SoK: Password-Authenticated Key Exchange – Theory, Practice, Standardization and Real-World Lessons

Feng Hao, Paul C. van Oorschot

ABSTRACT

Password-authenticated key exchange (PAKE) is a major area of cryptographic protocol research and practice. Many PAKE proposals have emerged in the 30 years following the original 1992 Encrypted Key Exchange (EKE), some accompanied by new theoretical models to support rigorous analysis. To reduce confusion and encourage practical development, major standards bodies including IEEE, ISO/IEC and the IETF have worked towards standardizing PAKE schemes, with mixed results. Challenges have included contrasts between heuristic protocols and schemes with security proofs, and subtleties in the assumptions of such proofs rendering some schemes unsuitable for practice. Despite initial difficulty identifying suitable use cases, the past decade has seen PAKE adoption in numerous large-scale applications such as Wi-Fi, Apple's iCloud, browser synchronization, e-passports, and the Thread network protocol for Internet of Things devices. Given this backdrop, we consolidate three decades of knowledge on PAKE protocols, integrating theory, practice, standardization and real-world experience. We provide a thorough and systematic review of the field, a summary of the state-of-the-art, a taxonomy to categorize existing protocols, and a comparative analysis of protocol performance using representative schemes from each taxonomy category. We also review real-world applications, summarize lessons learned, and highlight open research problems related to PAKE protocols.

1 INTRODUCTION

While user-chosen passwords remain in wide use for authentication [20], many password-based protocols are vulnerable to offline guessing attacks [100]. This motivates the use of *password-authenticated key exchange* (PAKE) protocols, dating back to Bellovin and Merritt's 1992 *Encrypted Key Exchange* (EKE) [15]. EKE allows two parties to establish a high-entropy session key with authentication based on a low-entropy shared password without being subject to offline guessing attacks. Distinct from earlier work [73], EKE *does not* require a trusted third party or any public-key infrastructure (PKI).

A great many PAKE proposals and variants followed, some with new theoretical models to support rigorous analysis. The area also attracted strong industrial interest, including prolonged patent disputes in the 2000s [21]. To reduce confusion and encourage deployment, standards bodies including IEEE, ISO/IEC and IETF have pursued the standardization of PAKE protocols—helping move them from academic study to commercial use. These activities suggest a PAKE research timeline with three main periods: 1992–2008, 2008–2018, and 2018–present. (Fig. 1 in Section 2.1 gives further details.)

Thirty years of PAKE research has left a field rich in theory, practice, standardization—and also real-world lessons, many extending to broader areas of cryptography and security. PAKE research has also led to many interesting questions. For example, a typical PAKE protocol involves only 2 or 3 flows of messages; why is a protocol involving so few messages, so difficult to get right? Of the many provable secure PAKE protocols proposed, why are so few used in practice? How did the standardization of PAKE proceed, and how is it that several *standardized* PAKE protocols are still found to have vulnerabilities? If PAKE protocols appear naturally resistant to phishing attacks, why have they not replaced password authentication in web applications?

Answers to these questions appear not yet to have been pursued in broad, organized manner in one place, or are absent. This motivates our comprehensive review and systematization of PAKE protocols. We review the theory, practice, standardization and real-world applications of PAKE, and draw lessons accordingly. Our contributions include the following.

- We systematically review major PAKE proposals from the past 30 years, including recent updates.
- We categorize PAKE protocols by their main properties and design strategies, and offer a taxonomy.
- Using selected PAKE category representatives, we compare performance of state-of-the-art protocols delivered by the leading design approaches.
- We review real-world applications that use PAKE, and discuss the pros and cons of using these protocols versus non-PAKE alternatives.

Our inclusion of recent work, standardization insights, and lessons learned complements and updates the extensive survey of Boyd et al. [21, Chapter 8]. Our taxonomy systematically highlights critical approach details (e.g., ideal cipher, hash-to-group/hash-to-curve and trusted setup) within classes of PAKE protocols, and challenges in implementing certain protocols including CPace and OPAQUE (both recently selected by IETF for standards). Our comparative analysis of PAKE performance takes into account crucial factors often neglected in previous studies, e.g., group setup and exponent length.

2 LANDSCAPE AND BACKGROUND

Before giving a taxonomy (Section 3), we summarize three periods of PAKE research (see Figure 1), and review EKE.

2.1 Three periods of PAKE research

During the first period (1992–2008), IEEE P1363.2 played a major role in the standardization of PAKE. In response to strong academic and industrial interest in the first half of this period, in 2000, IEEE formed a P1363.2 working group with the mission to review existing PAKE proposals in order to choose a secure subset for standardization. This IEEE project ended in 2008. Protocols in the final 1363.2 specification [51] included SRP [101], SPEKE [52], PAK [79], AMP [68] and variants.

Emails: feng.hao@warwick.ac.uk; paulv@scs.carleton.ca.

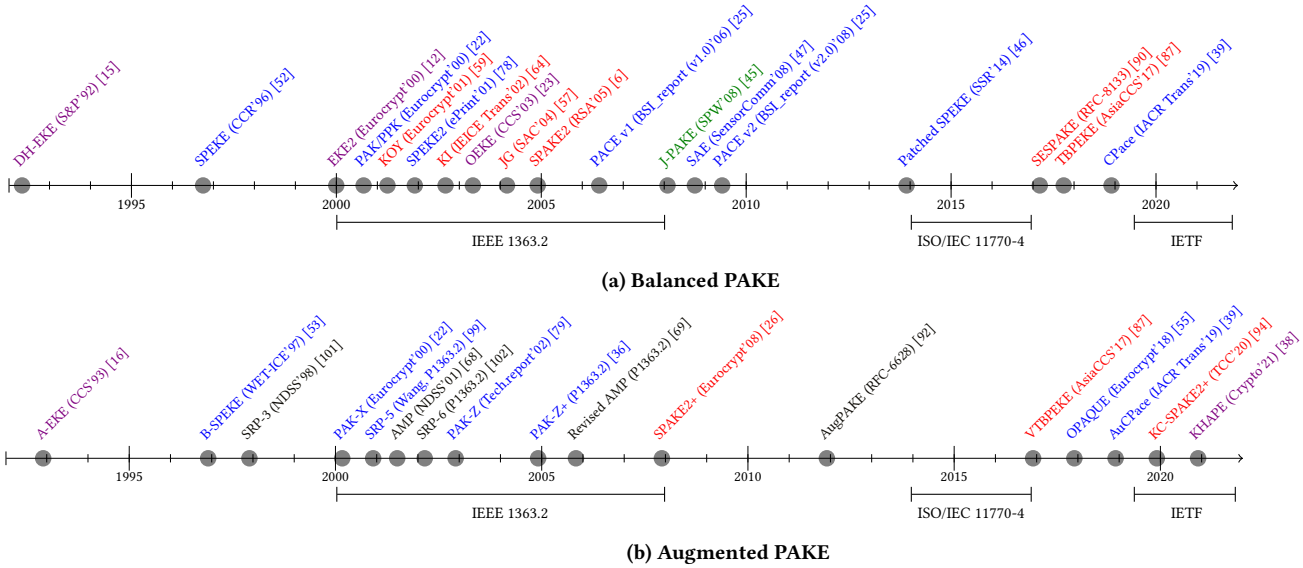

Figure 1: Timeline: PAKE protocols. Color-coding is based on the taxonomy explained in Section 3.

This 8-year standardization process served as a touchstone to test PAKE theory in practice. Unfortunately, nearly all PAKE protocols selected were found to have security issues. Several, including SRP, SPEKE and AMP were found vulnerable to attacks and required revision. In particular, AMP was repeatedly revised [69]. Among the schemes selected, only SRP and SPEKE appeared to be used in practical applications; both continued to be revised after IEEE 1363.2. The latest respective versions are SRP-6a and Patched SPEKE [44]. The IEEE P1363.2 group, initially proposed as a four-year project, was extended annually after 2004 to address various issues in the draft standard until 2008. After 2008, the IEEE specification was no longer maintained, and as of November 2019, the proposed standard in IEEE 1363.2 [51] had been officially withdrawn.

During the second period (2008–2018), ISO/IEC played a main role in standardizing PAKE. The 8-year IEEE P1363.2 effort was less successful than many had hoped—on ending in 2008, it was clear the PAKE problem remained unsolved. This spurred researchers to design new PAKE protocols. ISO/IEC inherited work from IEEE 1363.2 in an ISO/IEC 11770-4 standard, and continued in an active role to maintain this standard. In particular, 11770-4 was revised during 2014 and 2017 to include two new PAKE schemes, J-PAKE [45] and AugPAKE [93], and a patched version of SPEKE [44]

After 2008, the next 10 years saw more deployment of PAKE in real-world applications. SPEKE [52] was adopted in Blackberry Messager (BBM) for secure messaging. SRP-6a [95] was adopted to implement credential recovery in 1password, ProtonMail and Apple's iCloud. J-PAKE [45] was adopted in 2015 by the Thread Group. Section 3.2 also discusses the use of another PAKE protocol in IEEE 802.11: SAE (Dragonfly) [47].

During the third wave (2018–2021, ongoing at the time of writing), the IETF has been a main force driving further development of PAKE—in particular, integrating PAKE into TLS 1.3. The TLS 1.3 pre-shared key (PSK) ciphersuite allows two parties with a pre-shared secret to quickly establish secure communication. This provides a natural use case for PAKE because, in many cases, the pre-shared secret is a password, making the PSK ciphersuite vulnerable to offline dictionary attacks [32]. In June 2019, IETF initiated an open selection process to standardize PAKE protocols. This selection process concluded in March 2020 with two winners declared: CPace [39] and OPAQUE [55], whose details are given in the Appendix.

2.2 Balanced and augmented protocols

PAKE protocols are commonly classified as either *balanced* or *augmented*. A balanced PAKE assumes that the two parties share a secret, which is a password or derived from a password. Typical requirements for a secure balanced PAKE include [45]: 1) resisting offline dictionary attacks; 2) limiting online attacks to one password guess per protocol execution; 3) ensuring *session-key security*; and 4) providing *forward secrecy*.

When a balanced PAKE is used in a client-server setting, if the secret stored on the server is stolen, it can be directly used to impersonate the client. To address this, an augmented PAKE adds a "server compromise resistance" requirement: 5) even if the server is compromised, an offline dictionary attack is needed in order to impersonate the client. This is typically realised by requiring that the client remember a password, while the server stores only a one-way transformation of it. Jarecki et al. [55] recently suggested an extra "pre-computation resistance" requirement, such that: 6) an attacker must perform an offline dictionary attack that *cannot make use of any precomputed table*. Note that these requirements increase the burden on attackers, but do not stop attacks; once a server is compromised, an offline dictionary attack should be expected (one response is to update all passwords [33]).

2.3 EKE: Seeding a new field

EKE was the first PAKE protocol. In Bellovin and Merritt's 1992 paper [15], EKE referred to a suite of variants: RSA-EKE, ElGamal-EKE and DH-EKE. They followed the same basic idea: use a password to encrypt a public key as part of a key transport (RSA and ElGamal) or key agreement (DH) process. As a main difference, in key transport the session key is generated by one party; in key agreement it is jointly generated [83].

However, the use of a password to encrypt a public key is delicate. For the rest of the paper, we use w to denote an (often *weak*) password. Public keys have specific algebraic structures; this provides an oracle enabling offline dictionary attacks. This is especially problematic for an RSA key—for example, the fact that an RSA public exponent e must be odd enables a passive attack to filter out certain passwords. Although this issue may be mitigated by adding random padding to e, Patel [86] presented further attacks. Since the RSA modulus $n = p \times q$ must be sent in the clear [15], the attacker can freely manipulate n to establish an algebraic relationship with the public key e to be encrypted by a password. This relationship allows the attacker to filter password guesses offline. Patel concludes: "The attack [against RSA-EKE] is unavoidable unless the protocol is radically modified."

Other RSA-based PAKE designs have been proposed. One from 1997 was Open Key Exchange (OKE), by Lucks [74]. Rather than using the password to encrypt an RSA key, OKE sends RSA public keys in the open (hence the protocol name). The password is supposedly protected by a combined use of hash and RSA encryption operations. In 2000, MacKenzie et al. [80] presented an active attack on OKE to recover a party's password; they then modified OKE to obtain a new balanced PAKE, SNAPI, with an augmented version, SNAP-X. (They later showed SNAP-X to be insecure, replacing it by SNAP-Z in a journal version [81].)

A major limitation of RSA-based PAKE designs, including SNAPI, is that generating per-session RSA keys in realtime is more costly than Diffie-Hellman (DH) keys. To address this, MacKenzie et al. proposed reusing RSA key pairs across sessions (this gives up *forward secrecy*, above). This is a major reason why nearly all practical PAKE designs are based on discrete log (DH) rather than factoring (RSA) problems.

Among the three EKE variants, DH-EKE is considered the most promising—and is particularly important in the provable security literature, as its security was formally proven in 2000 by Bellare, Rogaway and Pointchevel [12]. The *BRP model* used in this proof has been widely adopted by other researchers. More precisely, this proof was for the EKE2 variant of DH-EKE, whose key exchange flows are the same as DH-EKE, except they include also user identities and the key exchange items in the key derivation function. The protocol works in the whole range of a multiplicative group Z_p^*, using a primitive root modulo p as the generator g (Fig. 3, Appendix).

The provable security of EKE2 was disputed by Zhao, Dong and Wang [103] during the IEEE P1363.2 review process. In 1996, Jaspan [56] discussed a DH-EKE password information leakage problem: if an eavesdropper captures $\mathcal{E}_w(g^x)$—here $\mathcal{E}_w$ denotes a secure block cipher like AES, with a weak password w as the key—use of the correct password will decrypt the ciphertext to a value in $[1, p-1]$ while other guesses yield a value in $[0, 2^{\lceil \log_2(p) \rceil} - 1]$. This discrepancy provides an oracle for a passive attacker to partition passwords. The same leakage problem applies to EKE2. In the 1992 EKE paper [15], the authors already noted this issue in a general context of using a password to encrypt a public key that does not fit precisely the data range for a symmetric cipher; they proposed adding a random pad to a public key using non-modulo arithmetic to fill the data range for encryption, but exact padding details were not specified. Choosing p as close as possible to a power of 2 [56] also reduces (without eliminating) secret leakage.

The formal proof of EKE2 avoids this information leakage by assuming $\mathcal{E}$ is an "ideal cipher" (IC) [12]; then encryption reveals no information about the content even for a low-entropy key. It is assumed that the cipher works like a random function in the encryption, but the decryption function "must take strings of a fixed size and map them to [group] elements" [21]. However, the EKE2 paper does not instantiate such a cipher.

After the formal proof of EKE2, Bellare and Rogaway [14] submitted EKE2 (renamed AuthA) to IEEE P1363.2 as a standardization candidate. This submission was supported in 2003 by Bresson, Chevassut and Pointcheval [23], whose simplified EKE2 variant, One-Encryption Key-Exchange (OEKE), was also supported by a security proof. Whereas EKE2 encrypts both flows in the key exchange process, OEKE only encrypts one (its proof assuming the same IC as EKE2).

As part of the IEEE P1363.2 submission [14], Bellare and Rogaway proposed several ways to instantiate an IC. One was: "$\mathcal{E}_w(x) = x \cdot H(w)$ where H is a random oracle", but they did not instantiate H (which cannot be just replaced by a one-way hash). In 2006, Zhao, Dong and Wang [103] analyzed these IC constructions, and argued that they were inadequate for practical use. As no secure instantiation of an IC was identified, EKE2 was not included into IEEE 1363.2, nor its simpler variant OEKE. However, the AuthA proposal inspired follow-up work using a password to derive random masks to obscure group elements, e.g., in PAK [79] and SPAKE2 [6], which in turn require hash-to-group and a trusted setup, respectively. These lead to different classes of protocols as explained next.

3 TAXONOMY: PAKE DESIGN CLASSES

The main PAKE designs have used passwords three ways: 1) as an encryption key; 2) as an input string to derive a generator; 3) as an integer in modular arithmetic (in the exponent for a multiplicative group, or as a scalar in an additive group over an elliptic curve). The third case includes protocols having different security properties depending on the design approach and formal analysis model (e.g., common reference string, random oracle). Based on these, we identify five classes of PAKE protocols as follows (see Table 1 for a summary of major PAKE schemes, color-coded into these classes; and the Appendix for further taxonomy details and relationships between classes).

C1 **Password used as encryption key**. This class includes using a password as encryption key. Examples: EKE [15], EKE2 [12], OEKE [23], A-EKE [16], KHAPE [38].

C2 **Password-derived generator**. A protocol group generator is derived from a password. Examples: SPEKE [52], SPEKE2 [80], Patched SPEKE [44] and B-SPEKE [53], PAK [79], SAE (Dragonfly) [47], PACE [30], SRP-5 [99], CPace/AuCPace [39], OPAQUE [55].

C3 **Trusted setup**. The protocol relies on a trusted setup, which defines two (or more) generators whose discrete logarithm relationship must be unknown. Examples: KOY [59], KI [64], JG [57], SPAKE2 [6], SESPAKE [89], TBPEKE/VTBPEKE [87], KC-SPAKE2+ [94].

C4 **Secure two-party computation**. Here PAKE is viewed as a two-party secure computation problem on an equality function; use of a non-interactive zero-knowledge proof (ZKP) aims to check that parties follow a specification honestly. Example: J-PAKE [45].

C5 **Password-derived exponent**. In this class, a password is used to derive g^w as a verifier in a type of Diffie-Hellman key exchange. Examples: SRP-3 [101], SRP-6 [102], AMP [68], revised AMP [69], AugPAKE [93].

As Table 1 shows, PAKE security proofs are generally constructed in one or more of three security models:

(1) *Ideal cipher* (IC),
(2) *Common reference string* (CRS), and
(3) *Random oracle* (RO).

IC models a symmetric encryption function that leaks no information about content even when using a low-entropy key [12]. CRS models a function that returns a common reference string trusted by everyone [60]; in PAKE, such a trusted string normally includes two or more generators whose discrete log relationship must be unknown (i.e., no one knows how to represent one generator as the power of another). RO models a function that returns a random string of fixed length (say n bits) for any input string but always the same string for the same input [13], [65], [8, §5.3.1]. So to *prove* security in the RO model means: construct a convincing argument by first replacing some function (e.g., a hash) in the actual protocol by an idealized one that returns a random value as described; prove the idealized protocol has certain properties; then hope the actual protocol delivers those properties when using a concrete hash function, e.g., SHA-3, assumed to behave similarly.

Many PAKE protocols "proven" in the RO model require the idealized function be instantiated by a secure *hash-to-group* function (H2G), or *hash-to-curve* (H2C) in an elliptic curve setting [49], similar to a one-way hash but whose output aims to be a random non-identity element (generator) in a designated prime-order group.

In the following sections, we consider PAKE specs in two settings: with a multiplicative group over a finite field (MODP), and an additive group on an elliptic curve (EC). As selection criteria, we focus on protocols that have attracted academic and industrial interest, and are used in commercial applications.

3.1 Class-1: Password as an encryption key

Section 2.3 discussed balanced techniques in this category. Here we discuss augmented versions. In 1993, Bellovin and Merritt [16] proposed a generic "A" method to transform a balanced EKE scheme to an augmented version denoted A-EKE. Their method is based on a digital signature scheme, with the server storing a password-derived public key for signature verification. This "A" method is generally applicable to transform a balanced PAKE to an augmented version.

In 2021, Gu, Jarecki and Krawczyk [38] proposed an augmented scheme called HKAPE (key-hiding asymmetric PAKE). HKAPE follows the same idea as in EKE by using a password to encrypt a key exchange protocol. However, instead of encrypting an *unauthenticated* (Diffie-Hellman) key exchange protocol, the authors proposed to encrypt an *authenticated key* exchange protocol based on HMQV [82] or 3DH [67, §5.1].[1] The authors proved security of HKAPE in an ideal-cipher-and-RO model under the Gap CDH assumption. To instantiate the ideal cipher, they first require hash-to-curve functions to encode group elements into binary strings in so-called "quasi bijective" mapping (called *quasi* as the strict bijective mapping appears hard). The construction of the hash-to-curve functions depends on an IETF internet draft [49], which remains a draft (unfinished) at the time we write this paper. Second, they require "implementations of ideal ciphers of sufficiently long block length", i.e., between 512 and 1024 bits for the combined input length for elliptic curves of 256 bits. The detailed construction and analysis of such an ideal cipher is "left for future work". Overall, the difficulty to instantiate an ideal cipher has prevented wide use of Class-1 protocols to date.

3.2 Class-2: Password-derived generator

Jablon [52] proposed a Class-2 protocol SPEKE (Simple Password Exponential Key Exchange) in 1992. The key idea is to use a H2G function, denoted f, to map a password to a generator (non-identity element) in a prime-order group. Initially, Jablon defined $f = w^2 \bmod p$, for p a *safe prime* (meaning $(p-1)/2 = q$ is also prime). The square operation maps the password into an element in a subgroup of Z_p^* of prime order q. In 2004, Zhang [104] showed that for this f, an active attacker can test multiple passwords at once by exploiting the exponential equivalence among passwords. During IEEE P1363.2 standardization, to address this issue, a hash operation was added before squaring, revising f to: $f = H(w)^2 \bmod p$. In 2014, Hao and Shahandashti [46] noted that the lack of identity binding in the SPEKE key exchange makes it vulnerable to an *unknown key-share* attack. As a result, SPEKE was revised [44] in ISO/IEC 11770-4:2017 to include user identities in the key derivation function. Figure 4 (Appendix) gives the latest SPEKE specification as in ISO/IEC 11770-4:2017.

The original SPEKE protocol was supported by heuristic security arguments. In 2001, MacKenzie [78] presented a formal analysis (in an unrefereed technical report) of a variant of SPEKE, SPEKE2, based on a new Decision Inverted-Additive Diffie-Hellman (DIDH) assumption in a RO model. SPEKE2 mandates an explicit key confirmation [83] at the expense of requiring more rounds; this process is optional in the original SPEKE paper and specifications per IEEE 1363.2 and ISO/IEC 11770-4. Thus, Mackenzie's security proofs do not directly apply to SPEKE. Furthermore, the proofs apply a weaker security

[1] For 3DH, see also: https://signal.org/blog/simplifying-otr-deniability/

Table 1: PAKE landscape (rows ordered by classes, which are colored-coded per the taxonomy)

Class: Scheme	Year	Security model	Assumptions	Related standards	Properties				
					Requires ideal cipher	Requires hash-to-curve	Requires trusted setup	Peer-reviewed pub exists	Used in commercial app
Balanced PAKE									
C1: DH-EKE [15]	1992	–	–	–	●			✓	
C1: EKE2 [12]	2000	IC	CDH	–	●			✓	
C1: OEKE [23]	2003	IC	CDH	–	●			✓	
C2: SPEKE [52]	1996	–	–	IEEE 1363.2, ISO/IEC 11770-4		●		✓	✓
C2: PAK [22]	2000	RO†	DDH	IEEE 1363.2		●		✓	
C2: PPK [22]	2000	RO†	DDH	IEEE 1363.2		●		✓	
C2: SPEKE2 [78]	2001	RO†	DIDH	–		●			
C2: PACE v1 [17]	2006	IC, RO†	Adaptive gPACE-DH	ISO/IEC 18013-3		●			✓
C2: PACE v2 [30]	2008	IC, RO†	GCBDH	ISO/IEC 18013-3		●			✓
C2: SAE (Dragonfly) [47]	2008	RO†	CDH, DIDH	IEEE 802.11		●		✓	✓
C2: Patched SPEKE [46]	2014	–	–	ISO/IEC 11770-4		●		✓	
C2: CPace [39]	2019	RO†	non-uniform sSDD	–		●		✓	
C3: KOY [59]	2001	CRS	DDH	–			●	✓	
C3: Kobara-Imai [64]	2002	CRS	DDH	–			●	✓	
C3: Jiang-Gong [57]	2004	CRS	DDH	–			●	✓	
C3: SPAKE2 [6]	2005	RO†, [CRS]	GCDH	–			●	✓	
C3: SESPAKE [89]	2015	RO†, [CRS]	CDH	–			●		
C3: TBPEKE [87]	2017	RO†, [CRS]	GSDH	–			●	✓	
C4: J-PAKE [45]	2008	RO	CDH, DDH	ISO/IEC 11770-4				✓	✓
Augmented PAKE									
C1: A-EKE [16]	1993	-	CDH	–	●			✓	
C1: KHAPE [38]	2021	IC, RO†	GCDH	–	●	●		✓	
C2: B-SPEKE [53]	1997	–	–	–		●		✓	
C2: PAK-X [22]	2000	RO†	DDH	–		●		✓	
C2: SRP-5 [99]	2001	–	–	IEEE 1363.2		●			
C2: PAK-Z [79]	2002	RO†	CDH	–		●			
C2: PAK-Z+ [36]	2005	RO†	CDH	IEEE 1363.2		●			
C2: OPAQUE [55]	2018	RO†	One-more DH	–		●		✓	
C2: AuCPace [39]	2019	RO†	non-uniform sSDD	–		●		✓	
C3: VTBPEKE [87]	2017	RO†, [CRS]	GSDH	–			●	✓	
C3: KC-SPAKE2+ [94]	2020	RO†, CRS	CDH	–			●	✓	
C5: SRP-3 [101]	1998	–	–	IETF RFC 2945				✓	
C5: AMP [68]	2001	–	–	–				✓	
C5: SRP-6 [102]	2002	–	–	IEEE 1363.2, ISO/IEC 11770-4					✓
C5: Revised AMP [69]	2005	–	–	IEEE 1363.2, ISO/IEC 11770-4					
C5: AugPAKE [93]	2010	RO	Strong DH	ISO/IEC 11770-4					

RO: *requires a one-way hash function with appropriate properties.* RO†: *requires an H2G function with RO-like properties.*
CRS: *common reference string, explicitly stated in the paper.* [CRS]: *implicit assumption in the paper.* IC: ideal cipher model.

definition whereby an online attacker may test multiple passwords in one go (recall that a stricter definition limits online attacks to one password guess per protocol execution). It also remains unclear whether formal security proofs can be given for SPEKE under more standard Computational Diffie-Hellman (CDH) and Decision Diffie-Hellman (DDH) assumptions. In 2019, Haase and Labrique [39] proposed another SPEKE variant, CPace, by including the session identifiers and the users' identities in the f function. They initially gave a security proof for CPace in a RO model under a standard CDH assumption, but this assumption was disputed during the IETF PAKE selection process [42], and subsequently changed to *strong simultaneous non-uniform CDH* (non-uniform sSDD) [4].

PACE (unrelated to CPace) is a protocol suite proposed in a 2006 report [25] (with corrections and revisions 2006–2016) for machine readable travel documents such as e-passports, by Germany's Federal Office for Information Security (BSI). As a core component, a set of functions map passwords to group elements. The original protocols in the BSI report saw limited public scrutiny. In 2009, Bender, Fischlin and Kügler [17] analyzed the initial version (PACE v1) in a random-oracle-and-ideal-cipher model under a General Password-Based Chosen-Element DH (gPACE-DH) Problem. In 2012, Coron et al. [30] analyzed an updated version (PACE v2) in a RO-and-IC model[2] under a Gap Chosen-Base Diffie-Hellman (GCBDH) assumption. The main change in v2 is a new Integrated Mapping (IM) method that hashes a password to points on certain elliptic curves more efficiently (with curves restricted to $p = 2 \bmod 3$; p is the characteristic of the prime field). The security analysis is limited to a *restricted model*: the adversary is not allowed to interact with a reader, and can only interact with a chip when it is not already interacting with a reader. In all versions of PACE [17, 30], user identities are omitted in the key exchange flow, making the protocol potentially vulnerable to an unknown key-share attack as reported on SPEKE [46]. However,

[2]The RO-and-IC model used to analyze PACE v1 and v2 does not require an ideal cipher (e.g., unlike EKE2), thus allowing use of AES as symmetric cipher.

such an attack is beyond the scope of the *restricted model*, since PACE is designed for an RFID reader and an RFID chip with fixed roles of "reader" and "chip" hardcoded in their respective software implementations. Applying PACE as a general PAKE protocol for Internet use will require modifying the protocol; at least user identities need to be included.

SAE (also called Dragonfly) is among the PAKE protocols that have been used in practical applications, most notably in Wi-Fi Protected Access 3 (WPA3), a commercial profiling of the IEEE 802.11 standard. The protocol was first proposed in 2008 by Harkins [47], unaccompanied by security proofs. In 2014, a small-subgroup attack was reported by Clarke and Hao [28] and subsequently fixed by a revision. To support SAE's standardization in the IETF, in 2015 Lancrenon et al. [71] offered an SAE security proof in a RO model under CDH and DIDH assumptions, adapting the SPEKE2 proof by MacKenzie [80]; the proof does not, however, take into account side-channel attacks. Shortly after being made mandatory in 2018 in WPA3 (personal mode), SAE was shown vulnerable to side-channel timing attacks [97]. The problem arises in implementing the f function (we give details shortly).

PAK is a suite of variants proposed in 2003 by Boyko, MacKenzie and Patel [22] (see also [21]). The variants include PAK, PPK and PAK-X, all relying on the same f function used in SPEKE; to our knowledge [76], none have been used in practical applications. A formal security proof of properties of these variants was given under a decision Diffie-Hellman (DDH) assumption in the RO model. PAK and PPK are both balanced schemes, with PAK mandating explicit key confirmation while PPK makes it optional; PAK-X is an augmented version.

We now briefly discuss augmented Class-2 PAKE schemes. In 1997, Jablon [53] proposed a generic "B" method to extend a balanced SPEKE scheme to an augmented version denoted B-SPEKE. This method is based on a Diffie-Hellman scheme with the server storing a password-derived Diffie-Hellman public key. The same "B" method is generally applicable to transform a balanced PAKE to an augmented version. Other augmented schemes include PAK-X [77], SRP-5 [99], OPAQUE [55] and AuCPace [39]. During the IEEE P1363.2 review process, PAK-X was found vulnerable and replaced by PAK-Z [79], which was found vulnerable and replaced by PAK-Z+ [36].

A challenge affecting all Class-2 protocols is how to instantiate the f function in different group settings. Recall in an EC setting, f is called a *hash-to-curve* (H2C) function. Jablon originally (1996) defined f only for a MODP setting with safe-prime modulus. IEEE 1363.2 extended it to work with DSA (Digital Signature Algorithm) [83] groups and elliptic curves as discussed below.

In a general MODP setting, $p = k \cdot q + 1$ is the modulus, where p, q are large primes and k a co-factor. For DSA groups, q is much shorter than p, for efficiency [83, §11.5]. IEEE 1363.2 [51] defines the mapping function: $f(w) = H(w)^k \bmod p$. Here, there is a theoretical possibility that $H(w)$ falls into a small subgroup of Z_p^*. In that case, f in IEEE 1363.2 simply returns "invalid". The specification does not mandate any checking on the output nor provide any exception handling for the "invalid" case, but notes that when "suitably large values" of the group parameters are chosen, the probability of this happening by chance is negligible (on the order of $1/q$). For completeness, we note that when an "invalid" output is returned, the user may have to change the password. Dragonfly tries to address this issue by including a counter in a loop so as to guarantee that a valid output is always returned (see "Hunting and Pecking with MODP Groups" in [48]). As a consequence, f is no longer constant-time. Nonetheless, the practical difference between the methods in IEEE and Dragonfly seems negligible given that the probability of an "invalid" output occurring by chance is negligible. (However, it is unclear if an attack action, e.g., fault injection [96], could significantly increase the probability of the "invalid" case.)

In the EC setting, IEEE 1363.2 [51] defines the H2C function for an elliptic curve $y^2 = x^3 + a \cdot x + b$ in three steps: 1) first apply a one-way hash to the password to obtain an x-coordinate value; 2) use a loop with a counter to iteratively map the x value to a point on the curve; 3) multiply the mapped point with a co-factor. There is a theoretical possibility that the mapped point in 2) might fall into a small subgroup, and then the computed result in step 3) will be an identity point. In this case, the function simply returns "invalid". Similar to the MODP setting, the IEEE specification does not mandate checking the output nor specify any exception handling, but notes that the probability for this happening at random is negligible. The H2C function in IEEE 1363.2 was adopted to implement an EC version of Dragonfly in 2018 in WPA3, but found vulnerable in 2020 to timing side-channel attacks [7, 97]. The main problem is that the mapping function in step 2) is not constant-time. We note that Blackberry Messenger (BBM) [18] uses an EC version of SPEKE based on the same H2C function in IEEE 1363.2 [51], and hence is potentially vulnerable to the same attack. IETF has been trying to address this problem by defining constant-time mapping [50, 91] in step 2) for selected curves, but this work is still on-going at the internet draft stage [49] as we write this paper.

3.3 Class-3: Trusted Setup (TS)

The first Class-3 protocol was KOY by Katz, Ostrovsky and Yung [59] in 2001. Its underlying motivation was to design a provably secure PAKE *without* requiring a RO model, following work to this end in 2000 by Goldreich and Lindell [37]. KOY has 3 rounds for unilateral authentication (4 rounds for mutual authentication). As a central idea, a trusted setup (TS) is assumed, including here a set of 5 generators, with no one knowing the discrete log of any of these generators with respect to any of the others (and the party choosing them must be trusted on this assumption). While the original paper asserts a formal security proof for the KOY protocol under a DDH assumption in a "standard" model, due to dependence on TS, a more precise statement is that the proof is in a *common reference string* (CRS) model, as acknowledged later [60]. A general framework for PAKE in a CRS model was given by Gennaro et al. in 2003 [35] and Katz et al. [61] in 2011; in 2014, Abdalla reviewed the CRS model for PAKE [1].

After KOY, other TS-based PAKE schemes included Kobara-Imai [64], Jiang-Gong [57], SPAKE2 [6], SESPAKE [89] and TBPEKE [87]. Among these, SPAKE2, from 2005 by Abdalla

and Pointcheval [6], is regarded as the most efficient. SPAKE2 (Fig. 7, Appendix) uses three generators and assumes the discrete log relation between any two of them is unknown. The authors gave a RO-model proof, initially under a CDH assumption. This proof does not deliver forward secrecy; for that, the assumption is changed to Gap Computational Diffie-Hellman (GCDH) [2]. They proposed to obtain a TS as "the output of a [H2G] random oracle" [2], a proof technique subsequently adopted by SESPAKE [90], TBPEKE/VTBPEKE [87] and KC-SPAKE2+ [94]. Given the reliance on TS, the proof is implicitly in a CRS model, as noted by Becerra et al. [11].

Class-3 augmented PAKE schemes include VTBPEKE and KC-SPAKE2+. VTBPEKE was proposed in 2017 by Pointcheval and Wang [87]. They gave a RO-model proof under a gap simultaneous Diffie-Hellman (GSDH) assumption. Again due to reliance on a TS, the proof is implicitly in a CRS model. KC-SPAKE2+ was proposed in 2020 by Shoup [94] as a variant of SPAKE2+ [26] with mandatory key confirmation. Shoup analyzed the security of the protocol in the RO and CRS models under a CDH assumption.

A critical issue for Class-3 protocols is how the TS is instantiated. One frequently suggested way [35, 57, 59] is to employ a trusted third party (TTP) to choose the parameters used. Of course, a TTP may also be viewed as "a third party who can break your security policy" [8]; no universally trusted third party exists. Another way is using a secure hash function to generate fixed generators for standardized use (e.g., see the SPAKE2 internet draft [70]). Note that reliance on the discrete log relationship of fixed values remaining unknown is a significant risk for a system to be used by very large user bases over many years. In non-TS-based PAKE systems, solving a single discrete log problem instance breaks a single session; in TS-based systems, it breaks security for all sessions and all users (typically without their knowledge).

3.4 Class-4: Secure two-party computation

The only Class-4 protocol to date is J-PAKE, proposed in 2008 by Hao and Ryan [45]. The core idea is to treat PAKE as a secure two-party computation problem. The aim is that each party learns only the 1-bit output of a function that tests the equality of two passwords. J-PAKE extends the traditional two-party computation problem, by deriving a session key when the passwords are equal. Its design modifies the 2006 AV-net multi-party computation (MPC) protocol of Hao and Zielínski [43], by which multiple parties compute a boolean-OR function, to instead compute an equality function. J-PAKE requires neither H2C nor a TS, simplifying implementation. In 2016, Lancrenon et al. [72] proposed two variants which use less computation, but respectively rely on H2C and TS.

J-PAKE (Fig. 8, Appendix) relies on a zero-knowledge proof (ZKP) technique, namely Schnorr's non-interactive ZKP [83, §10.4.4], to ensure that a sender knows the private key x when sending g^x. ZKP had previously seen little use in PAKE as first, ZKP can be computationally expensive (J-PAKE uses 3 ZKPs in each direction); and second, ZKP was viewed as incompatible with the mainstream BRP model due to the difficulty to use rewinding arguments to prove extraction of knowledge in simulation [3]. While the original paper's security proof for J-PAKE (in a RO model with CDH and DDH assumptions) is considered informal by some (it is not constructed in the formal BRP model), in 2015 Abdalla et al. [3] defined a modified BRP model with algebraic adversaries (to be compatible with ZKP), used this for a formal proof of J-PAKE under a Decision Square DH assumption, and then reduced this assumption to CDH and DDH by using a RO model.

3.5 Class-5: Password-derived exponent

Class-5 schemes are all augmented. Here g^w is stored at the server for use in verification, in a DH-like key exchange with a client. The first Class-5 protocol was SRP-3, in 1998 from Wu [101] ("-3" as two earlier versions were abandoned). After submission to IEEE P1363.2, it was noted that an active attacker could test two passwords at once; SRP-3 was replaced by SRP-6. After the P1363.2 group concluded in 2008, security concerns led to a further update, SRP-6a (Fig. 9, Appendix).

Other Class-5 protocols follow a design similar to SRP. AMP, proposed in 2001 by Kwon [68], claimed to be the "most efficient" augmented PAKE. After AMP was submitted to IEEE P1363.2, it was found vulnerable to the two-guess attack reported on SRP-3, and revised to AMP+; efficiency concerns led to a further revision in 2002. While AMP+ was being standardized in IEEE P1363.2, it was found to lack the claimed server compromise resistance, again modified and replaced by the "the unified variant AMP2" [69]. AugPAKE was initially proposed in 2010 by Shin and Kobara in an RFC [92], but the validity of its security proof is challenged by Jarecki et al. [55].

4 COMPARING EFFICIENCY

We now select representative protocols from each class for a comparative performance analysis. As Table 1 notes, PAKE protocols use a wide variety of security models and assumptions in their security proofs—which precludes precise security comparisons. Thus we focus here on computational efficiency for parameter choices, e.g., modulus bitlengths and exponent sizes, that provide comparable (asserted) security. Our summary (in Table 3) provides a view of relative performance, while separately noting further costs due to required H2C functions or trusted setup (TS), the latter being a non-computational cost.

We first list the class representatives. Our selection criteria included academic interest and industrial use, plus simplicity, maturity and efficiency within classes. We do not distinguish balanced and augmented schemes as this does not affect our comparative analysis between different design strategies.

- C1: EKE2. It is a classic scheme in its class. Variants cost roughly the same computationally.
- C2: Patched SPEKE. It is the most efficient in its class (with the same efficiency as SPEKE, while addressing known attacks). We also include CPace [39] and OPAQUE [55], based on IETF having selected them for standardization.
- C3: SPAKE2. It is the most efficient in this class.
- C4: J-PAKE. It is the sole class member.
- C5: SRP-6a. It is the only class member widely used in practice, to our knowledge.

4.1 Long and short exponents

For Diffie-Hellman exponentiation over the range Z_p^*, exponents are mod $p-1$, so full-sized (*long*) exponents have bitlength $\log_2(p)$. Shorter exponents reduce computational cost, as modular exponentiation cost is linear in exponent bitlength [83, §14.6]. However, a too-short exponent compromises security. For example, Pollard's rho algorithm (which is parallelizable) finds discrete logs in running time square-root of the size of the search space, while a modification of Pollard's lambda method allows recovery of a number of exponent bits, depending on how $p-1$ factors [85]; the latter motivates using safe primes, which help address small-subgroup attacks [84, §4.8], as do prime-order subgroups as used in DSA.

At first glance, EKE and SPEKE require only two exponentiations, which seems impossibly few, as even *unauthenticated* DH requires that many (plus one more, if one charges for public-key validation). However, simply counting the number of exponentiations is misleading. Both EKE and SPEKE specify a safe prime as modulus (in their original specs). Given a 2048-bit safe prime p, the exponents in EKE and SPEKE are 2048 and 2047 bits resp. (Fig. 3 and 4, Appendix). In contrast, using DSA groups, exponents of only 224 bits are commonly used. As a result, for a 2048-bit p, one full-length SPEKE exponentiation costs about $2047/224 \approx 9$ times as much as for a (112-bit security) DSA group; and for a 3072-bit p, $3071/256 \approx 12$ times as much as for a (128-bit security) DSA group. Counting this way, for a given modulus, EKE and SPEKE (with long exponents) are far less efficient than a regular (DSA-group) DH protocol.

Direct use of short exponents in SPEKE was initially mentioned [52], e.g., for a 3072-bit safe prime p, choose a random 256-bit secret exponent; but it was later recommended [54] to use them while ensuring computations in a prime-order (DSA) subgroup, consistent with other recommendations [85]. Following this path, IEEE 1363.2 [51] generalized its SPEKE specification to allow short exponents as in DSA groups, resulting in changing f (as noted in §3.2) to $f = H(w)^k \mod p$; again $p = k \cdot q + 1$ and k is a co-factor. Compared with a safe-prime modulus, the probability for getting an "invalid" output *by chance* in a DSA group (same modulus size) increases, but remains negligible (without considering attack actions like fault injection [96]). However, in a DSA group, it now requires one exponentiation to validate a received public key (exponential). With a safe prime modulus, validation involves checking only the exponential is not 1 or $p-1$, ruling out small subgroups ([52]; [84, §4.8]).

Table 2 summarizes costs for SPEKE in different groups. With short exponents reducing exponentiation costs, the f function now dominates the overall cost. Hence, using the same DSA group, SPEKE remains far more costly than a DH.

In an EC setting, SPEKE requires an H2C function to map a password to a random generator. The H2C defined in IEEE 1363.2 (later adopted by ISO/IEC 11770-4 and WPA3), based on a trial-and-increment method, leaks side-channel information about the password [97]. This can be addressed by Icart's method [50] to map a string to a point on a curve in constant-time, suitable for certain curves where $p \equiv 2 \bmod 3$. Another method due to Shallue, Woestijne and Ulas [24, 91], suits curves where $p \equiv 3 \bmod 4$. These methods require that one define custom H2C solutions for different curves, and the mappings may not work for (future) new curves. Furthermore, existing "constant-time" methods [24, 50, 91] map an input string to a point on an elliptic curve, including small-subgroup points [42]. It is possible to check the output iteratively to ensure it always returns a valid generator (required by Class-2 schemes, as discussed) but doing so forfeits the (desired) constant-time property. An IETF internet draft [49] aims to address these issues by defining custom H2C functions for (ten) selected curves; until this work is finalized, the actual H2C cost remains unknown. For this reason, Table 3 uses "H2C" to denote this function's (unknown) cost.

	2048-bit modulus		3072-bit modulus	
	Safe prime	DSA	Safe prime	DSA
Hash to group	–	1 (×8)	–	1 (×11)
Generate g^x	1 (×9)	1	1 (×12)	1
Validate g^y	–	1	–	1
Compute g^{xy}	1 (×9)	1	1 (×12)	1
Total	**2 (×9)**	**11**	**2 (×12)**	**14**

Table 2: SPEKE costs in different MODP groups. Under "2048-bit/Safe prime", long exponents are assumed for prudence; 1 (×9) denotes that the cost of one exponentiation (with 2047-bit exponent) is about the same as 9 typical (224-bit exponent) DSA-group exponentiations. The hash-to-group function requires an exponentiation with a 1823-bit exponent (co-factor k), which similarly has cost equal to 8 exponentiations with 224-bit exponent. Similar scaling applies for a 3072-bit modulus.

4.2 Performance comparison

Table 3 compares system performance for the selected PAKE schemes (respective Appendix sections give cost breakdown details). The aspects noted are: type (balanced/augmented), class (C1-C5), number of rounds, number of flows, key confirmation (implicit/explicit), number of modular exponentiations in MODP groups, number of scalar multiplications in EC groups, and whether the scheme requires a trusted setup. A *round* (different from a *flow*) is a step which all participants can complete without depending on each other [21, §1.5.12]. For example, the original EKE protocol is 1-round, but it requires 2 flows of message exchange. EKE2 [12] breaks the symmetry of EKE by including the ordered identities in the key derivation function. As a result, it becomes 2-round, still with 2 flows of message exchange; the two users cannot send the data at the same time in one round because of the ordered identities. With the exception of SRP-6a, selected PAKE protocols generally support implicit key confirmation; explicit key confirmation can be realized by adding one more round or flow without significantly changing the computational costs in the table. We use a MODP setting with 3072-bit modulus; a 2048-bit modulus would not change the main results shown. For public key validation, we note that in a DSA-like group this requires one exponentiation, but for a safe-prime modulus or in an EC setting, the cost is negligible.

Class: Scheme	Year	Type	Rnd	Flow	KC	MODP (safe prime)	MODP (DSA)	EC	Other
C1: EKE2 [12]	2000	Bal	2	2	Imp	2 (×12)	–	–	
C2: Patched SPEKE [46]	2014	Bal	1	2	Imp	2 (×12)	1 (×11) + 3	2+H2C	
C2: CPace [39]	2019	Bal	3	3	Imp	2 (×12)	1 (×11) + 3	2+H2C	
C2: OPAQUE [55]	2018	Aug	2	2	Imp	C: 4.5 (×12), S: 3.5 (×12)	C: 1 (×11) + 6.5, S: 5.5	C: 4.5+H2C, S: 3.5	
C3: SPAKE2 [6]	2005	Bal	2	2	Imp	–	3	2	TS
C4: J-PAKE [45]	2008	Bal	2	3	Imp	–	14	11	
C5: SRP-6a [95]	2002	Aug	4	4	Exp	C: 2 (×12) + 1, S: 2 (×12) + 1	–	–	

Table 3: Comparison of selected PAKE protocols with focus on computational costs (protocol specifications and detailed cost breakdowns can be found in the Appendix). Computational cost columns give the number of exponentiations in the MODP setting (assuming a 3072-bit modulus for concreteness, and to apply scaling to account for long exponents as explained in Table 2) or the number of multiplications in the EC setting (this is independent of the EC bitsize). TS implies an extra non-computational cost (trust or risk). C: Client side cost. S: Server side cost.

In Table 3, in OPAQUE we use HMQV [82] as originally proposed [55] as the "most efficient" way to instantiate the key exchange scheme. (After OPAQUE's selection by IETF, the designers proposed replacing HMQV with a different 3DH protocol; however, the proposed change remains unfinalized in an IETF internet draft [66] at the time we write this paper.) The OPAQUE paper does not explicitly specify whether a party should validate the received public keys in HMQV. Based on Menezes [82] and Hao [40], we consider public-key validation essential in HMQV to prevent known attacks, with the cost counted accordingly in the table (i.e., one exponentiation in DSA and negligible cost in safe prime and EC). For both CPace and OPAQUE, the original papers assume an H2C function as the main building block, leaving the protocols undefined for the MODP setting. Neither paper instantiates H2C; both assume it incurs negligible cost. To give a complete picture, Table 3 records the cost for CPace and OPAQUE in a MODP setting based on using the same H2G function as other Class-2 protocols. In an EC setting, since H2C has not been instantiated, the table uses "H2C" to denote this non-zero cost.

On round efficiency, EKE2, Patched SPEKE, SPAKE2 and OPAQUE have the fewest flows (i.e., 2). Among these, Patched SPEKE has the theoretical advantage that its 2 flows can be completed in one round. J-PAKE and CPace require 3 flows; SRP-6a requires 4 (note that it includes explicit key confirmation by default; other protocols treat it as optional). To add explicit key confirmation requires one more flow. The round efficiency of CPace is based on its original paper [39]. After CPace was selected by IETF, the designers proposed to modify the protocol by removing one session ID and defining user identities as *optional* input to H2C in order to reduce rounds; but this proposed change remains unfinalized in an internet draft [5] as we write this paper.

A computational efficiency comparison requires considering TS and H2C as factors. First, consider TS. From Table 3, SPAKE2 has the lowest computational cost in both MODP and EC settings, but its security critically relies on the secure instantiation of a TS; a compromised TS compromises the security of all sessions–similar to breaking TS in Dual EC [27].

Next, consider H2C in the EC setting. Since H2C has not been instantiated in many cases, a direct comparison of Class-2 (Patched SPEKE, CPace, OPAQUE) to other protocols is precluded. However in the MODP case, with the f function fully defined in IEEE 1363.2 [51], we can calculate the cost of Class-2 protocols using either a safe-prime modulus or DSA group; we use the lower cost in our comparisons. As seen from Table 3, when CPace and OPAQUE are extended in the MODP setting using known H2G mapping techniques, the cost under *DSA* is lower than under *safe prime* for the same modulus size; in a DSA group, the overall cost is dominated by H2G. In a MODP setting, the computational cost is roughly equal for Patched SPEKE, J-PAKE, CPace and OPAQUE; SRP-6a is higher.

5 REAL WORLD USE CASES

Since EKE's conception, debate has continued on the best target applications for PAKE. Many arguments have suggested replacing password authentication in web applications, including to address phishing, but to little effect—apparently due to major deployment barriers and the inertia of incumbent web authentication password protocols (see Manulis et al. [75]). Strong arguments have likewise been made for integrating PAKE into TLS, prompting a 2019 IETF PAKE selection process; an earlier 2007 TLS-SRP effort (based on SRP-6) supported by RFC 5054 failed to result in wide adoption [31]. Participants on CFRG e-mail[3] offered the conclusion: "PAKE in SSL has always been a solution in search of a problem". PAKE has also been proposed for use in end-to-end secure email [98]. This begs the question: what are the real use cases for PAKE?

To pursue this, we review major instances of PAKE adoption in real-world commercial systems, and compare with non-PAKE alternatives, in three selected areas: credential recovery, device pairing, and end-to-end (E2E) secure channels (see Table 4). We distinguish a method as *preventive* or *detective* with the latter relying on user vigilance to detect attacks.

We note two basic requirements for a practical PAKE application: 1) a trustworthy password-entry interface, and 2) a trustworthy out-of-band channel to share passwords between two parties (e.g., reading a password from one device and entering it to another). In existing PAKE applications, the password-entry interface is usually integrated into the underlying operating system, or the app. By contrast, a web page is not a trustworthy interface since it can be easily manipulated (e.g., by JavaScript) [8, §3.4]. This, along with other web deployment hurdles as noted above ([31], [75, §9]), has prevented PAKE from being used in many web applications. However,

[3]Email of 16/02/2016, https://irtf.org/cfrg

Use case	Methods	Example	Type	Property		
				No dependence on a PKI	Prevents external attacks	Prevents insider attacks
Credential recovery	SRP-6a	iCloud	Preventive	◐	●	○
	PKI+pwd	Google sync	Preventive	○	●	○
Device pairing	PACE	E-passport	Preventive	●	●	–
	Dragonfly	WPA3	Preventive	●	◐	–
	Passkey	Bluetooth	Preventive	●	○	–
	Num com†	Bluetooth	Detective	●	●	–
E2E secure channel	J-PAKE	Thread	Preventive	●	●	●
	EC-SPEKE	BBM	Preventive	●	◐	●
	PKI+pwd	TLS 1.3	Preventive	○	●	●
	Signal†	WhatsApp	Detective	●	●	●
	ZRTP†	VoIP	Detective	●	●	●

● (provides property); ◐ (partially provides); ○ (does not provide); – (not applicable). †Requires manual security-check by end-users.

Table 4: Use case properties: PAKE, non-PAKE

the adoption of PAKE in e-passport, Wi-Fi and IoT suggests the utility and demand for PAKE beyond web applications.

5.1 Credential recovery

SRP-6a has been used in iCloud (and similarly in 1Password and ProtonMail) to recover user credentials (called "escrow recovery" in iCloud documentation). To recover a keychain stored at the iCloud server, users log onto their iCloud account using username and password plus an SMS code as a second factor for authentication, then enter another (6-digit) iCloud security code serving as a low-entropy w for establishing a secure channel with the iCloud server based on SRP-6a (running on top of TLS) to retrieve the keychain. The SRP-6a registration phase is done over a TLS channel, but the (PAKE) key exchange does not rely on TLS.

Google Sync provides an alternative non-PAKE solution based on a PKI and passwords (Mozilla Sync 1.5 works similarly). In Google Sync, the user's profile (containing *all cached passwords* in Chrome) is stored at the Google server. To retrieve the sync data, users simply log onto their Google account using a Chrome browser. The user can optionally provide a further password to encrypt the sync profile stored at the server.

As Table 4 shows, the partial dependence on TLS or a PKI appears to give SRP-6a in iCloud an advantage over Google sync under a specific scenario. In iCloud, TLS is only required during the registration phase of SRP-6a, not during key exchange. Therefore, iCloud may provide more protection than Google sync during credential recovery *if TLS or its underlying PKI is broken*. However, as a major limitation, neither iCloud nor Google Sync protects against insider attacks. If the service provider hands over password-encrypted user credentials to a government agency, the credentials will be vulnerable to offline dictionary attacks.

5.2 Device pairing

PACE is used by third generation e-passports. An e-passport reader scans the Machine Readable Zone (MRZ) of a passport page, with MRZ data used as a shared secret for PAKE mutual authentication with the e-passport's embedded RFID chip [30]. WPA3 uses another PAKE protocol, Dragonfly, to establish a secure channel between a Wi-Fi access point and client. However, as noted in Section 3.2, the trial-and-increment H2C function used (based on IEEE 1363.2) is vulnerable to side-channel attacks [97]. The non-PAKE alternatives include Passkey Entry (PE) and Numeric Comparison (NC) in the latest Bluetooth pairing specification, 5.3 [19]. PE requires a user to read a 6-digit number from one device and enter it to another device, but has been found vulnerable to impersonation attacks [29]. NC requires a user to manually compare a 6-digit display on both devices after performing an ECDH protocol in order to confirm authentication, hence working as a *detective* measure. Here, a PKI-based TLS is unsuitable for these pairing applications as there is no pre-existing PKI.

5.3 End-to-end (E2E) secure channel

Examples of PAKE applications here include use of J-PAKE in Thread (and similarly in Palemoon and Smoke Chat); and EC-based SPEKE (EC-SPEKE) in Blackberry Messager (BBM) Protect to establish end-to-end secure channels between parties over the Internet. EC-SPEKE uses the same IEEE 1363.2 trial-and-increment H2C function as Dragonfly in WPA3, which as noted in Section 3.2, has side-channel issues.

In this category, we note three non-PAKE alternatives to establish E2E secure channels. One is to rely entirely on TLS—but in many applications (including IoT), an underlying PKI is not in place or trusted. Another is to use *detective* methods as in Signal [88] or ZRTP [105]. Signal requires users to manually compare (60-digit) fingerprints of other users' public keys before running an authenticated (X3DH) key agreement. However, studies have shown that users cannot be relied on to carry out such checks [58, 88]. Similarly, ZRTP first executes a DH protocol between two phone users and then requires them to manually compare an authentication string (typically 6 digits) derived from the session key.

6 LESSONS AND OPEN PROBLEMS

Here we extract lessons (Ln) from the theory, practice, standardization and real-world applications of PAKE protocols.

L1 (Complete specifications). *A PAKE protocol should be completely specified to enable analysis and open implementations.* PAKE schemes face adoption barriers if their published security proofs assume constructs for which any implementation details are unavailable. Examples include assumptions of ideal ciphers, and H2G functions missing for some group settings. Omitted details also hide subtle issues that must be addressed in full analysis, and hamper implementation. In future standardization of PAKE (and other security techniques), we recommend open-source full prototypes accompany submitted candidate protocols as reference implementations, to ensure all details are defined.

L2 (Realistic assumptions). *Security proofs should specify both the underlying model and realistic assumptions; unrealistic or questionable assumptions erode the value of proofs and impede adoption.* As an example, the motivation for several provable

PAKE schemes was to remove the RO assumption, to avoid criticism that equating an RO to a hash function is *heuristic in nature* [13]. However, the alternative of employing a trusted third party (TTP) [59] in a CRS model increases implementation challenges. To address this, researchers (re)introduced RO to avoid a TTP defining a CRS [6] (see Table 1). But then, as §3.3 notes, finding one discrete log relationship between two hardcoded generators forever breaks all sessions—a concern limiting the adoption of Class-3 protocols. From this, we learn that some new models and assumptions remove old issues, but introduce new ones.

L3 (Revising standards). *PAKE standards, like many other security standards, must be regularly revisited to address new attacks.* Designing public key protocols is notoriously difficult [9]. Several standardized PAKE schemes with designs based on heuristic arguments (such as SRP, AMP and SPEKE), required repeated revisions as new attacks emerged. Many PAKE schemes with security proofs were accompanied by incomplete protocol specifications (e.g., lacking details on instantiation of IC, H2C and TS, per L1 and L2 above). While IEEE P1363.2, initially set out as a 4-year standardization project, was extended to 8 years, the final spec was eventually withdrawn in 2019, after flaws continued to be found in the 2008 specification [51].

L4 (Emergent use cases). *Use cases for new protocols emerge or evolve with deployment environments.* A motivation for the original EKE protocol was E2E encrypted telephone calls [15]. This was difficult to implement in 1992 (due to the transcoding of analogue voice data across heterogeneous telecom networks [10]) but appears to be less problematic in today's SIP-based phone network [63]. Back in 1992, Wi-Fi, e-passports and the notion of IoT had yet to emerge, or were ahead of technology evolution; today, they are examples of the large-scale, real-world use of PAKE (cf. Table 4).

L5 (Trade-offs). *PAKE protocols are rarely directly comparable, with different trade-offs between security models, performance and other functionality.* It is impossible to declare any one PAKE protocol "best" for all applications, due to the variance in the importance of given properties in different applications and environments, e.g., H2C costs vs. a TS (non-computational complexity). Thus, we believe a systematization as given herein offers valuable insights regarding tradeoffs.

Among open problems, we highlight three. First, several PAKE protocols, including IETF-favored CPace and OPAQUE, critically depend on an H2C function to map a password to a base generator on an elliptic curve. However, finding a secure and efficient H2C that guarantees "valid" output and works with general elliptic curves remains an open problem (§3.2). Second, existing augmented PAKE schemes provide limited protection of stored passwords in the case of server compromise. OPAQUE's *pre-computation resistance* aims to ensure offline dictionary attacks cannot employ pre-computation tables (§2.2), but this is only partial protection. An ideal augmented PAKE scheme might fully protect stored passwords even in case of a server compromise (e.g., via hardware security modules, but doing so adds cost). Third, few PAKE protocols resist attacks based on quantum computers [8, §5.7]; none of the PAKE schemes selected by IEEE, ISO/IEC or IETF do so. The design and standardization of quantum-secure PAKE protocols remains an open challenge (e.g., see Gao et al. [34]).

As a final remark, PAKE presents an interesting case study to reflect on how the theory of provable security has been developed, refined and tested in this field. Heuristic designs, common in early PAKE research, have been found to be unreliable, falling into repeated break-and-patch cycles. Since 2000, provable security has been proposed as the path to escape this cycle. Yet, of many now-available PAKE protocols accompanied by proofs asserting security properties, few have been fielded. Most designs swing between difficult choices, relying on 1) an ideal cipher, 2) a hash-to-group function, or 3) a trusted setup, but none of these has turned out to be straightforward to implement. As knowledge about PAKE protocols continues to evolve, we have provided a snapshot-in-time picture of where we are after 30 years of research. We hope that the insights gained from systematizing knowledge in this domain are useful to readers—including lessons on theory vs. practice, standardization efforts, and real-world deployments.

REFERENCES

[1] M. Abdalla. Password-based authenticated key exchange: An overview. In *ProvSec*, 2014.
[2] M. Abdalla and M. Barbosa. Perfect forward security of SPAKE2. *IACR ePrint*, 1194, 2019.
[3] M. Abdalla, F. Benhamouda, and P. MacKenzie. Security of the J-PAKE password-authenticated key exchange protocol. In *IEEE S&P*, 2015.
[4] M. Abdalla et al. Security analysis of CPace. *IACR ePrint*, 114, 2021.
[5] M. Abdalla, B. Hasse, and J. Hesse. CPace, a balanced composable PAKE. *IETF draft-irtf-cfrg-cpace-02*, 2021.
[6] M. Abdalla and D. Pointcheval. Simple password-based encrypted key exchange protocols. In *CT-RSA*, 2005.
[7] D. de Almeida Braga, P. Fouque, and M. Sabt. Dragonblood is still leaking: Practical cache-based side-channel in the wild. In *ACSAC*, 2020.
[8] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* John Wiley & Sons, 2020.
[9] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Crypto*, 1995.
[10] V. Balasubramaniyan et al. Pindr0p: Using single-ended audio features to determine call provenance. In *CCS*, 2010.
[11] J. Becerra et al. Forward secrecy of SPAKE2. In *ProvSec*, 2018.
[12] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated key exchange secure against dictionary attacks. In *Eurocrypt*, 2000.
[13] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS*, 1993.
[14] M. Bellare and P. Rogaway. The AuthA protocol for password-based authenticated key exchange. https://www.cs.ucdavis.edu/~rogaway/papers/autha.pdf, 2000. Accessed 4 August 2021.
[15] S. Bellovin and M. Merritt. Encrypted key exchange: password-based protocols secure against dictionary attacks. In *IEEE S&P*, 1992.
[16] S. Bellovin and M. Merritt. Augmented encrypted key exchange: A password-based protocol secure against dictionary attacks and password file compromise. In *CCS*, 1993.
[17] Jens Bender, Marc Fischlin, and Dennis Kügler. Security analysis of the PACE key-agreement protocol. In *ISC*, 2009.
[18] BlackBerry. Native SDK for BlackBerry 10. https://developer.blackberry.com/native/reference/core/com.qnx.doc.crypto.lib_ref/topic/hu_ECSPEKEKeyGen.html, 2021. Accessed 4 August 2021.
[19] Bluetooth. Bluetooth Core Specification 5.3. https://www.bluetooth.com/specifications/specs/core-specification/, 2021. Accessed 4 August 2021.
[20] J. Bonneau, C. Herley, P.C. van Oorschot, and F. Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *IEEE S&P*, 2012.
[21] C. Boyd, A. Mathuria, and D. Stebila. *Protocols for Authentication and Key Establishment (2nd edition).* Springer Nature, 2019.
[22] V. Boyko, P. MacKenzie, and S. Patel. Provably secure password-authenticated key exchange using Diffie-Hellman. In *Eurocrypt*, 2000.
[23] E. Bresson, O. Chevassut, and D. Pointcheval. Security proofs for an efficient password-based key exchange. In *CCS*, 2003.
[24] E. Brier et al. Efficient indifferentiable hashing into ordinary elliptic curves. In *Crypto*, 2010.

[25] BSI, Germany (Federal Office for Information Security). Advanced security mechanism for machine readable travel documents and eIDAS Token. *BSI TR-03110*, 2016. (v1.0 published in 2006, latest v2.1 in 2016).

[26] D. Cash, E. Kiltz, and V. Shoup. The twin Diffie-Hellman problem and applications. In *Eurocrypt*, 2008.

[27] S. Checkoway et al. On the practical exploitability of Dual EC in TLS implementations. In *USENIX Security*, 2014.

[28] D. Clarke and F. Hao. Cryptanalysis of the Dragonfly key exchange protocol. *IET Info. Security*, 8(6):283–289, 2014.

[29] T. Claverie and J. Esteves. BlueMirror: Reflections on Bluetooth pairing and provisioning protocols. In *USENIX WOOT*, 2021.

[30] J. Coron, A. Gouget, T. Icart, and P. Paillier. Supplemental access control (PACE v2): Security analysis of PACE integrated mapping. In *Cryptography and Security: From Theory to Applications*, pages 207–232, 2012.

[31] J. Engler, C. Karlof, E. Shi, and D. Song. PAKE-based web authentication: The good the bad and the hurdles. In *IEEE Web 2.0 S& P Workshop*, 2009.

[32] D. Felsch et al. The dangers of key reuse: practical attacks on IPsec IKE. In *USENIX Security*, 2018.

[33] D. Florêncio, C. Herley, and P.C. van Oorschot. An administrator's guide to internet password research. In *USENIX LISA*, 2014.

[34] X. Gao, J. Ding, J. Liu, and L. Li. Post-quantum secure remote password protocol from RLWE problem. In *Inscrypt*, 2017.

[35] R. Gennaro and Y. Lindell. A framework for password-based authenticated key exchange. In *Eurocrypt*, 2003.

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

[37] D. Goldreich and Y. Lindell. Session-key generation using human passwords only. In *Crypto*, 2001.

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

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

[40] F. Hao. On robust key agreement based on public key authentication. In *FC*, 2010.

[41] F. Hao. Schnorr non-interactive zero-knowledge proof. *RFC 8235*, 2017.

[42] F. Hao. Prudent practices in security standardization. *IEEE Communications Standards Magazine*, 2021.

[43] F. Hao et al. A 2-round anonymous veto protocol. In *SPW*, 2006.

[44] F. Hao, R. Metere, S Shahandashti, and C. Dong. Analyzing and patching SPEKE in ISO/IEC. *IEEE TIFS*, 13(11):2844–2855, 2018.

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

[46] F. Hao and S. Shahandashti. The SPEKE protocol revisited. In *SSR*, 2014.

[47] D. Harkins. Simultaneous Authentication of Equals: A secure, password-based key exchange for mesh networks. In *SensorComm*, 2008.

[48] D. Harkins. Dragonfly Key Exchange. *RFC 7664*, 2015.

[49] A. Hernández, S. Scott, N. Sullivan, R. Wahby, and C. Wood. Hashing to elliptic curves. *IETF draft-irtf-cfrg-hash-to-curve-11*, 2021.

[50] T. Icart. How to hash into elliptic curves. In *Crypto*, 2009.

[51] 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. Accessed 4 August 2021.

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

[53] D. Jablon. Extended password key exchange protocols immune to dictionary attack. In *IEEE Workshop on ETICE*, 1997.

[54] D. Jablon. Password authentication using multiple servers. In *CT-RSA*, 2001.

[55] S. Jarecki, H. Krawczyk, and J. Xu. OPAQUE: An asymmetric PAKE protocol secure against pre-computation attacks. In *Eurocrypt*, 2018.

[56] B. Jaspan. Dual-workfactor encrypted key exchange: efficiently preventing password chaining and dictionary attacks. In *USENIX Security*, 1996.

[57] S. Jiang and G. Gong. Password based key exchange with mutual authentication. In *SAC*, 2004.

[58] R. Kainda, I. Flechais, and A. Roscoe. Usability and security of out-of-band channels in secure device pairing protocols. In *SOUPS*, 2009.

[59] J. Katz, R. Ostrovsky, and M. Yung. Efficient password-authenticated key exchange using human-memorable passwords. In *Eurocrypt*, 2001.

[60] J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM*, 57(1):1–39, 2009.

[61] J. Katz and V. Vaikuntanathan. Round-optimal password-based authenticated key exchange. In *TCC*, 2011.

[62] C. Kaufman and R. Perlman. PDM: A new strong password-based protocol. In *USENIX Security*, 2001.

[63] H. Kilinc and T. Yanik. A survey of SIP authentication and key agreement schemes. *IEEE Comm. Surv. & Tut.*, 16(2):1005–1023, 2013.

[64] K. Kobara and H. Imai. Pretty-simple password-authenticated key-exchange under standard assumptions. *IEICE Trans.*, E85-A(10), 2002.

[65] N. Koblitz and A. Menezes. The random oracle model: A twenty-year retrospective. *Des. Code Crypto.*, 77(2-3):587–610, 2015.

[66] H. Krawczyk, D. Bourdrez, K. Lewi, and C. Wood. The OPAQUE asymmetric PAKE protocol. *IETF draft-irtf-cfrg-opaque-6*, 2021.

[67] C. Kudla and K. Paterson. Modular security proofs for key agreement protocols. In *AsiaCrypt*, 2005.

[68] T. Kwon. Authentication and key agreement via memorable password. In *NDSS*, 2001.

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

[70] W. Ladd et al. SPAKE2, a PAKE. *IETF draft-irtf-cfrg-spake2-20*, 2021.

[71] J. Lancrenon and M. Škrobot. On the provable security of the Dragonfly protocol. In *ISC*, 2015.

[72] J. Lancrenon, M. Škrobot, and Q. Tang. Two more efficient variants of the J-PAKE protocol. In *ACNS*, 2016.

[73] T Lomas, Li Gong, J Saltzer, and R Needhamn. Reducing risks from poorly chosen keys. *SIGOPS Operating Systems Review*, 23(5):14–18, 1989.

[74] S. Lucks. Open Key Exchange: How to defeat dictionary attacks without encrypting public keys. In *SPW*, 1997.

[75] M. Manulis, et al. Secure modular password authentication for the web using channel bindings. *Int. J. Inf. Sec.*, 15(6):597–620, 2016.

[76] P. MacKenzie. Personal email communication to first author, 17-May-2021.

[77] P. Mackenzie. More efficient password-authenticated key exchange. In *CT-RSA*, 2001.

[78] P. MacKenzie. On the security of the SPEKE password-authenticated key exchange protocol. *IACR ePrint*, 2001, 2001.

[79] P. MacKenzie. The PAK suite: Protocols for password-authenticated key exchange. In *DIMACS Technical Report*, 2002.

[80] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. In *AsiaCrypt*, 2000.

[81] P. MacKenzie, S. Patel, and R. Swaminathan. Password-authenticated key exchange based on RSA. *Journal of Info. Security*, 9(6):387–410, 2010.

[82] A. Menezes. Another look at HMQV. *J. Math. Crypt.*, 1(1):47–64, 2007.

[83] A. Menezes, P. van Oorschot, and S. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996. https://cacr.uwaterloo.ca/hac/.

[84] P. van Oorschot. *Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin (2nd edition)*. Springer International, 2021.

[85] P. van Oorschot and M. Wiener. On Diffie-Hellman key agreement with short exponents. In *Eurocrypt*, 1996.

[86] S. Patel. Number theoretic attacks on secure password schemes. In *IEEE S&P*, 1997.

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

[88] S. Schröder, et al. When SIGNAL hits the fan: On the usability and security of state-of-the-art secure mobile messaging. In *EuroUSEC*, 2016.

[89] S. Smyshlyaev, et al. On the security of one password authenticated key exchange protocol. *IACR ePrint*, 2015, 2015.

[90] S. Smyshlyaev, et al. The security evaluated standardized password-authenticated key exchange (SESPAKE) protocol. *RFC 8133*, 2017.

[91] A. Shallue and C. van de Woestijne. Construction of rational points on elliptic curves over finite fields. In *ANT*, 2006.

[92] S. Shin and K. Kobara. Efficient augmented password-only authentication and key exchange for IKEv2. *RFC 6628*, 2012.

[93] S. Shin, K. Kobara, and H. Imai. Security proof of AugPAKE. *IACR ePrint*, 334, 2010. See also [92].

[94] V. Shoup. Security Analysis of SPAKE2+. In *TCC*, 2020.

[95] SRP Protocol Design. http://srp.stanford.edu/design.html. Includes description of SRP-6a. Accessed 4 August 2021.

[96] A. Takahashi and M. Tibouchi. Degenerate fault attacks on elliptic curve parameters in OpenSSL. In *Euro S&P*, 2019.

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

[98] I. Vazquez et al. PakeMail: Authentication and key management in decentralized secure email and messaging via PAKE. In *ICETE*, 2020.

[99] Y. Wang. Submission D2001-06-21 to IEEE P1363.2. https://webpages.uncc.edu/yonwang/papers/ecsrp.pdf, 2001. Accessed 4 August 2021.

[100] M. Weir et al. Testing metrics for password creation policies by attacking large sets of revealed passwords. In *CCS*, 2010.

[101] T. Wu. The Secure Remote Password protocol. In *NDSS*, 1998.

[102] T. Wu. SRP-6: Improvements and refinements to the secure remote password protocol. *Submission to IEEE P1363.2*, 2002.

[103] Z. Zhao, et al. Security analysis of a password-based authentication protocol proposed to IEEE 1363. *TCS*, 352(1-3):280–287, 2006.

[104] M. Zhang. Analysis of the SPEKE password-authenticated key exchange protocol. *IEEE Comm. Letters*, 8(1):63–65, 2004.

[105] P. Zimmermann, A. Johnston, and J. Callas. ZRTP: Media path key agreement for unicast secure RTP. *IETF RFC 6189*, 2011.

Acknowledgements. We thank those who provided constructive feedback that improved this paper, and Philip MacKenzie regarding his awareness of use of PAK protocols in real-world applications. The first author is supported by EPSRC [EP/T014784/1]. The second author acknowledges NSERC for funding his Canada Research Chair and Discovery Grant.

A APPENDIX

Here we give further details on the relationships between the five classes of PAKE protocols in the taxonomy. We also provide algorithmic summaries for the representative schemes, both for reference and to support the conclusions in Section 4 regarding computational cost comparisons.

A.1 PAKE taxonomy

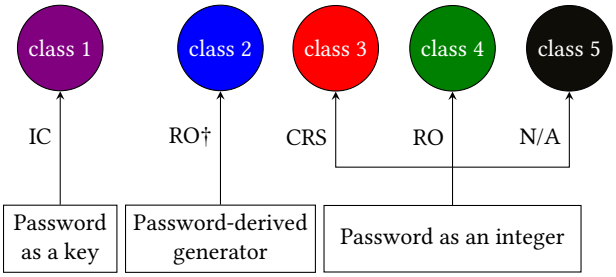

Figure 2: Taxonomy. IC: ideal cipher model. RO†: random oracle model assuming secure hash-to-group function (hash-to-curve in EC setting). CRS: common reference string model assuming a trusted setup. RO: random oracle model assuming secure one-way hash function. N/A: no applicable model or established proofs.

Three common ways to use a password in a PAKE protocol are: 1) as an encryption key; 2) as an input string to derive a generator; and 3) as an integer used in modular arithmetic.[4] The first two ways correspond to Class 1 and 2 resp. (Fig. 2). When a password is used as an integer in modular arithmetic (the modulus for exponent arithmetic being $p-1$ for protocols like EKE2 and SRP that use the whole range of Z_p^*, or q in protocols using a subgroup of Z_p^* of prime-order q), there are three further cases depending on a protocol's design and proof model. In general, in Class 3 security proofs are in a CRS model, assuming a trusted setup; Class 4 proofs are in a RO model, assuming a secure one-way function; and Class 5 protocols lack commonly accepted security proofs in a formal model.[5]

Representative schemes from these five classes are listed in Table 5, with algorithmic summaries given in the sections below, along with detailed explanations of computational costs to support the costs summarized in Table 3 (§4.2). Recall that w denotes an (often weak) password, and to simplify our descriptions here, we use a one-way hash H as a key derivation function. As before, for the MODP case we choose a 3072-bit modulus for concreteness (allowing Table 3's pragmatic comparison of the costs of short and long exponentiations).

[4]The PDM (Password-Derived Moduli) protocol of Kaufman and Perlman [62] uses a client password to derive a modulus, but PDM is viewed as a password-authenticated key retrieval (PAKR) protocol [51], with properties distinct from PAKE protocols. PDM is thus omitted from our taxonomy. We are not aware of any PAKE protocol based on a password-derived modulus—perhaps due to per-session modulus derivation adding a considerable cost [21].

[5]Among Class 5 protocols, only AugPAKE claims to have formal security proofs, but to date it appears in only an unrefereed technical report [93], and the validity of the proofs has been questioned [55].

A.2 EKE2 protocol (Class 1)

Alice (A)		Bob (B)
$x \in_R [0, p-1]$	$A, \mathcal{E}_w(g^x \bmod p) \rightarrow$	
	$\leftarrow B, \mathcal{E}_w(g^y \bmod p)$	$y \in_R [0, p-1]$
Compute K		Compute K

Figure 3: EKE2 protocol [12]

Fig. 3 summarizes EKE2 [12] in Z_p^*. Here p is a *safe* prime and g a primitive root (generator) mod p. A and B denote the identities of Alice and Bob resp. The session key is ($\|$ denotes concatenation): $K = H(A \,\|\, B \|\, g^x \|\, g^y \|\, g^{xy})$. Given a 3072-bit safe-prime modulus p, each party performs two exponentiations (3072-bit exponent): one to compute an ephemeral public key and one to compute the session key. There appears no easy way to extend this specification to other groups settings, e.g., DSA and EC, without modifying the protocol.

A.3 Patched SPEKE protocol (Class 2)

Alice (A)		Bob (B)
$x \in_R [1, q-1]$	$A, f(w)^x \bmod p \rightarrow$	Validate key
Validate key	$\leftarrow B, f(w)^y \bmod p$	$y \in_R [1, q-1]$
Compute K		Compute K

Figure 4: Patched SPEKE [44] (spec: ISO/IEC 11770-4:2017)

Fig. 4 summarizes Patched SPEKE [44]. It works in a subgroup of Z_p^* of prime order q, with $p = 2 \cdot q + 1$ a *safe* prime. Function f is defined: $f(w) = H(w)^2 \bmod p$. To validate the received key is to check it is in Z_p^* (non-zero), and not 1 or $p-1$ (small subgroup elements). The session key is computed as: $K = H\big(sID \,\|\, f(w)^{xy}\big)$, where sID includes user identities and exchanged items. Given a 3072-bit safe-prime modulus p, Alice performs two exponentiations (3071-bit exponent in Z_q): 1 for computing an ephemeral public key $f(w)^x$, and 1 for computing the session key. The cost for Bob is the same.

Class	Representative scheme(s)
1	EKE2
2	Patched SPEKE, CPace, OPAQUE
3	SPAKE2
4	J-PAKE
5	SRP-6a

Table 5: Summary of representative schemes

To extend the protocol to a DSA group requires a hash-to-group function; see IEEE 1363.2 [51]. Given a DSA group with 3072-bit p and 256-bit q, Alice performs one exponentiation (2815-bit exponent) to compute $f(w) = H(w)^{(p-1/q)}$, and three further exponentiations (256-bit exponent): one to generate an ephemeral public key $f(w)^x$, one to validate the received public key $f(w)^y$, and one to compute the session key. Here, to validate a public key in a DSA group requires checking that the received value is in Z_p^* (implying also non-zero), not equivalent (mod p) to 1, then one exponentiation raising it to the power q and confirming the result equals 1. This is in contrast to the safe-prime case above, where validation is simply checking that the received value is in Z_p^*, and not 1 or $p-1$. Extending the protocol to an EC setting requires a hash-to-curve function to implement $f(w)$. In the EC setting, the cost of public key validation is negligible. Therefore, the computational cost for each party becomes one H2C plus two scalar multiplications.

A.4 CPace protocol (Class 2)

Alice (A)		Bob (B)
$SID_A \in_R \{0,1\}^n$	$\xrightarrow{A, SID_A}$	$SID_B \in_R \{0,1\}^n$
Validate key	$\xleftarrow{B, SID_B, f(T)^y}$	$y \in_R [1, q-1]$
$x \in_R [1, q-1]$	$\xrightarrow{A, f(T)^x}$	Validate key
Compute K		Compute K

Figure 5: CPace protocol [39]

Fig. 5 summarizes CPace [39]. It works in a subgroup of Z_p^* of prime order q where $q \mid p-1$. Modular operations are mod p. SID_A and SID_B are random bit strings generated by A and B resp., specified to be of length $n = 512$ bits in the CPace reference implementation [39]. $T = H(SID_A \| SID_A \| w \| A \| B)$. The session key is computed as: $K = H(SID_A \| SID_B \| f(T)^{xy})$ where f is the same hash-to-group function as in Patched SPEKE (f in Fig. 4). To validate a received key is to check if it is an element in the designated prime-order group. Extending the protocol to an EC setting requires a hash-to-curve function to implement f. While the original paper [39] only defines CPace in an EC setting assuming an unspecified hash-to-curve function, here we define it for MODP and EC settings using the same f as in other Class-2 protocols. CPace's computational cost is the same as Patched SPEKE (Fig. 4).

A.5 OPAQUE protocol (Class 2)

Fig. 6 summarizes OPAQUE [55]. It operates in a subgroup of Z_p^* of prime order q, where $q \mid p-1$. Modular operations are mod p. At the registration phase, the user and the server run an Oblivious Pseudorandom Functions (OPRF) protocol, whose central component is a hash-to-group f function (the same f as in Patched SPEKE in Fig. 4, and CPace in Fig. 5). The OPAQUE authors emphasize that the registration must be done over a secure channel, otherwise the protocol security can be compromised. At the end of the registration process, the server stores long-term secret keys k and p_s (chosen from

Registration		
Client (C)		Server (S)
$p_u \in_R [0, q-1]$	$\xrightarrow{C}$	$k, p_s \in_R [0, q-1]$
$P_u = g^{p_u}$	$\xleftarrow{k, P_s}$	$P_s = g^{p_s}$
$m = H\left(w, f(w)^k\right)$		
$c = \mathcal{E}_m(p_u, P_u, P_s)$	$\xrightarrow{c}$	Stores $(C: k, p_s, c, P_u)$
Login		
$r, x \in_R [0, q-1]$		
$\alpha = f(w)^r, X = g^x$	$\xrightarrow{C, \alpha, X}$	$y \in_R [0, q-1]$
$m = H(w, \beta^{1/r})$	$\xleftarrow{S, \beta, c, Y}$	$\beta = \alpha^k, c, Y = g^y$
$p_u, P_u, P_s \leftarrow \mathcal{D}_m(c)$		
$K = KE(p_u, x, P_s, Y)$		$K = KE(p_s, y, P_u, X)$

Figure 6: The OPAQUE protocol [55]

Z_q), together with $P_u = g^{p_u}$ and $c = \mathcal{E}_m(p_u, P_u, P_s)$ where $\mathcal{E}$ is an authenticated encryption scheme with key m. The client has selected $p_u \in Z_q$, computed $P_u = g^{p_u}$ and c, and sent c to the server as shown.

In the login phase, $\mathcal{D}$ is an authenticated decryption scheme. KE is an authenticated key exchange scheme. When HMQV [82] is used to instantiate KE as originally recommended [55], both parties compute $d = \bar{H}(X, s), e = \bar{H}(Y, u)$ where $\bar{H}$ is a one-way hash of $\lceil \log_2 q \rceil / 2$ bits in the output. The session key is $K = H\left(g^{(x+dp_u)(y+ep_s)}\right)$. Originally [55], OPAQUE is defined only in an EC setting assuming an unspecified hash-to-curve function; we generalize this here to MODP and EC settings, using the same f function as other Class-2 protocols.

For a 3072-bit safe-prime modulus $p = 2q+1$ (3071-bit exponents in Z_q) the client needs 4.5 exponentiations: 1 to compute $f(w)^r$; 1 to compute g^x; 1 to compute $\beta^{1/r}$; and 1.5 to compute session key $K = H\left((Y \cdot {P_s}^e)^{x+dp_u}\right) = H\left(g^{(x+dp_u)(y+ep_s)}\right)$. The server performs 3.5 exponentiations (3071-bit exponent): 1 to compute α^k; 1 for g^y; and 1.5 for the session key.

If the protocol is implemented in a DSA group with 3072-bit p and 256-bit q, the client must perform one exponentiation (2815-bit exponent) to compute $f(w)$, and 6.5 further exponentiations (256-bit exponent) including 3 to compute $f(w)^r$, g^x, and $\beta^{1/r}$ resp., 2 to verify β and Y are valid elements in the prime-order subgroup, and 1.5 to compute the session key. The server must perform 5.5 exponentiations (256-bit exponent): 2 to verify the received α and X are valid elements in the prime-order subgroup resp., 2 to compute α^k and g^y resp., and 1.5 to compute the session key. In the EC setting, the cost of public key validation is negligible, with client cost H2C plus 4.5 scalar multiplications, and server cost 3.5 scalar multiplications.

After OPAQUE was selected by the IETF in 2020, the designers replaced HMQV with 3DH [67, §5.1]. (3DH was used in an early version of Signal as the key exchange protocol; it was replaced by X3DH.[6]) As the modified OPAQUE is still being defined in an IETF internet draft [66] when we write this paper, we do not analyze its efficiency here.

[6] For X3DH, see: https://signal.org/docs/specifications/x3dh/

A.6 SPAKE2 protocol (Class 3)

Alice (A)		Bob (B)
$x \in_R [0, q-1]$	$\xrightarrow{A, g^x M^w \bmod p}$	Validate key
Validate key	$\xleftarrow{B, g^y N^w \bmod p}$	$y \in_R [0, q-1]$
Compute K		Compute K

Figure 7: SPAKE2 protocol [6].

Fig. 7 summarizes SPAKE2 [6]. It works in a subgroup in Z_p^* of prime order q, where $q \mid p-1$. The trusted setup includes three generators {g, M, N}; the discrete log relationship between every pair of these must remain unknown. To validate a received key is to confirm that it is an element in the prime-order subgroup (cf. §A.3). The session key is $K = H(A, B, g^x, g^y, w, g^{xy})$. Given a DSA group with 3072-bit p and 256-bit q, Alice performs three exponentiations (256-bit exponent): one to compute $g^x M^w$ (using a simultaneous computation technique [83, §14]), one to verify the received public key $g^y N^w$ is a valid element in the prime-order subgroup, and one to compute g^{xy} in the session key. Bob's cost is the same. In an EC setting, the algorithm works the same, but the cost of public key validation is negligible (so the main cost for each party is two scalar multiplications).

A.7 J-PAKE protocol (Class 4)

Alice (A)		Bob (B)
$x_1 \in_R [0, q-1]$		
$x_2 \in_R [1, q-1]$	$\xrightarrow{A, g^{x_1}, g^{x_2}, \mathrm{ZKP}\{x_1, x_2\}}$	Validate ZKPs
		$y_1 \in_R [0, q-1]$
Validate ZKPs	$\xleftarrow{B, g^{y_1}, g^{y_2}, \mathrm{ZKP}\{y_1, y_2\}}$	$y_2 \in_R [1, q-1]$
Validate ZKP	$\xleftarrow{\beta^{y_2 \cdot w}, \mathrm{ZKP}\{y_2 \cdot w\}}$	
	$\xrightarrow{\alpha^{x_2 \cdot w}, \mathrm{ZKP}\{x_2 \cdot w\}}$	Validate ZKP
Compute K		Compute K

Figure 8: J-PAKE protocol [45]

Fig. 8 summarizes J-PAKE [45]. It works in a subgroup in Z_p^* of prime order q where $q \mid p-1$. All modular operations are mod p. To validate a ZKP (technically, a Schnorr non-interactive ZKP) means verifying simple equations [41, §2.4] to confirm that the sender knows the exponent; this takes one exponentiation to generate the ZKP and two to verify it. Let $\alpha = g^{y_1} g^{y_2} g^{x_1}$ and $\beta = g^{x_1} g^{x_2} g^{y_1}$ serve as new generators to compute $U = \alpha^{x_2 \cdot w}$ and $V = \beta^{y_2 \cdot w}$. Alice computes the session key: $K = H((V / g^{y_2 \cdot x_2 \cdot w})^{x_2}) = H\left(g^{(x_1+y_1) \cdot x_2 \cdot y_2 \cdot w}\right)$. Symmetrically, Bob computes the same session key: $K = H((U / g^{x_2 \cdot y_2 \cdot w})^{y_2}) = H\left(g^{(x_1+y_1) \cdot x_2 \cdot y_2 \cdot w}\right)$. Given a DSA group with 3072-bit p and 256-bit q, Alice performs 14 exponentiations (256-bit exponent): 3 to compute g^{x_1}, g^{x_2}, and $\alpha^{x_2 \cdot w}$; 3 to compute the three ZKPs; 6 to validate three received ZKPs; and 2 to compute the session key. The cost for Bob is the same. In an EC setting the protocol works the same—but here, because the public key validation incurs negligible cost, it takes one scalar multiplication to generate the ZKP, and one (vs. two in a DSA group) to verify it [41, §3.4], so overall each party performs 11 scalar multiplications.

A.8 SRP-6a protocol (Class 5)

Client (C)		Server
$a \in_R [2, p-1], A = g^a$	$\xrightarrow{C, A}$	Look up s, v
		$b \in_R [2, p-1]$
$x = H(s, w), u = H(A, B)$	$\xleftarrow{s, B}$	$B = k \cdot v + g^b$
$S = (B - k \cdot g^x)^{a+u \cdot x}$		$u = H(A, B)$
$K = H(S)$		$S = (Av^u)^b$
$M_1 = H\Big(H(p) \oplus H(g),$		$K = H(S)$
$H(C), s, A, B, K\Big)$	$\xrightarrow{M_1}$	Check M_1
Check M_2	$\xleftarrow{M_2}$	$M_2 = H(A, M_1, K)$

Figure 9: SRP-6a (http://srp.stanford.edu/)

.

Fig. 9 summarizes SRP-6a. It works in the whole range of a multiplicative group Z_p^* where $p = 2 \cdot q + 1$ is a *safe* prime and g is a primitive root (generator) mod p. All modular operations are mod p. At registration, the server stores s and $v = g^{H(s,w)}$, where s is a salt (e.g., 64-bit). During the login phase (Fig. 9), $k = H(p, g)$. SRP-6a differs from SRP-6 [102] which uses $k = 1$, and from SRP-3 [101] which also uses $k = 1$ but there u is a "randomly generated parameter" instead of $H(A, B)$. Also, while SRP-3 and SRP-6 require 6 flows, SRP-6a needs only 4. SRP-6a improves the round efficiency by sending C and A in one flow instead of in separate flows. Given a 3072-bit modulus p and a 256-bit hash function H, the client performs one exponentiation (3072-bit exponent) to compute g^a, one exponentiation (256-bit exponent) to compute g^x, and another exponentiation (3072-bit exponent) to compute S. The server performs one exponentiation (3072-bit exponent) to compute g^b, one exponentiation (256-bit exponent) to compute v^u and another exponentiation (3072-bit exponent) to compute S.

There appears no direct way to extend SRP-6a (which supports MODP, with a safe prime) to other group settings (DSA and EC). However, the little-known SRP-5 [99] (see also Zhao et al. [103, §5]) was specifically designed for EC implementation (on the other hand, not supporting MODP), and is included in IEEE 1363.2 [51, §9.9], but is best viewed as a distinct protocol and has received less analytic attention.