strong as the former: while the former security model shows that sessions keys are individually indistinguishable from random, the latter shows that the session keys are globally indistinguishable from random, and independent from each other. It is well-known that both scenarios are polynomially equivalent for encryption schemes [7], but with a linear loss in the number of Test-queries. This makes them quite different for PAKE, where the advantage should remain in q_s/N, whatever the number of Test-queries. It has then been showed [5] that in this Real-or-Random scenario, Reveal-queries are not useful anymore, hence simplifying the security game (multiple Test-queries and no Reveal-queries). There are natural restrictions:

– in a key exchange protocol, when everything works fine, two partners should agree on the same key, while in the random case the Test-oracle would answer independent random keys. One can simply prevent the adversary from testing two partners;
– when the adversary knows the password (after a corruption) one cannot guarantee anymore the secrecy of the future session keys. Therefore, Test-queries on sessions that completed after corruptions are forbidden. Anyway, the forward-secrecy just considers the secrecy of the session keys agreed before the corruptions.

Note that this is a slight variant of the so-called *weak corruption* model in BPR, since the long term secrets (passwords) only are leaked, and not the internal states, in case of corruption, but this is the important notion for the *forward-secrecy*.

We stress that the main difference with the UC security notions is the assumption about the password distribution: we usually consider a uniform distribution, hence the optimal q_s/N bound. We could extend the result to the min-entropy of the passwords, or by using the probability to be in the most probable set of q_s passwords, as in [19].

3 Variants of SPEKE

In the same vein as SPAKE instantiates the ideal cipher model $\mathcal{E}_{pw}(X)$ by $X \cdot U^{pw}$ for a fixed element U, our goal is to find an efficient instantiation of $G(pw)$, in such a way that it leads to a random generator in the group $\mathbb{G}$.

3.1 A Naive Construction

The first natural instantiation is $g \leftarrow h^{pw}$ for a fixed generator h. But an easy off-line dictionary follows from a single active attack: the adversary impersonates A to B and sends $X \leftarrow h^z$ for a random z. Bob sends $Y \leftarrow (h^{pw})^y$ for the correct password pw and computes $Z \leftarrow X^y$, that can be checked after a Test-query. The adversary can indeed now try any password π, and compute the candidate $T \leftarrow Y^{z/\pi} = ((h^{pw})^y)^{z/\pi} = (h^y)^{z(pw/\pi)} = (h^z)^{y(pw/\pi)} = (X^y)^{pw/\pi} = Z^{pw/\pi}$, which is indeed Z if $\pi = pw$. It then leads to the correct sk.

3.2 A Secure Construction: TBPEKE

The problem in this construction is that from the built g, the adversary can extract the correct x for each tentative password π. This has to be prevented, and the candidate $g \leftarrow U \cdot V^{pw}$, where U and V are two random, independent, and fixed group elements, looks appropriate.

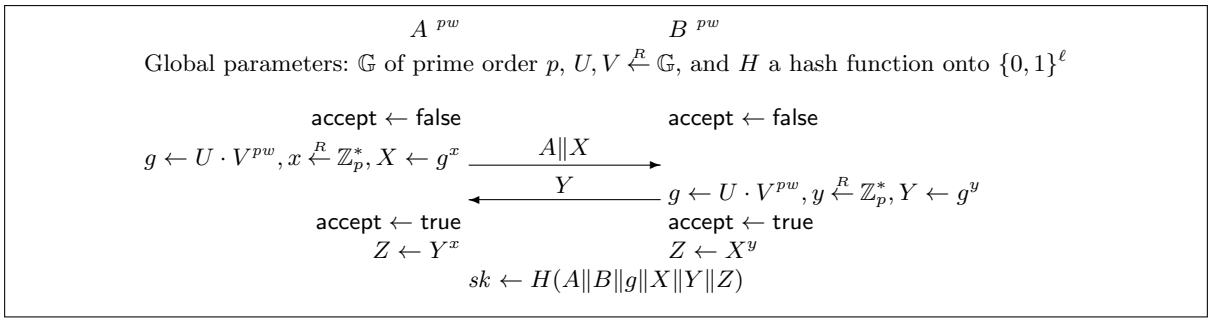

Figure 3. Two-Basis Password Exponential Key Exchange (**TBPEKE**)

The resulting scheme, called TBPEKE for *Two-Basis Password Exponential Key Exchange*, is presented on Figure 3.

As SPAKE, it instantiates the ideal primitive in the body of the protocol with a simple product that involves fixed but random group elements and the password. The complexity is quite similar: two full exponentiations and one small exponentiation, plus a multiplication for each user, and just one group element to send in each direction. There is one division and one small exponentation less to compute for each user than in SPAKE, and much smaller elements to exchange, compared from the original SPEKE protocol analyzed in the random oracle model in [42] that works in finite fields only. We can indeed hope this construction to work in any group where the CDH assumption holds (or a similar assumption), which would allow elliptic curves with 256-bit long group elements: 32 bytes only have to be sent in each direction. Eventually, as in SPAKE, only one random oracle is required for the final key derivation.

4 Security Analysis of TBPEKE

Before providing security results for our new construction, let us precise the computational assumptions.

4.1 Assumptions

- The **Computation Diffie-Hellman** (CDH) **Assumption** states that, given three random generators g, $X = g^x$, and $Y = g^y$, in a cyclic group $\mathbb{G}$ of prime order p, it is hard to fond $\mathsf{DH}_g(X, Y) = g^{xy}$.
- The **Decisional Linear** (DLin) **Assumption** states that, given six random generators f, g, h, and $X = f^x$, $Y = g^y$, $Z = h^z$, in a cyclic group $\mathbb{G}$ of prime order p, it is hard to decide whether $z = x + y \bmod p$.
- The **Simultaneous Diffie-Hellman** (SDH) **Assumption** states that, given three random generators X, $g = X^a$, and $h = X^b$ in a cyclic group $\mathbb{G}$ of prime order p, it is hard to find $Y \neq 1$ and R, S that simultaneously satisfies $R = \mathsf{DH}_g(X, Y) = Y^{1/a}$ and $S = \mathsf{DH}_h(X, Y) = Y^{1/b}$.

Reduction of the SDH Assumption to the DLin Assumption. We first show that the SDH assumption, on which our protocol will rely, can be reduced to the well-known DLin assumption, that is widely assumed to hold in pairing-friendly settings [16].

Theorem 1. *In a pairing-friendly setting, the* DLin *assumption implies the* SDH *assumption:* $\mathsf{Adv}^{\mathsf{dlin}}(t) \geq \mathsf{Succ}^{\mathsf{sdh}}(t)$.

Proof. On input a DLin instance, f, g, h, and $X = f^x$, $Y = g^y$, $Z = h^z$ (if $b = 0$, then z is random, else $z = x + y$), one runs the SDH adversary on inputs $X' \leftarrow h$, $g' \leftarrow f = X'^a$, and $h' \leftarrow g = X'^b$

(where we formally set $a = 1/x \bmod p$ and $b = 1/y \bmod p$). The adversary outputs $Y', R' = Y'^{1/a}, S' = Y'^{1/b}$ with probability ε, which validity can be checked: $e(f, R') \stackrel{?}{=} e(h, Y') \stackrel{?}{=} e(g, S')$. If this test is not satisfied, and so $\mathsf{test} = \mathsf{false}$, our algorithm outputs a random value b'.

If the test is satisfied, and so $\mathsf{test} = \mathsf{true}$, using the bilinear map e:

$$e(X, R') = e(f^x, Y'^{\frac{1}{a}}) = e(f^{\frac{1}{a}}, Y'^x) = e(h, Y'^x) = e(h^x, Y')$$
$$e(Y, S') = e(g^y, Y'^{\frac{1}{b}}) = e(g^{\frac{1}{b}}, Y'^y) = e(h, Y'^y) = e(h^y, Y').$$

Then, we have $e(X, R') \times e(Y, S') = e(h^{x+y}, Y')$, which is thus $e(Z, Y')$ if and only if the DLin instance is a linear one. Then the algorithm outputs $b' = 1$ in the positive case, and $b' = 0$ in the negative case. Then, $\Pr[b' = b]$ is equal to

$$\Pr[b' = b|\mathsf{test}] \times \Pr[\mathsf{test}] + \Pr[b' = b|\neg\mathsf{test}] \times \Pr[\neg\mathsf{test}]$$
$$= \Pr[\mathsf{test}] + \frac{1}{2} \times (1 - \Pr[\mathsf{test}]) = \frac{1}{2} + \frac{\Pr[\mathsf{test}]}{2}$$

and $\Pr[\mathsf{test}] = \mathsf{Succ}^{\mathsf{sdh}}(\mathcal{A})$, while $\mathsf{Adv}^{\mathsf{dlin}}(t) \geq 2 \times \Pr[b' = b] - 1$. □

As a consequence, the SDH assumption is reasonable in pairing-friendly settings, and we now analyze it in the generic group model, which validates it on classical elliptic curves.

Intractability of the SDH Assumption in the Generic Model. We can also directly study the SDH assumption in the generic group model [49], where the adversary is given the representation of X, that is a generator, $g = X^a$ and $h = X^b$. Only group operations are possible, which lead to elements of the form $X^{P(a,b)}$, where $P \in \mathbb{Z}_p[A, B]$ is a polynomial of total degree 1, in the two variables A and B. We identify X with the constant polynomial $P_0 = 1$, g with the polynomial $P_1 = A$, and h with the polynomial $P_2 = B$. For a group operation (the product between two elements), we simply generate the new polynomial (the sum of the polynomials to which the two elements were identified). If the polynomial is a new one, a new random representation is given as a result to the group operation. Otherwise, the same representation as before is sent.

This simulation of group operations is perfectly indistinguishable from the real execution unless two polynomials are distinct while their evaluations in (a, b) are the same: at most $q + 3$ polynomials are involved, where q is the number of group operations, and thus at most $(q + 3)(q + 4)/2$ differences, for which the probability to evaluate to 0 on a random pair (a, b) is $1/p$. Hence, the simulation is perfect but with probability at most $(q + 3)(q + 4)/2p$.

Except in the bad above cases, let us evaluate the probability the adversary outputs a valid tuple (Y, R, S), where Y is identified to the polynomial $\alpha + \beta A + \gamma B$, R to the polynomial $\alpha_1 + \beta_1 A + \gamma_1 B$, and S to the polynomial $\alpha_2 + \beta_2 A + \gamma_2 B$. This tuple is valid $Y = R^a = S^b$, which traduces to $\alpha + \beta A + \gamma B = \alpha_1 A + \beta_1 A^2 + \gamma_1 AB = \alpha_2 B + \beta_2 AB + \gamma_2 B^2$. This implies $\alpha = \beta = \gamma = \alpha_1 = \beta_1 = \gamma_1 = \alpha_2 = \beta_2 = \gamma_2 = 0$, and thus $Y = 1$, which is not allowed. So the probability to break the SDH assumption with a generic attack is bounded by $(q + 3)(q + 4)/2p$ after q group operations. This can be upper-bounded by $q^2/2p + 10/p$.

Note that using the same kind of argument, the probability to break the CDH assumption with a generic attack is bounded by $(q + 2)(q + 3)/2p$ after q group operations. This can be upper-bounded by $q^2/2p + 6/p$.

Gap Problems. In the security analysis of the forward-secrecy, the simulator will need a DDH oracle. The two above problems will then become gap problems, where the algorithm has to find the same outputs but with the additional access to the DDH oracle, hence the GDH and the GSDH problems. We will denote $\mathsf{Succ}^{q-\mathsf{gdh}}(t)$ and $\mathsf{Succ}^{q-\mathsf{gsdh}}(t)$ the best success probabilities an adversary can get against the CDH and the SDH problems, respectively, within time t and with at most q DDH-oracle queries. In the generic model, the complexities of the attacks remain the same since decisional oracles do not help to solve computational problems.

4.2 Security Results for TBPEKE

In Sections 6.2 and 6.3, we provide the security result and the full proof of the verifier-based protocol, in the Real-or-Random security model, which is even stronger than what is actually required for the TBPEKE since we allow the adversary to choose the salt s. We thus postpone the security analysis to the verifier-based protocol.

5 VTBPEKE: Verifier-based TBPEKE Protocol

5.1 Password Hashing Scheme

In [14], Benhamouda and Pointcheval proposed a methodology with a password hashing scheme, which defines a pre-hash value $\mathcal{P}$ (from the secret password pw) and a hash value $\mathcal{V}$ (the verifier, from both the secret password pw and the public salt s) which can be verified from the pre-hash value, the salt, and an additional trapdoor. This is a complex mechanism which is motivated by a one-round protocol, and thus when the client does not need to know the salt when he sends his (first) flow. Kiefer and Manulis [39] later extended this definition requiring the salt from the beginning for the client, which implies a pre-flow from the server, in order to send the salt to the user, or a deterministic salt that can be known by the client. This will not be a problem in our case if the client can initiate the proof of knowledge of the password without knowing the salt. We thus use this variant.

We first recall the naive password hashing scheme from [14], just to explain the idea. With a hash function $\mathcal{H}$ onto $\{0,1\}^{2k}$ and a salt $s \in \{0,1\}^{2k}$, the verifier is $\mathcal{V} = \mathcal{H}(s, pw)$ while the pre-hash value is $\mathcal{P} = pw$. It has been proven, when the password is not too large, which means $pw \in \mathcal{D} \subseteq \{0,1\}^n$ and $n < k$, where k is the security parameter, that the following properties hold.

- *Second pre-image resistance*, which says that only one password should match a given verifier $\mathcal{V}$ and a salt s: For any password $pw \in \{0,1\}^n$ and any salt s, the probability there exists another password $pw' \in \{0,1\}^n$ such that $\mathcal{H}(s, pw) = \mathcal{H}(s, pw')$ is less than $2^n/2^{2k} \leq 1/2^k$, since $\mathcal{H}$ has values in $\{0,1\}^{2k}$, so the second pre-image resistance statistically holds.
- *Entropy preservation*, which says that for any s, the distribution of $\mathcal{V}$ has the same min-entropy β as pw: If the adversary makes q_H queries to the hash function, the probability of a collision is less than $q_H^2/2^{2k}$, in the random oracle model. Let us now suppose that there are no collisions. We denote by s the salt and by H the hash value returned by the adversary, as its guess for $\mathcal{H}(s, pw)$, while $pw \xleftarrow{R} \mathcal{D}$. Two situations appear:
  - either H is not an answer to any pair (s, x) asked by the adversary to $\mathcal{H}$. Then, either (s, pw) has been asked by the adversary to the random oracle, in which case $\mathcal{H}(s, pw) \neq H$, or (s, pw) has never been asked to the random oracle, and so $\mathcal{H}(s, pw)$ is a random string in $\{0,1\}^{2k}$ and is equal to H with probability $1/2^{2k}$;
  - or H is the answer to $\mathcal{H}(s, x)$. Then, the probability that $pw = x$, is at most $2^{-\beta}$, and if $pw \neq x$, the probability $\mathcal{H}(s, pw) = H$ is at most $1/2^{2k}$, as above.

  Therefore, one can guess $\mathcal{V}$ with probability bounded by $2^{-\beta} + (q_H^2 + 1) \times 2^{-2k}$.
- *Tight one-wayness*, which is the most important notion, since it says that extracting just one password $\mathcal{P} = pw$ from a huge list of pairs $(s, \mathcal{V})$ needs a computational time linear in 2^β, where β is the min-entropy of the distribution $\mathcal{D}$ of the passwords: First, the probability that two salts are equal is less than $n_s^2/2^{2k}$ (where n_s is the number of salts)[1]. When there is no collision, each query to $\mathcal{H}$ enables to check at most one password for only one salt. So the probability to extract pw from $\mathcal{V}$ and s is bounded by $q_H \times 2^{-\beta} + n_s^2 \times 2^{-2k}$, since each password appears with probability at most $2^{-\beta}$.

Unfortunately, with this password hashing scheme, one cannot (efficiently) prove the knowledge of $\mathcal{P}$ that leads to $\mathcal{V}$ from s.

[1] if the salt is the pair server-user identities, this probability is 0, whatever the length of the salt is.

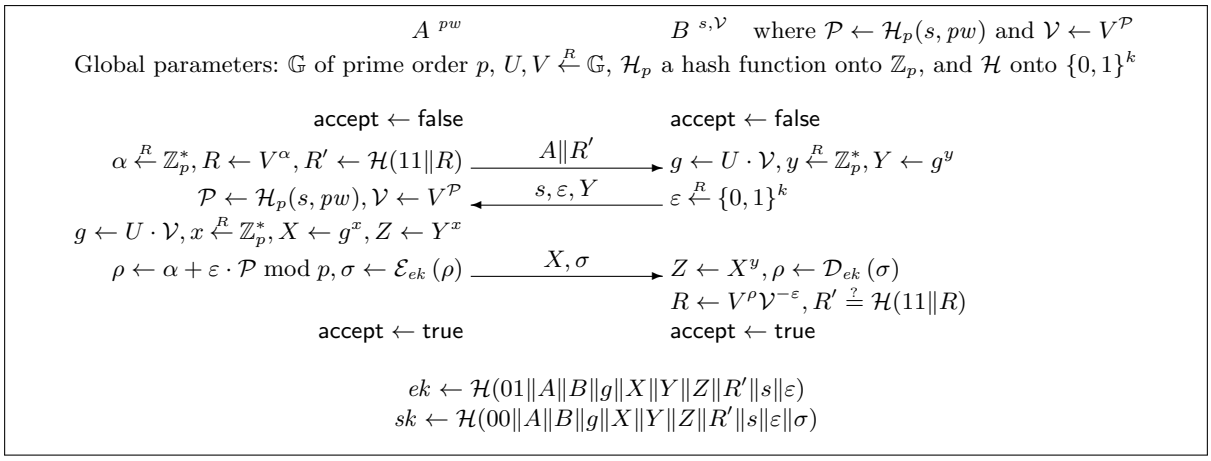

Figure 4. Verifier-based Two-Basis Password Exponential Key Exchange

5.2 Our Password Hashing Scheme

We thus define another password hashing scheme, with a hash function $\mathcal{H}_p$ onto $\mathbb{Z}_p$ and a salt $s \in \{0,1\}^{2k}$, the pre-hash is $\mathcal{P} = \mathcal{H}_p(s, pw)$, while the verifier is $\mathcal{V} = h^{\mathcal{P}}$, where h is a random generator in $\mathbb{G}$:

- *Second pre-image resistance*: as above, since the exponent is an injection, a collision on $\mathcal{V}$ is a collision on $\mathcal{P} = \mathcal{H}_p(s, pw)$ which happens with probability less than $2^n/p \leq 1/2^k$.
- *Entropy preservation*: again, because of the injectivity of the exponentiation, one can guess $\mathcal{V}$ with probability bounded by $2^{-\beta} + (q_H^2 + 1)/p$.
- *Tight one-wayness*: unless one can solve faster the DL problem, the probability to get one $\mathcal{P}$ from a huge list of pairs $(s, \mathcal{V})$ is bounded by $q_H \times 2^{-\beta} + n_s^2/p$.

In addition, one can efficiently prove the knowledge of $\mathcal{P}$ that leads to $\mathcal{V}$, using a Schnorr-like proof of knowledge [44].

5.3 Description of VTBPEKE

The idea of the verifier-based version our TBPEKE is to start from a secret password pw known to the client only, and a public and common salt s, but not assumed to be remembered by the client. Hence, once s is known by the client, he can compute the verifier $\mathcal{V} = V^{\mathcal{H}_p(s,pw)}$ that has been stored by the server: this is the common secret used by the client and the server to run our previous TBPEKE, with an additional proof of knowledge of $\mathcal{P} = \mathcal{H}_p(s, pw)$ by the client, using a Schnorr-like proof of knowledge [44], with the final answer encrypted under an ephemeral secret key derived from the TBPEKE final key. The resulting protocol is described on Figure 4.

We stress that pw is the only value known by the client (as well as the global parameters of the system, as $\mathbb{G}$, U, V, and $\mathcal{H}_p, \mathcal{H}$ which are hard-coded in the software). The client-server specific salt s is not required to be known in advance by the client, but just sent to him by the server. Indeed, the server stores, for each client, the salt s and the verifier $\mathcal{V}$. Once the client knows s, he can also compute $\mathcal{V}$, and they can both run the TBPEKE on this common value.

Note that an adversary can send a wrong salt, but this does not alter the security of the protocol, as shown below.

6 Security Analysis of VTBPEKE

6.1 Discussions

The intuition behind this protocol is

– first, the basic TBPEKE with $\mathcal{V}$ as common password does not leak any information about $\mathcal{V}$ against a passive adversary;
– an additional proof of knowledge of $\mathcal{P}$, to prevent an attacker, after compromised the server, from cheating the server by impersonating the client via using $\mathcal{V}$ without knowing $\mathcal{P}$ and so the password pw. This is a Schnorr-like proof which consists of a random hash R' and a random challenge ε together with a ciphertext σ under the indistinguishable key ek. This way, it does not leak any information about $\mathcal{V}$ either;
– an adversary trying to impersonate the server will have to guess $\mathcal{V}$ to have a chance to learn something, while an adversary trying to impersonate the client will have to guess pw or $\mathcal{P}$, because of the proof of knowledge.

About the additional encryption $\sigma \leftarrow \mathcal{E}_{ek}(\rho)$, one might wonder if this is required or not. Let us assume we let the client send ρ in clear, then this value would satisfy $R' = \mathcal{H}(11\|V^{\rho}V^{-\mathcal{P}\varepsilon})$, which would lead to an off-line dictionary attack for a passive adversary. However, we additionally have to prove this encryption is enough for the security.

6.2 Forward-Secrecy & Verifier-Based

Theorem 2. *Under the* GDH *and* GSDH *assumptions, the VTBPEKE (see Figure 4) is a forward-secure VPAKE: the best advantage an adversary can get in the Real-or-Random security game is bounded by*

$$\mathsf{Adv}(\mathcal{A}) \leq \frac{q_s}{N} + q_{\mathcal{P}}^2 \times \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gsdh}}(t) + \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gdh}}(t) + \frac{q_{\mathcal{P}}^2 + q_S^2}{p} + q_S \cdot \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind}}(t),$$

where $q_S = q_s + q_e$ is the global number of sessions (q_e for the passive sessions and q_s for the active sessions), $q_{\mathcal{H}}$ is the number of queries to $\mathcal{H}$ and $q_{\mathcal{P}}$ is the number of queries to $\mathcal{H}_p$.

Since both the problems GDH and GSDH are hard in the generic group model, with the best attacks leading to a success probability bounded by $q^2/2p + 10/p$ and $q^2/2p + 6/p$ respectively, where q is the number of group operations in $\mathbb{G}$ (either the number of additions of points, in an elliptic curve, or the number of multiplications, in a multiplicative subgroup of a finite field), we can additionally state:

$$\mathsf{Adv}(\mathcal{A}) \leq \frac{q_s}{N} + q_S \cdot \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind}}(t) + \frac{q_{\mathcal{P}}^2 \times (q^2 + 20) + q_S^2}{p},$$

Thanks to the secure password hashing scheme, in case of corruption of the server, the adversary will not be able to extract the passwords too quickly.

6.3 Security Proof

We do the proof with a series of games, starting from the real game $\mathbf{G}_0$, which makes use of a random oracle $\mathcal{H}$ and a symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$.

We say that two users (a client C and a server S) are *compatible* if they use the same salt-verifier pair $(s, \mathcal{V})$. They are initially all set as the same for each client and the server, but a corruption of the server with a new salt or a new verifier can replace them by different values: C and S are then said *incompatible*. Note that as in [35,**?**], the compatibility is defined at the beginning of the execution of the protocol (by uploading passwords in the local memory), which means that even in case of password change in the database during the protocol, this does not affect the passwords used during this execution.

Game $\mathbf{G}_1$: In this game, one simulates the random oracle $\mathcal{H}$ on new queries $(0, 1, C, S, g, X, Y, Z, R', s, \varepsilon)$, $(0, 0, C, S, g, X, Y, Z, R', s, \varepsilon, \sigma)$ or $(1, 1, R)$ with random answers, either ek, sk, or $R' \xleftarrow{R} \{0,1\}^\ell$. For keeping consistent, one stores $((0, 1, C, S, g, X, Y, Z, R', s, \varepsilon), ek)$, $((0, 0, C, S, g, X, Y, Z, R', s, \varepsilon, \sigma), sk)$ or $((1, 1, R), R')$ in Λ that is used to give the same answer if the same query is asked twice. One also simulates the random oracle $\mathcal{H}_p$ on new queries (s, pw): it outputs a random $\mathcal{P} \xleftarrow{R} \mathbb{Z}_p$. One then stores $((s, pw), \mathcal{P})$ in Λ_p that is used to give the same answer if the same query is asked twice. This is a perfect simulation of the random oracles $\mathcal{H}$ and $\mathcal{H}_p$: $\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A})$.
We formally restrict to $\bar{\Lambda}$ the set of the truncated tuples (C, S, X, Y, s) corresponding to the above tuples in Λ that satisfy $Z = \mathsf{DH}_g(X, Y)$, with $g = U \cdot \mathcal{V}$, for any verifier $\mathcal{V}$ of any user, or $g = U \cdot V^{\mathcal{H}_p(s,pw)}$, for any password of any user. In addition to the initial salts associated to the users, for any salt s sent for a client C (by the adversary), one computes $\mathcal{P}_C = \mathcal{H}_p(s, pw_C)$ and $g = U \cdot V^{\mathcal{P}_C}$ to build this list.

Game $\mathbf{G}_2$: Execute-queries between compatible users, before corruption.
We first deal with passive attacks (Execute-queries), in which we know from the beginning whether the players are compatible or not. In this game, we modify the way Execute-queries between compatible users, before corruption, are answered. We make C^i and S^j send the correct salt s, random group elements X and Y, but a correct proof of knowledge of $\mathcal{P}_C$, with $R' = \mathcal{H}(11\|R)$, ε, and ρ encrypted under a random key $ek \xleftarrow{R} \{0,1\}^\ell$, into σ. One also randomly draws $sk \xleftarrow{R} \{0,1\}^\ell$, and stores (C, S, X, Y, s) in Λ_1.
Unless $(*, C, S, g, X, Y, Z, s, *)$, for the appropriate g and Z, has been asked to $\mathcal{H}$, this is indistinguishable from the previous game. We thus set the event Bad1H to true (and let the adversary win) as soon as $\bar{\Lambda} \cap \Lambda_1 \neq \emptyset$. Unless Bad1H = true, the two games are indistinguishable: $\mathsf{Adv}_{\mathbf{G}_1}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) + \Pr[\mathsf{Bad1H}]$.

Game $\mathbf{G}_3$: We can now replace $R' \xleftarrow{R} \{0,1\}^\ell$ and σ by a random ciphertext. Under the indistinguishability of the encryption scheme, the two games are indistinguishable: $\mathsf{Adv}_{\mathbf{G}_2}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}) + q_e^{\mathsf{comp}} \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathcal{A})$.

Game $\mathbf{G}_4$: Execute-queries between incompatible users, before corruption.
Execute-queries between incompatible users, before corruption, are answered the same way as above, but with independent sessions keys. Unless the same Bad1H = true, the two games are indistinguishable under the indistinguishability of the encryption scheme $(\mathcal{E}, \mathcal{D})$: $\mathsf{Adv}_{\mathbf{G}_3}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) + q_e^{\mathsf{incomp}} \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathcal{A})$. As a conclusion, since the initial game, we have

$$\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) + q_e \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathcal{A}) + \Pr[\mathsf{Bad1H}].$$

We will now modify the Send-queries, where Send_0 is the start-query for a client to initiate an execution of the protocol, followed by Send_1, Send_2, and Send_3.

Game $\mathbf{G}_5$: Send_2-queries, before corruption.
We now modify the behavior of the client before corruption. To a Send_2-query (s, ε, Y), one selects $X \xleftarrow{R} \mathbb{G}$ and keys $ek, sk \xleftarrow{R} \{0,1\}^\ell$. The simulator also stores (C, S, X, Y, s) in Λ_2.
As above, an inconsistency can be detected if an $\mathcal{H}$-query has been asked with the appropriate g and Z. But this time Y has been chosen by the adversary, without knowing the real password/verifier (before corruption). More generally, we set the event Bad2H to true (and let the adversary win) as soon as $\bar{\Lambda} \cap \Lambda_2 \neq \emptyset$. Unless Bad2H = true, the two games are indistinguishable: $\mathsf{Adv}_{\mathbf{G}_4}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}) + \Pr[\mathsf{Bad2H}]$.

Game $\mathbf{G}_6$: In addition, we replace σ by a random ciphertext, which is indistinguishable since ek is random:

$$\mathsf{Adv}_{\mathbf{G}_5}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_6}(\mathcal{A}) + q_C \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathcal{A}).$$

Game $\mathbf{G}_7$: Send_3-queries, before corruption.
We now modify the behavior of the server before corruption. To a Send_3-query (X, σ), one selects a random key $sk \xleftarrow{R} \{0,1\}^\ell$, and stores (C, S, X, Y, s) in Λ_2.

As above, an inconsistency can be detected if an $\mathcal{H}$-query has been asked with the appropriate g and Z. But with X chosen by the adversary, without knowing the real password/verifier (before corruption), which is already covered by the event $\mathsf{Bad2H}$ that makes the adversary win: $\mathsf{Adv}_{\mathbf{G}_7}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_6}(\mathcal{A})$.

Game $\mathbf{G}_8$: $\mathcal{H}$-queries.

We enhance the simulation of the $\mathcal{H}$-queries with a possible $\perp$ instead of Z, which is assumed to stand for $Z = \mathsf{DH}_g(X, Y)$. Let us focus on the (g, X, Y, Z) part in the input and the output y:

- for a new query (from the simulator) involving $(g, X, Y, \perp)$, one looks for any (Z, y) for which Z would complete the query in Λ and checks whether $\mathsf{DDH}(g, X, Y, Z)$ is true or not. If this is true for a pair (Z, y), then the answer is y, otherwise a random $y \xleftarrow{R} \{0,1\}^\ell$ is drawn and output. The query, completed with y, is stored in Λ;
- for a new query involving (g, X, Y, Z) such that completed query-answer $((g, X, Y, \perp), y) \in \Lambda$, one checks whether $\mathsf{DDH}(g, X, Y, Z)$ is true or not. If this is true, then the answer is y ($\perp$ is replaced by Z in the list), otherwise a random $y \xleftarrow{R} \{0,1\}^\ell$ is drawn and output, and the query, completed with y, is stored in Λ.

Of course, as before, one gives the same answer if the same query is asked twice. This is a perfect simulation of the random oracle $\mathcal{H}$, but with access to a DDH oracle which, on input (g, X, Y, Z), checks whether $Z = \mathsf{DH}_g(X, Y)$ or not. The number of DDH-oracle-queries will be bounded by $q_{\mathcal{H}}$: $\mathsf{Adv}_{\mathbf{G}_8}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_7}(\mathcal{A})$.

Game $\mathbf{G}_9$: Send_3-queries, after corruption.

We now deal with the corruptions, and the answer to a Send_3-query (X, σ) but for a Y generated before the corruption, and so the adversary might possibly be correct for the X and g: the simulator asks for the appropriate $(g, X, Y, \perp)$-query to $\mathcal{H}$, to get sk, with $g = U \cdot \mathcal{V}_{S,C}$, using the above simulation of the random oracle $\mathcal{H}$. Thanks to the enhanced simulation of the random oracle, one will get the same answers as the adversary asked for the correct queries, with the correct Z: $\mathsf{Adv}_{\mathbf{G}_9}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_8}(\mathcal{A})$.

Game $\mathbf{G}_{10}$: Send_1-queries, before corruption.

We can complete the behavior of the server before corruption, since Z is not needed anymore to answer Send_3-queries. To a Send_1-query (A, R'), one selects $Y \xleftarrow{R} \mathbb{G}$: $\mathsf{Adv}_{\mathbf{G}_{10}}(\mathcal{A}) = \mathsf{Adv}_{\mathbf{G}_9}(\mathcal{A})$.

Game $\mathbf{G}_{11}$: In the above game, one can remark that the passwords/verifiers are not needed for the simulation of the honest players, unless they have been corrupted, but just required to build the list $\bar{\Lambda}$, and to evaluate the events $\mathsf{Bad1H}$ and $\mathsf{Bad2H}$. This is enough to evaluate them at the very end only. However, the server must have generated a salt for each client, from the beginning.

So, in the final game, the passwords/verifiers are not known at the beginning, but just at the corruption time or at the very end only:

- $\mathsf{Execute}(C^i, S^j)$-queries: the simulator randomly selects $X, Y \xleftarrow{R} \mathbb{G}$, and stores (C, S, X, Y, s) in Λ_1. If the users are compatible, they are given the same random key $sk = sk_C = sk_S \xleftarrow{R} \{0,1\}^\ell$. If they are incompatible, they are given two independent random keys $sk_C, sk_S \xleftarrow{R} \{0,1\}^\ell$;
- $\mathsf{Send}_0(C^i)$-queries (which asks C to initiate an execution of the protocol): the simulator randomly selects $\alpha \xleftarrow{R} \mathbb{Z}_p^*$, sets $R \leftarrow V^\alpha$ and $R' \leftarrow \mathcal{H}(11\|R)$, and outputs (C, R');
- $\mathsf{Send}_1(C, R')$-queries: before a corruption, the simulator selects the appropriate salt s and a random challenge ε, but randomly chooses $Y \xleftarrow{R} \mathbb{G}$. Otherwise it randomly selects $y \xleftarrow{R} \mathbb{Z}_p^*$ and sets $Y \leftarrow g^y$, for $g \leftarrow U \cdot \mathcal{V}_{S,C}$;
- $\mathsf{Send}_2(s, \varepsilon, Y)$-queries: before a corruption, the simulator randomly selects $X \xleftarrow{R} \mathbb{G}$, the keys ek, $sk \xleftarrow{R} \{0,1\}^\ell$, and the ciphertext σ, and stores the tuple (C, S, X, Y, s) in Λ_2, otherwise it randomly selects $x \xleftarrow{R} \mathbb{Z}_p^*$, sets $X \leftarrow g^x$ and $Z \leftarrow Y^x$, for $g \leftarrow U \cdot V^{\mathcal{P}_C}$ with $\mathcal{P}_C \leftarrow \mathcal{H}(s, pw_C)$, and asks for ek and sk to $\mathcal{H}$ and answers σ correctly;

- $\mathsf{Send}_3(X, \sigma)$-queries: thee cases appear
  - before corruption, one chooses $sk_C \xleftarrow{R} \{0,1\}^\ell$;
  - after corruption, but Y has been generated before corruption, one asks for sk from $\mathcal{H}$ on the tuple $(g, X, Y, \perp)$, for $g \leftarrow U \cdot \mathcal{V}_{S,C}$;
  - after corruption, and $Y = g^y$ for $g = U \cdot \mathcal{V}_{S,C}$, one asks for sk from $\mathcal{H}$ on the tuple (g, X, Y, Z), where $Z \leftarrow X^y$.
- $\mathsf{Corrupt}(C)$ and $\mathsf{Corrupt}(S, C, \perp)$-queries: if this is the first corruption query involving C, one first chooses a random password pw (to be pw_C) and defines $\mathcal{P}_C = \mathcal{H}(s, pw)$ from the already chosen salt s, and sets $\mathcal{V}_{S,C} = V^{\mathcal{P}_C}$. One then checks for events $\mathsf{Bad1H}$ and $\mathsf{Bad2H}$, for sessions involving C;
- $\mathsf{Corrupt}(S, C, (s, \mathcal{V}))$-query: it sets $\mathcal{V}_{S,C} \leftarrow \mathcal{V}$ as well as $s_{S,C} \leftarrow s$;
- Test-queries are answered using the defined session key, and according to the bit b.

In case of collision between the X's or the Y' chosen by the simulator (either in $\mathsf{Execute}$-answer or a Send-answer), and a previously sent value (either by the simulator or the adversary), we set Coll to true (and let the adversary win). Unless a collision happens, thanks to the definition of the session ids (the entire transcripts), no instance of a player can have more than one partner (with the same session id). At the very end, or at the time of a corruption, the passwords are drawn at random, and the pre-hashes $\mathcal{P}$ and verifiers $\mathcal{V}$ are computed. As a consequence,

$$\mathsf{Adv}_{\mathbf{G}_{10}}(\mathcal{A}) \leq \mathsf{Adv}_{\mathbf{G}_{11}}(\mathcal{A}) + \Pr[\mathsf{Coll}].$$

In this last game, the way the session keys are defined is exactly the same as in the random and the real cases (chosen at random before corruption). Then the probability for $b' = b$ is exactly one half:

$$\begin{aligned}\mathsf{Adv}_{\mathbf{G}_0}(\mathcal{A}) \leq \Pr[\mathsf{Bad1H}] + \Pr[\mathsf{Bad2H}] + \Pr[\mathsf{Coll}] \\ + (q_e + q_C) \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(\mathcal{A}).\end{aligned}$$

Collisions. The probability the event Coll is upper-bounded by $(q_e + q_s)^2/p$, where q_e is the number of $\mathsf{Execute}$-queries and q_s is the number of Send-queries.

Event Bad1H. The event $\mathsf{Bad1H}$ means that for some tuple $(g = U \cdot V^{\mathcal{P}}, X = g^x, Y = g^y, Z)$, we have $Z = g^{xy}$. Let us be given a Diffie-Hellman instance $(u, v = u^a, w = y^b)$. We set $U \leftarrow u$, $V \leftarrow u^z$ for a random z, and for any random element X, Y, one chooses random x, y and sets $X \leftarrow v^x$ and $Y \leftarrow w^y$: the basis is $g = u^{1+\mathcal{P}z}$, while $X = u^{ax} = g^{ax/(1+\mathcal{P}z)}$ and $Y = u^{by} = g^{by/(1+\mathcal{P}z)}$. Therefore, $Z = g^{abxy/(1+\mathcal{P}z)^2} = (u^{ab})^{xy/(1+\mathcal{P}z)}$. We thus have solved the CDH problem with $Z^{(1+\mathcal{P}z)/xy}$, which can be checked with $q_{\mathcal{H}}$ additional DDH-oracle queries: $\Pr[\mathsf{Bad1H}] \leq \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gdh}}(t)$.

Event Bad2H. The event $\mathsf{Bad2H}$ cannot be proven to be negligible, since it can at least happen when the adversary guesses correctly the password. We have to show this cannot happen more often than once per active session. Let us assume that the adversary can make a given tuple (C, S, g, X, Y, Z, s) raise the event $\mathsf{Bad2H}$ for two distinct choices $pw_1 \neq pw_2$ of the password and so two distinct pre-hashs $\mathcal{P}_1 \neq \mathcal{P}_2$ with the same salt s (unless there is a collision on $\mathcal{H}_p$). Let us be given an instance $(u, v = u^a, w = u^b)$ for which we want to get R, $R^{1/a}$, $R^{1/b}$ (break the SDH problem). We set $U \leftarrow (v^{\mathcal{P}_2}/w^{\mathcal{P}_1})^{1/\mathcal{P}_2-\mathcal{P}_1}$ and $V \leftarrow (v/U)^{1/\mathcal{P}_1}$. With this definition, one can note that $U \cdot V^{\mathcal{P}_1} = v$ and $U \cdot V^{\mathcal{P}_2} = w$. For a random choice of X or Y in $\mathbb{G}$, one sets it to u^x for a random scalar x. Let us assume this is $X = u^x$ (but by symmetry it would be the same with Y). If (g, X, Y, Z, s) would raise the event $\mathsf{Bad2H}$ for both $\mathcal{P}_1$ and $\mathcal{P}_2$, there are Z_1 and Z_2 in the list Λ such that $Z_1 = \mathsf{DH}_v(X, Y)$ and $Z_2 = \mathsf{DH}_w(X, Y)$, which can be checked with DDH-oracle queries. For $X = u^x$: $Z_1^{1/x} = Y^{1/a}$ and $Z_2^{1/x} = Y^{1/b}$. With $R \leftarrow Y$, we have $Z_1^{1/x} = R^{1/a}$ and $Z_2^{1/x} = R^{1/b}$.

Scheme	Communication (Both Sides)	Computation (Both Sides)	Forward Secrecy	Security Model	Assumptions	Limitations
PAKE						
EKE [12]	$1\mathbb{G}$ / $1\mathbb{G}$	2E / 2E	Yes	ICM	CDH [19]	
SPAKE [6]	$1\mathbb{G}$ / $1\mathbb{G}$	2E+2sE / 2E+2sE	No	ROM	CDH	
SPAKE2 [6]	$1\mathbb{G}$ / $1\mathbb{G}$	2E+2sE / 2E+2sE	No	ROM	CDH	
SPEKE [33]	$1\mathbb{G}$ / $1\mathbb{G}$	2E / 2E	No	ROM	CDH	in $\mathbb{Z}_p^*$ only
SAE [32]	$2\mathbb{G}$ / $2\mathbb{G}$	3E / 3E	No	ROM	CDH [42]	in $\mathbb{Z}_p^*$ only
SRP [50]	$3\mathbb{G}$ / $3\mathbb{G}$	2E / 3E	No proof			in $\mathbb{Z}_p^*$ only
GL-SPOKE [2]	$4\mathbb{G}$ / $3\mathbb{G}$	10E /10E	Yes	Standard	DDH	
GK-SOPKE [2]	$2\mathbb{G}$ / $4\mathbb{G}$	8E / 9E	Yes	Standard	DDH	
TBPEKE	$1\mathbb{G}$ / $1\mathbb{G}$	2E+1sE / 2E+1sE	**Yes**	ROM	GSDH	
Verifier-based PAKE						
SPAKE2+ [23]	$1\mathbb{G}$ / $1\mathbb{G}$	5E / 5E	No	ROM	CDH	
AugPAKE [45]	$1\mathbb{G}+k$ / $1\mathbb{G}+k$	2E / 3E	No	ROM	Strong DH [48]	
VTBPEKE	$1\mathbb{G}+k+\lvert p\rvert$ / $1\mathbb{G}+k$	4E / 4E	**Yes**	ROM	GSDH	

Figure 5. Comparisons

As a consequence, let us randomly choose $pw_1 \neq pw_2$ and $\mathcal{P}_1, \mathcal{P}_2 \xleftarrow{R} \mathbb{Z}_p$. we set $U \leftarrow (v^{\mathcal{P}_2}/w^{\mathcal{P}_1})^{1/\mathcal{P}_2-\mathcal{P}_1}$ and $V \leftarrow (v/U)^{1/\mathcal{P}_1}$, and for a query $\mathcal{H}_p(s, pw_1)$ we output $\mathcal{P}_1$, and for the future $\mathcal{H}_p(s, pw_2)$ we will output $\mathcal{P}_2$. With probability $1/N^2 q_{\mathcal{P}}$, we have done the good choices for pw_1 (among N), pw_2 (among N) and s (among at most $q_{\mathcal{P}}$ queries to $\mathcal{H}(\cdot, pw_1)$): a tuple (C, S, g, X, Y, Z, s) is very unlikely to raise the event Bad2H for two distinct choices $\mathcal{P}_1 \neq \mathcal{P}_2$, but only for one, and there is one such tuple per session:

$$\Pr[\mathsf{Bad2H}] \leq \frac{q_s}{N} + \frac{q_{\mathcal{P}}^2}{p} + N^2 q_{\mathcal{P}} \times \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gsdh}}(t),$$

where t is the running time of the adversary (since the basic reduction is essentially the classical attack). However, because of the N^2 loss, in case of a large dictionary, we can simply answer $\mathcal{P}_1$ and $\mathcal{P}_2$ for two randomly chosen $\mathcal{H}_p$ queries, and with probability $1/q_{\mathcal{P}}^2$ these are the good ones:

$$\Pr[\mathsf{Bad2H}] \leq \frac{q_s}{N} + \frac{q_{\mathcal{P}}^2}{p} + q_{\mathcal{P}}^2 \times \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gsdh}}(t).$$

Conclusion. By combining all the bad cases, we have

$$\begin{aligned}\mathsf{Adv}(\mathcal{A}) \leq \frac{q_s}{N} + q_{\mathcal{P}}^2 \times \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gsdh}}(t) + \mathsf{Succ}^{2q_{\mathcal{H}}-\mathsf{gdh}}(t) \\ + \frac{q_{\mathcal{P}}^2 + q_S^2}{p} + q_S \cdot \mathsf{Adv}_{\mathcal{E}}^{\mathsf{ind}}(t),\end{aligned}$$

where $q_S = q_s + q_e$ is the global number of sessions (q_e for the passive sessions q_s for the active sessions), $q_{\mathcal{H}}$ is the number of queries to $\mathcal{H}$ and $q_{\mathcal{P}}$ is the number of queries to $\mathcal{H}_p$.

7 Parameters and Efficiency

On Figure 5, we list a number of representative PAKE and VPAKE protocols, and compare their security levels as well as the efficiency, from both the communication and the computation points of view. For communication, we count the number of group elements ($\mathbb{G}$), scalars ($|p|$), and bit strings of length k, the security parameter, in both directions, but ignore the salt (for VPAKE). For the computations, we focus on exponentiations (E is the cost of one exponentiation, while sE is the cost of an exponentiation to a small scalar). While SRP does not admit any formal security analysis, the authors recommend the use of a group of order p in a finite field $\mathbb{Z}_q^*$ with a

k	$\lvert p\rvert$
64	320
80	400
112	560
128	640

Figure 6. Bit-length of the order of the groups

safe prime $q = 2p + 1$, which leads to a very inefficient construction, as for any implementation that requires discrete logarithm in finite fields.

Our TBPEKE is the most efficient PAKE (it even additionally provides forward secrecy): the EKE construction is only proven in the ideal-cipher model (ICM), with a symmetric encryption scheme over the cyclic group $\mathbb{G}$, which is not efficiently implementable.

As AugPAKE, our VTBPEKE also allows some off-line pre-computation independent of the password and the salt, on the client side. They have a similar communication complexity. Our VTBPEKE is a bit more costly, but it is proven to guarantee the forward-secrecy in the Real-or-Random security game, whereas AugPAKE is just proven secure in the Find-then-Guess security game, without forward-secrecy.

For a PAKE or VPAKE, one expects the advantage of the adversary to be bounded by $q_s/N+\varepsilon$. With a safe and efficient symmetric encryption scheme (as AES), we can ignore $q_S \cdot \mathsf{Adv}^{\mathsf{ind}}_{\mathcal{E}}(t)$. And we need to make $q_{\mathcal{P}}^2 \times (q^2 + 20) + q_S^2 \leq p \cdot 2^{-k}$, for the security parameter k: $p \geq 2^k \times (q_{\mathcal{P}}^2 \times (q^2 + 20) + q_S^2)$. A safe bound is $p \geq 2^{5k}$ (See Figure 6).

8 Conclusion

We have proposed a quite efficient forward-secure verifier-based password-authenticated key exchange: the computations on the client side just consist of 4 exponentiations, and the global communication is approximately $14k$, for a security parameter k, which is approximately 224 Bytes for a 128-bit security level. It follows the line of SPEKE that has not been studied as much as EKE, while it is also an interesting family, as this work shows. We have indeed proven that

- off-line dictionary attacks are only possible after the compromise of the server, but the computational cost will be linear in N, the cardinality of the dictionary;
- on-line dictionary attacks also require a linear number of active attacks to guess the password;
- thanks to the forward-secrecy, even when the password or the verifier are known, the privacy of previous sessions still holds.

References

1. M. Abdalla, F. Benhamouda, and D. Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. Cryptology ePrint Archive, Report 2014/609, 2014. http://eprint.iacr.org/2014/609.
2. M. Abdalla, F. Benhamouda, and D. Pointcheval. Public-key encryption indistinguishable under plaintext-checkable attacks. In J. Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 332–352. Springer, Heidelberg, Mar. / Apr. 2015.
3. M. Abdalla, D. Catalano, C. Chevalier, and D. Pointcheval. Efficient two-party password-based key exchange protocols in the UC framework. In T. Malkin, editor, *CT-RSA 2008*, volume 4964 of *LNCS*, pages 335–351. Springer, Heidelberg, Apr. 2008.
4. M. Abdalla, C. Chevalier, and D. Pointcheval. Smooth projective hashing for conditionally extractable commitments. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 671–689. Springer, Heidelberg, Aug. 2009.
5. M. Abdalla, P.-A. Fouque, and D. Pointcheval. Password-based authenticated key exchange in the three-party setting. In S. Vaudenay, editor, *PKC 2005*, volume 3386 of *LNCS*, pages 65–84. Springer, Heidelberg, Jan. 2005.

6. M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In A. Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 191–208. Springer, Heidelberg, Feb. 2005.
7. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th FOCS*, pages 394–403. IEEE Computer Society Press, Oct. 1997.
8. M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 139–155. Springer, Heidelberg, May 2000.
9. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In V. Ashby, editor, *ACM CCS 93*, pages 62–73. ACM Press, Nov. 1993.
10. M. Bellare and P. Rogaway. Entity authentication and key distribution. In D. R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 232–249. Springer, Heidelberg, Aug. 1994.
11. M. Bellare and P. Rogaway. Provably secure session key distribution: The three party case. In *27th ACM STOC*, pages 57–66. ACM Press, May / June 1995.
12. S. M. Bellovin and M. Merritt. Encrypted key exchange: Password-based protocols secure against dictionary attacks. In *1992 IEEE Symposium on Security and Privacy*, pages 72–84. IEEE Computer Society Press, May 1992.
13. S. M. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In V. Ashby, editor, *ACM CCS 93*, pages 244–250. ACM Press, Nov. 1993.
14. F. Benhamouda and D. Pointcheval. Verifier-based password-authenticated key exchange: New models and constructions. Cryptology ePrint Archive, Report 2013/833, 2013. http://eprint.iacr.org/2013/833.
15. D. Boneh and X. Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of Cryptology*, 21(2):149–177, Apr. 2008.
16. D. Boneh, X. Boyen, and H. Shacham. Short group signatures. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 41–55. Springer, Heidelberg, Aug. 2004.
17. V. Boyko, P. D. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In B. Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 156–171. Springer, Heidelberg, May 2000.
18. E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM CCS 03*, pages 241–250. ACM Press, Oct. 2003.
19. E. Bresson, O. Chevassut, and D. Pointcheval. New security results on encrypted key exchange. In F. Bao, R. Deng, and J. Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 145–158. Springer, Heidelberg, Mar. 2004.
20. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/067, 2000. http://eprint.iacr.org/2000/067.
21. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, Oct. 2001.
22. R. Canetti, S. Halevi, J. Katz, Y. Lindell, and P. D. MacKenzie. Universally composable password-based key exchange. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 404–421. Springer, Heidelberg, May 2005.
23. D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In N. P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 127–145. Springer, Heidelberg, Apr. 2008.
24. D. Catalano, D. Pointcheval, and T. Pornin. IPAKE: Isomorphisms for password-based authenticated key exchange. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 477–493. Springer, Heidelberg, Aug. 2004.
25. R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 45–64. Springer, Heidelberg, Apr. / May 2002.
26. Y. Ding and P. Horster. Undetectable on-line password guessing attacks. *SIGOPS Oper. Syst. Rev.*, 29:77–86, October 1995.
27. R. Gennaro. Faster and shorter password-authenticated key exchange. In R. Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 589–606. Springer, Heidelberg, Mar. 2008.
28. R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In E. Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 524–543. Springer, Heidelberg, May 2003. http://eprint.iacr.org/2003/032.ps.gz.
29. C. Gentry, P. MacKenzie, and Z. Ramzan. A method for making password-based key exchange resilient to server compromise. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 142–159. Springer, Heidelberg, Aug. 2006.
30. C. Gentry, P. D. Mackenzie, and Z. Ramzan. Password authenticated key exchange using hidden smooth subgroups. In V. Atluri, C. Meadows, and A. Juels, editors, *ACM CCS 05*, pages 299–309. ACM Press, Nov. 2005.
31. A. Groce and J. Katz. A new framework for efficient password-based authenticated key exchange. In E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, editors, *ACM CCS 10*, pages 516–525. ACM Press, Oct. 2010.

32. D. Harkins. Simultaneous authentication of equals: A secure, password-based key exchange for mesh networks. In *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications*, SENSORCOMM '08, pages 839–844, Washington, DC, USA, 2008. IEEE Computer Society.
33. D. P. Jablon. Strong password-only authenticated key exchange. *SIGCOMM Comput. Commun. Rev.*, 26(5):5–26, Oct. 1996.
34. J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 475–494. Springer, Heidelberg, May 2001.
35. J. Katz, R. Ostrovsky, and M. Yung. Forward secrecy in password-only key exchange protocols. In S. Cimato, C. Galdi, and G. Persiano, editors, *SCN 02*, volume 2576 of *LNCS*, pages 29–44. Springer, Heidelberg, Sept. 2003.
36. J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM*, 57(1):78–116, 2009.
37. J. Katz and V. Vaikuntanathan. Smooth projective hashing and password-based authenticated key exchange from lattices. In M. Matsui, editor, *ASIACRYPT 2009*, volume 5912 of *LNCS*, pages 636–652. Springer, Heidelberg, Dec. 2009.
38. J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In Y. Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 293–310. Springer, Heidelberg, Mar. 2011.
39. F. Kiefer and M. Manulis. Zero-knowledge password policy checks and verifier-based PAKE. In M. Kutylowski and J. Vaidya, editors, *ESORICS 2014, Part II*, volume 8713 of *LNCS*, pages 295–312. Springer, Heidelberg, Sept. 2014.
40. W. Ladd. SPAKE2, a PAKE, 2015. https://tools.ietf.org/html/draft-irtf-cfrg-spake2-02.
41. S. Lucks. Open key exchange: How to defeat dictionary attacks without encrypting public keys. In *Workshop on Security Protocols*, École Normale Supérieure, 1997.
42. P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. Cryptology ePrint Archive, Report 2001/057, 2001. http://eprint.iacr.org/2001/057.
43. P. D. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 599–613. Springer, Heidelberg, Dec. 2000.
44. C.-P. Schnorr. Efficient identification and signatures for smart cards (abstract) (rump session). In J.-J. Quisquater and J. Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 688–689. Springer, Heidelberg, Apr. 1990.
45. S. Shin and K. Kobara. RFC 6628: Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2, 2012. https://tools.ietf.org/pdf/rfc6628.pdf.
46. S. Shin and K. Kobara. Augmented Password-Authenticated Key Exchange (AugPAKE), 2016. https://tools.ietf.org/html/draft-irtf-cfrg-augpake-06.
47. S. Shin and K. Kobara. Augmented Password-Authenticated Key Exchange for Transport Layer Security (TLS), 2016. http://www.ietf.org/id/draft-shin-tls-augpake-07.txt.
48. S. Shin, K. Kobara, and H. Imai. Security proof of AugPAKE. Cryptology ePrint Archive, Report 2010/334, 2010. http://eprint.iacr.org/2010/334.
49. V. Shoup. Lower bounds for discrete logarithms and related problems. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 256–266. Springer, Heidelberg, May 1997.
50. T. D. Wu. The secure remote password protocol. In *NDSS'98*. The Internet Society, Mar. 1998.