IEEE Standard Specifications for Password-Based Public-Key Cryptographic Techniques

IEEE Computer Society

Sponsored by the
Microprocessor Standards Committee

1363.2™

IEEE
3 Park Avenue
New York, NY 10016-5997, USA

29 January 2009

IEEE Std 1363.2™-2008

IEEE Standard Specification for Password-Based Public-Key Cryptographic Techniques

Sponsor

Microprocessor Standards Committee
of the
IEEE Computer Society

Approved 26 September 2008

IEEE-SA Standards Board

Abstract: This standard covers specifications of public-key cryptographic techniques for password-based authentication and key establishment, supplemental to the techniques described in IEEE Std 1363™-2000 and IEEE Std 1363a™-2004. It is intended as a companion standard to IEEE Std 1363-2000 and IEEE Std 1363a-2004. It includes specifications of primitives and schemes designed to utilize passwords and other low-grade secrets as a basis for securing electronic transactions, including schemes for password-authenticated key agreement and password-authenticated key retrieval.
Keywords: authentication, key agreement, password, public-key cryptography

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA

2nd Printing 12 June 2009. The following ISBNs were incorrect in the 1st Printing on 29 January 2009.
PDF: ISBN 978-0-7381-5806-8 STD95824
Print: ISBN 978-0-7381-5807-5 STDPD95824

The correct ISBNs are:
PDF: ISBN 978-0-7381-6016-0 STD95824
Print: ISBN 978-0-7381-6017-7 STDPD95824

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon his or her independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be submitted to the following address:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854
USA

Introduction

This introduction is not part of IEEE Std 1363.2-2008, IEEE Standard Specification for Password-Based Public-Key Cryptographic Techniques.

The history of the IEEE P1363.2[a] project began in late 1996 with a presentation to the IEEE P1363 Working Group of a password-based public-key method for key agreement. Originally submitted as material for potential inclusion in the P1363 document, which later became IEEE Std 1363-2000,[b] this class of technique was deemed to be sufficiently interesting and yet sufficiently different from the original focus of the IEEE P1363 Working Group as to merit further study to determine how it should be addressed. At that time, the techniques included in the P1363 draft were fairly stable; however, there were also other additional techniques that had been submitted to the Working Group that warranted further study and consideration.

The P1363a project became the entry point for newly proposed methods that were similar in function to and in the same mathematical families as methods in the P1363 project, which focused on key exchange, digital signature, and public-key encryption schemes in the integer factorization, discrete logarithm (DL), and elliptic curve (EC) families. Work on P1363a progressed while IEEE Std 1363-2000 was solidified and prepared for the IEEE balloting process. In time, the Working Group similarly chose to close P1363a to additional techniques and prepare it for ballot, leaving standardization of additional techniques to future standards. P1363a resulted in the IEEE Std 1363a-2004 amendment.

Throughout the process of creating these standards, the Working Group continued to receive numerous submissions of creative, useful, and well-designed public-key cryptographic techniques, including password-based techniques. To address the number of submitted techniques that fell outside the functional or familial scope of both IEEE Std 1363-2000 and IEEE Std 1363a-2004, the Microprocessor Standards Committee (MSC) commissioned a study group, in which many of the IEEE P1363 Working Group members participated, to explore the possibility of creating additional standards related to public-key cryptography.

In late 2000, the Working Group began work on P1363.2 (password-based public-key cryptographic techniques) and P1363.1 (public-key cryptographic techniques based on hard problems over lattices). The Working Group has also begun work on P1363.3 (identity-based cryptographic techniques using pairings) and has discussed other potential projects to continue the work from IEEE Std 1363a-2004.

The IEEE P1363 Working Group continues to be an excellent forum for experts to discuss technical and standardization issues associated with public-key cryptography. It has provided a focal point for the presentation of new developments in public-key cryptography and remains a source for up-to-date information on the topic. For the duration of its existence, the Working Group intends to maintain a Web site that will support all of the IEEE 1363 standards and current projects.[c]

The IEEE P1363 Working Group would once again like to thank all of the participants and outside experts that have contributed to this standard, to the development of public-key technology and to the P1363 process.

[a] The P indicates an IEEE authorized standards project that was not approved by the IEEE-SA Standards Board at the time the Working Group was formed.
[b] Information on references can be found in Clause 2.
[c] Information can be found at: http://grouper.ieee.org/groups/1363/index.html.

Notice to users

Laws and regulations

Users of these documents should consult all applicable laws and regulations. Compliance with the provisions of this standard does not imply compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Copyrights

Updating of IEEE documents

Users of IEEE standards should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE Standards Association Web site at http://ieeexplore.ieee.org/xpl/standards.jsp, or contact the IEEE at the address listed previously.

For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA Web site at http://standards.ieee.org.

Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/reading/ieee/updates/errata/index.html. Users are encouraged to check this URL for errata periodically.

Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/reading/ieee/interp/index.html.

Patents

Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses. Other Essential Patent Claims may exist for which a statement of assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the IEEE P1363 Working Group had the following membership:

William Whyte, *Chair*
Don B. Johnson, *Vice Chair*
Mike Brenner, *Primary Editor*
David Jablon, *1363.2 Project Editor*

Guido Appenzeller
Xavier Boyen
Daniel Brown
Mark Chimley
Andy Dancer
Luther Martin
Roger Schlafly
Hovav Shacham
Ari Singer
Jerry Solinas
Terence Spies
Yongge Wang

In addition, the Working Group would like to thank the following people for their contributions to the standard:

Wole Akpose
Kendall Ananyi
Mihir Bellare
Liqun Chen
Wei Dai
Warwick Ford
Eric Fung
Craig Gentry
Jim Hughes
Bill Jennings
Burt Kaliski
Pieter Kasselman
Jonathan Katz
David Kravitz
Hugo Krawczyk
Taekyoung Kwon
Philip D. MacKenzie
James H Manger
Chris Mitchell
Rafail Ostrovsky
Zulfikar Ramzan
Phil Rogaway
Victor Shoup
Ram Swaminathan
Michael Wiener
Tom Wu
Kim-Ee Yeoh
Moti Yung

The Working Group apologizes for any inadvertent omissions from the preceding lists. Please note that inclusion of a person's name does not imply that the person agrees with all the materials in the standard.

The following members of the individual balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

Matthew Ball
Mike Brenner
Juan Carreon
Weijen Chen
Keith Chow
Tommy Cooper
Andy Dancer
James Davis
Yaacov Fenster
Michael Geipel
Randall Groves
Werner Hoelzl
Atsushi Ito
David Jablon
Raj Jain
Piotr Karocki
William Lumpkins
G. Luri
Philip D. MacKenzie
Michael Markowitz
Edward McCall
Earl Meiers
Gary Michel
Apurva Mody
Michael S. Newman
Nick S. A. Nikjoo
Vikram Punj
Robert Robinson
Fernando Lucas Rodriguez
Michael Rush
Suman Sharma
Gil Shultz
Steven Smith
Thomas Starai
Rene Struik
Gerald Stueve
Mark-Rene Uchida
William Whyte
Oren Yuen
Janusz Zalewski

When the IEEE-SA Standards Board approved this standard on 26 September 2008, it had the following membership:

Robert M. Grow, *Chair*
Thomas Prevost, *Vice Chair*
Steve M. Mills, *Past Chair*
Judith Gorman, *Secretary*

Victor Berman
Richard DeBlasio
Andy Drozd
Mark Epstein
Alexander Gelman
William R. Goldbach
Arnold M. Greenspan
Kenneth S. Hanus
Jim Hughes
Richard H. Hulett
Young Kyun Kim
Joseph L. Koepfinger*
John Kulick
David J. Law
Glenn Parsons
Ronald C. Petersen
Chuck Powers
Narayanan Ramachandran
Jon Walter Rosdahl
Anne-Marie Sahazizian
Malcolm V. Thaden
Howard L. Wolfman
Don Wright

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

Satish K. Aggarwal, *NRC Representative*
Michael Janezic, *NIST Representative*

Lorraine Patsco
IEEE Standards Program Manager, Document Development

Malia Zaman
IEEE Standards Program Manager, Technical Program Development

Contents

IEEE Standard Specification for Password-Based Public-Key Cryptographic Techniques

1. Overview

1.1 Scope

This standard covers specifications of common public-key cryptographic techniques for performing password-based authentication and key establishment, supplemental to the techniques described in IEEE Std 1363™-2000 and IEEE Std 1363a™-2004.[1] It includes specifications of primitives and schemes designed to utilize passwords and other low-grade secrets as a basis for securing electronic transactions, including schemes for password-authenticated key agreement and password-authenticated key retrieval.

1.2 Purpose

Ensuring privacy and authenticity in personal electronic transactions is a process that necessarily involves human beings. Memorized secrets are an important factor in human authentication. Many common cryptographic methods for authentication require large, random high-grade secret keys; yet, the secrets that human beings can conveniently memorize and reliably reproduce tend to be low-grade secrets. Passwords are widely used low-grade secrets that are typically not-so-random and relatively small, and introduce risks of brute-force attack when inappropriately used as cryptographic keys.

[1] Information on references can be found in Clause 2.

IEEE Std 1363.2™-2008 specifies public-key cryptographic techniques specifically designed to securely perform password-based authentication and key establishment. These techniques provide a way to authenticate people and distribute high-quality cryptographic keys for people, while preventing off-line brute-force attacks associated with passwords. A resulting high quality key may be more confidently used in combination with other cryptographic methods, such as symmetric encryption methods and public-key encryption, identification, and digital signature methods. IEEE Std 1363.2-2008 provides a reference for a variety of such password-based techniques within a suitable framework.

It is not the purpose of this document to mandate any particular set of password-based techniques or security requirements (including key sizes). Rather, the purpose is to provide: (1) a reference for specification of a variety of techniques from which applications may select, (2) the appropriate theoretic background, and (3) extensive discussion of security and implementation considerations so that a solution provider can choose appropriate security requirements.

1.3 Organization of the document

This standard is written as a companion document to IEEE Std 1363-2000 and IEEE Std 1363a-2004. IEEE Std 1363.2-2008 can be used as a standalone reference for description of password-based methods, but it heavily relies upon background material in, and uses techniques defined in, these earlier documents. Some familiarity with IEEE Std 1363-2000 and the field of cryptography is assumed.

The structure of this document roughly parallels that of IEEE Std 1363-2000, to make it easy to find relevant information, and to facilitate a potential merger of these documents in a future version of the standard.

This document contains two parts: the main document and seven annexes. The annexes provide background and helpful information for the users of this standard, as follows.

1.3.1 Structure of the main document

— Clause 1, Overview

— Clause 2, Normative references

— Clause 3, Definitions, acronyms, and abbreviations

— Clause 4, Types of cryptographic techniques

— Clause 5, Mathematical conventions

— Clause 6, The DL setting

— Clause 7, The EC setting

— Clause 8, Primitives

— Clause 9, Password-authenticated key agreement schemes

— Clause 10, Password-authenticated key retrieval schemes

— Clause 11, Key derivation functions

— Clause 12, Auxiliary techniques

1.3.2 Structure of the annexes

The annexes include the following:

— Annex A (informative) Number-theoretic background

— Annex B (normative) Conformance

— Annex C (informative) Rationale

— Annex D (informative) Security considerations

— Annex E (informative) Formats

— Annex F (informative) Bibliography

1.4 Document conventions

1.4.1 Bracket notation

For method descriptions that describe two or more related methods that work in different contexts, context-specific description may be found in { } brackets. Examples of how this notation is used are as follows:

— {DL,EC}, and the corresponding {discrete logarithm, elliptic curve}, are used to describe methods in both the discrete logarithm (DL) and elliptic curve (EC) families.

— {CLIENT,SERVER}, and the corresponding {Client, Server}, are used to describe pairs of methods for both parties in a two-party Client/Server scheme.

— {SRP3,SRP6} is used to describe each of two similar methods.

— {DL,EC}BPKAS-PAK-{CLIENT,SERVER} is equivalent to {DLBPKAS-PAK-CLIENT, DLBPKAS-PAK-SERVER, ECBPKAS-PAK-CLIENT, ECBPKAS-PAK-SERVER}.

A method description should be read consistently using just one of each alternative context.

The notation *For DL only:* indicates a feature that applies only to the DL context. The two parties that use -CLIENT and -SERVER methods are respectively designated *Client* and *Server*.

In the titles of methods defined in Clause 8, Clause 9, and Clause 10, a prefix of [DL] or [EC] indicates that the method is defined for the DL or EC setting only; otherwise the method is defined for both settings.

1.4.2 Ordering of steps

Many descriptions include an ordered sequence of steps, which may generally be performed in any order that results in an equivalent operation.

However, where a step includes substeps, all substeps shall be performed before the step is considered to be complete. For example, a step that receives a *public-key value* (see 8.1.3) and any associated substeps that abort if the value is unacceptable should all be completed before the value is used in other steps.

Steps that "output [some value] and stop" are used to limit disclosure of sensitive information. These steps shall be performed in the indicated order relative to any subsequent steps that either output a different value or send a value to another party.

1.4.3 Method parameters

Many methods are parameterized by choices of input domain parameters, functions, and related values. These parameters are generally considered to be options of the invoking scheme. For two parties to

successfully interoperate using a method, they need to use the same scheme options or parameters for the method.

Cases where related methods need to use the same value for a shared parameter in order to interoperate are specifically noted.

1.4.4 Future amendment

Where one or more functions are specified as choices that should be used for a scheme option or parameter, an amendment to this standard may specify additional functions that are also acceptable choices.

2. Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 1363-2000, IEEE Standard Specifications for Public-Key Cryptography.[2, 3]

IEEE Std 1363a-2004, IEEE Standard Specifications for Public-Key Cryptography, Amendment 1: Additional Techniques.

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply. *The Authoritative Dictionary of IEEE Standards Terms* [B21][4] should be referenced for terms not defined in this subclause.

NOTE—For some terms, multiple definitions are presented, in which case the first represents the primary definition for use within this document. Alternate definitions show different meanings that may be found in referenced literature and other common use, and are presented for historical context. See Annex D for further background discussion of these terms and related concepts.[5]

3.1.1 augmented password-authenticated key agreement (APKAS): A system for password-authenticated key agreement between a client and server that forces an adversary to perform a successful brute-force attack in order to masquerade as the client using stolen server data.

3.1.2 balanced password-authenticated key agreement (BPKAS): A system for password-authenticated key agreement in which two parties use the same shared password value to negotiate and authenticate a shared key.

[2] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).
[3] The standards or products referred to in this clause are trademarks of the Institute of Electrical and Electronics Engineers, Inc.
[4] The numbers in brackets correspond to those of the bibliography in Annex F.
[5] Notes in text, tables, and figures of a standard are given for information only and do not contain requirements needed to implement this standard.

3.1.3 brute-force attack: An attack on a cryptographic system that systematically employs a trial and error search of a space of keys, passwords, or other secret data.

NOTE—For passwords, a *brute-force attack* is similar to, but may be much easier than, an *exhaustive attack*. It may not be necessary to explore the full space of all possible passwords when the targeted password is likely to be confined to a small subset of that space.

3.1.4 crack (a password): (A) (informal) Successfully brute-force attack password-derived data. **(B)** (informal) Successfully attack a password-based cryptographic system.

3.1.5 entropy (of a password or key): A measure of the uncertainty of the specific value that a password or key may be assigned, which is, roughly speaking, inversely related to the probability of guessing the value.

NOTE—The *entropy* of a password is relative to the state of knowledge and capabilities of the guesser.

3.1.6 dictionary attack (on a password-based system): (A) A brute-force attack using repeated guessing and verification of password values. **(B)** A brute-force attack using a stored list of specific password values. **(C)** A brute-force attack using a stored list of words from a natural language dictionary.

3.1.7 hashed password: The result of a cryptographic hash function of a password.

3.1.8 interactive proof: A two-party protocol in which a prover interacts with a verifier by exchanging messages, where both parties share a common input value, and where at the end of the interaction the verifier computes a predicate depending on the input value and the exchanged messages in order to accept or reject the input value.

NOTE—Definition of interactive proof adapted from Goldreich and Krawczyk [B19].

3.1.9 iterated hash: Repeated application of a cryptographic hash function on input data, typically used to increase the computation needed to crack a password.

3.1.10 low-grade secret: A secret that is vulnerable or presumed to be vulnerable to brute-force attack given the result of a cryptographic hash function of the secret.

3.1.11 password: A low-grade secret word, phrase, number, or sequence of characters used for authentication of identity or authorization.

3.1.12 password-authenticated key agreement: A class of key agreement methods in which parties, based only on their knowledge of password-derived data, establish a cryptographic key using an exchange of messages, such that one who controls the communication channel but does not possess the password-derived data will not be able to participate in the method and will be severely constrained from guessing the password.

3.1.13 password-authenticated key retrieval: (A) A process in which a client obtains a static key that is derived from a password-based negotiation with a server that knows data associated with the password. **(B)** (broadly) The retrieval of a key from a secure key repository or escrow requiring authentication derived in part from a password.

3.1.14 password-entangled public key: A public key derived from both a password and a high-grade secret private key, constructed specifically to avoid password disclosure.

3.1.15 password-limited private key: A private key where the sole source of randomness may be limited to that of a password.

3.1.16 password-limited public key: A value used as password verification data that is derived from a password-limited private key using a public-key generation method.

3.1.17 password verification data: Data that is used to verify a party's knowledge of a specific password and is typically regarded as secret data.

3.1.18 password verifier: (A) One of the parties in a password proof system that verifies knowledge of a password. **(B)** Password verification data.

NOTE—The second definition of *password verifier* is not used in this document, but has been used in relevant references.

3.1.19 PIN: Acronym for personal identification number, a short numeric password.

3.1.20 proof of knowledge: (A) A proof of an assertion that a prover knows something, in a system where whenever a verifier is convinced of the assertion, then the prover indeed knows it. **(B)** A proof of knowledge of a solution to a publicly stated computationally difficult problem.

NOTE—Definition **(A)**, adapted from Bellare and Goldreich [B4], provides a basis for the definition of **zero knowledge password proof**. Many references refer to the narrower definition **(B)**, which is limited to a proof that requires secret knowledge to be from a computationally large space. *See also:* **zero knowledge proof of knowledge**.

3.1.21 salt: (A) A variable incorporated as secondary input to a one-way or encryption function that derives password verification data. **(B)** An element that gives flavor or zest. [Presumed basis for definition **(A)**.]

3.1.22 zero knowledge password proof: An interactive zero knowledge proof of knowledge of password-derived data shared between a prover and the corresponding verifier. *See also:* **interactive proof**; **password; zero knowledge proof of knowledge**.

3.1.23 zero knowledge proof: A proof of an assertion that has the property of yielding nothing but the validity of the assertion.

NOTE—Definition adapted from Goldreich and Krawczyk [B19].

3.1.24 zero knowledge proof of knowledge: A proof of knowledge that has the property of yielding nothing but the validity of the assertion that the prover knows something. *See also:* **proof of knowledge; zero knowledge proof**.

3.2 Acronyms and abbreviations

ANSI	American National Standards Institute
AMP	identifier for techniques based on Kwon [B38]
APKAS	augmented password-authenticated key agreement scheme
BPKAS	balanced password-authenticated key agreement scheme
BS2IP	bit string to integer (conversion) primitive
BS2OSP	bit string to octet string (conversion) primitive
DH	identifier for techniques based on Diffie and Hellman [B10]
DL	discrete logarithm (family of techniques)
EC	elliptic curve (family of techniques)
FE2IP	field element to integer (conversion) primitive

FE2OSP	field element to octet string (conversion) primitive
FIPS	Federal Information Processing Standards
GE2OSP	group element to octet string (conversion) primitive
GE2SVFEP	group element to secret value field element (conversion) primitive
I2BSP	integer to bit string (conversion) primitive
I2FEP	integer to field element (conversion) primitive
I2OSP	integer to octet string (conversion) primitive
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
KCF	key confirmation function
KDF	key derivation function
KRBP	key retrieval blinding primitive
KRPP	key retrieval permutation primitive
KRUP	key retrieval unblinding primitive
MGF	mask generation function
MVCF	multiplier value creation function
NIST	National Institute of Standards and Technology
OS2BSP	octet string to bit string (conversion) primitive
OS2FEP	octet string to field element (conversion) primitive
OS2IP	octet string to integer (conversion) primitive
PAK	identifier for techniques based on Boyko, MacKenzie, Patel [B8]
PEPKGP	password-entangled public-key generation primitive
PKAS	password-authenticated key agreement scheme
PKGP	public-key generation primitive
PKRS	password-authenticated key retrieval scheme
PPK	identifier for techniques based on Boyko, MacKenzie, Patel [B8]
PVDGP	password verification data generation primitive
REDP	random element derivation primitive
SPEKE	identifier for techniques based on Jablon [B31]
SRP	identifier for techniques based on Wu [B57]
SVDP	secret value derivation primitive

4. Types of cryptographic techniques

This clause gives an overview of the types of cryptographic techniques that are specified in this standard as well as some requirements for conformance with those techniques. See Annex B for more on conformance.

4.1 Brief history of the field

The first generation of the era of public cryptography was a time of rapid discovery, research, and development. Within this period, the concept of *zero knowledge proof* was introduced and rapidly refined. However, clear formulations of similar problems in password-based cryptography were initially elusive, like the concept of a *zero knowledge password proof.* [6] How can one party securely prove knowledge of a password to another party over an insecure channel without using other prearranged cryptographic keys? This question was answered only within the last few years of this period. Once the first solutions were found (see Bellovin and Merritt [B7], [B6], and Gong et al. [B20]), the field blossomed, resulting in the methods described and referenced in this document.

4.2 Variations of the network password problem

This standard describes three classes of password-based methods that solve three variations of the password-only network login problem. These methods can provide mutual zero knowledge password proof and remote password-authenticated establishment of cryptographic keys.

In the first class of methods, for *balanced password-authenticated key agreement*, two parties share a common password and they want to prove to each other that they know the password, and only then engage in secure communications, without revealing the password to others.

The second class of *augmented password-authenticated key agreement* methods is similar to the first, except that one of the parties, the Server, has password verification data derived using a one-way function of the password. In this case, although the sensitivity of the password verification data is still dependent on the quality of the password, the one-way function may provide extra protection for the Server's stored password-derived data.

The third class of *password-authenticated key retrieval* methods addresses the scenario where one desires to further decrease the sensitivity of stored password-derived data. These methods may be used to split password verification data into two or more shares, such that no number of shares less than a given threshold permits a brute-force password attack.

All these methods require one or more parties to use specific password-related data to make the method succeed. But when the participant does not use the correct password-related data, then the method is designed to fail in a way that does not reveal the password to those that do not already know the password. An adversary that does not know the appropriate password, who observes a legitimate interaction, or attempts to participate in the method with a legitimate party, is prevented from obtaining information that leads to unconstrained off-line guessing of the password. However, even in optimal cases, all methods permit an adversary to verify a minimum of one guess in each run of the method.

4.3 General model

As stated in Clause 1, the purpose of this standard is to provide a reference for specifications of a variety of password-based public-key cryptographic techniques from which applications may select. Therefore, it is desirable to define these techniques in a framework that would allow selection of techniques appropriate for particular applications. Different types of cryptographic techniques can be viewed abstractly according to the following general model, including primitives, schemes, and additional methods, the definitions of which are in IEEE Std 1363-2000 and are repeated here.

[6] A zero knowledge password proof is, essentially, an interactive zero knowledge proof of knowledge of a small secret. See 3.1.22 for a definition of this and other italicized terms.

— *Primitives*: Basic mathematical operations. Historically, they were discovered based on number-theoretic hard problems. Primitives are not meant to achieve security just by themselves, but they serve as building blocks for schemes.

— *Schemes*: A collection of related operations combining primitives and additional methods. Schemes can provide complexity-theoretic security, which is enhanced when they are appropriately applied in protocols.

— *Additional methods*: Other components of schemes. These include key derivation functions (KDF), hash functions, key confirmation functions (KCF), and other techniques that provide security based on a broad blend of cryptographic analytic assumptions beyond number theory.

From an implementation viewpoint, primitives can be viewed as low-level implementations (e.g., implemented within cryptographic accelerators or software modules), schemes and additional methods can be viewed as medium-level implementations (e.g., implemented within cryptographic service libraries), and protocols can be viewed as high-level implementations (e.g., implemented within entire sets of applications).

In a departure from IEEE Std 1363-2000, this standard defines schemes that interact with related schemes in multiple stages, using intermediate transmitted and received values, as well as input and output values. Intermediate values are exchanged between parties during the operation of these schemes. And in addition to the added functions of password-based authentication, the scope of the key agreement schemes here is also broader in describing key confirmation operations.

This document describes the requirements for schemes at a detailed functional level, but application-specific data transmission formats and techniques are outside the scope of this standard.

General frameworks for primitives, additional methods, and schemes, are provided in 4.4, 4.5, 4.6, and 4.7 and specific techniques are defined in Clause 6 through Clause 12. This document also references techniques defined in IEEE Std 1363-2000 as components for constructing cryptographic primitives and schemes. This standard does not define application-specific protocols; the techniques defined in this standard are rather components used in such protocols. Annex D discusses security considerations related to how the techniques can be used in protocols and applications to achieve certain security attributes.

4.4 Primitives

The following types of primitives are defined in this standard:

— Random element derivation primitives (REDP), components of password-authenticated key agreement schemes (PKAS) and password-authenticated key retrieval schemes (PKRS).

— Password-entangled public-key generation primitives (PEPKGP); components of PKASs and PKRSs.

— Secret value derivation primitives (SVDP), components of password-authenticated key agreement and PKRSs.

— Password verification data generation primitives (PVDGP), components of augmented password-authenticated key agreement schemes (APKAS).

— Key retrieval blinding primitives (KRBP), key retrieval unblinding primitives (KRUP), and key retrieval permutation primitives (KRPP), components of key retrieval schemes.

Primitives in this standard are presented as mathematical operations, and they are typically not meant to provide security benefits by themselves. Such benefits can only be provided by the use of a primitive, along with other operations, in a properly implemented scheme.

Primitives assume that their inputs satisfy certain assumptions, as listed with the specification of each primitive. An implementation of a primitive is unconstrained on an input not satisfying the assumptions, as long as it does not adversely affect future operation of the implementation; the implementation may or may not return an error condition. For example, an implementation of a public-key generation primitive (PKGP) may return something that looks like a public key if its input was not a valid private key. It may also reject the input. It is up to the user of the primitive to guarantee that the input will satisfy the constraints or to include the relevant checks. For example, the user may choose to use relevant key and domain parameter validation techniques.

The specification of a primitive consists of the following information:

— *Input* to the primitive

— *Assumptions* about the input made in the description of the operation performed by the primitive

— *Output* from the primitive

— *Operation* performed by the primitive, expressed as a series of steps

— *Conformance region recommendations* describing the minimum recommended set of inputs for which an implementation should operate in conformance with the primitive (see Annex B for more on conformance)

Primitive specifications are functional specifications, not interface specifications. The format of inputs and outputs and the procedure by which an implementation of a scheme is invoked are outside the scope of this standard. See Annex E for more information on input and output formats.

The input and output values for primitives include *domain parameters* and *keys*, which may be defined as integers, EC points, field elements or *group elements*. These concepts are defined in 6.1 for the DL setting, in 7.1 for the EC setting, and in 8.1 for generalized groups.

4.4.1 Random element derivation primitives (REDP) model

An REDP has the following form:

Input:

— Domain parameters

— A random selector value

Output: The derived generator of the desired group, or “invalid” according to the result.

The REDP applies cryptographic hash operations to the selector value to derive a pseudo-random element of order r that is a generator of the desired group defined by the domain parameters. The function may output “invalid” for an input value that does not correspond to a generator of the desired group; however, the probability of a random input producing an invalid result should be negligibly small for any suitable choice of parameters.

The required security properties of REDPs are discussed in D.2.1.20.

4.4.2 Public-key generation primitives (PKGP) model

A PKGP has the following form:

Input:

— Domain parameters

— A valid private key

Output: The public key, or "invalid"

The PKGP applies cryptographic operations to the input to generate a public key. It outputs the public key or "invalid" according to the result.

NOTE—*Public keys* are a common feature of all key agreement schemes. This document defines the Diffie-Hellman public-key generation primitives (PKGP-DH), in accordance with the definitions of Diffie-Hellman public keys in IEEE Std 1363-2000, to support various schemes defined in this document.

4.4.3 Password-entangled public-key generation primitives (PEPKGP) model

A PEPKGP has the following form:

Input:

— Domain parameters

— A low-grade secret π, typically derived from a password

— A valid private key

Output: The password-entangled public key, or "invalid"

The PEPKGP applies cryptographic operations to the input to generate a *password-entangled public key*. It outputs the password-entangled public key or "invalid" according to the result.

A password-entangled public key is a public key that is additionally derived in part from a password. The password becomes entangled in the process of deriving the high-grade shared key in a way that prevents parties without access to the sensitive password-related data from successfully participating in the method and also prevents them from performing off-line password guessing.

A PEPKGP function *Pepkgp* computes a password-entangled public key based on a password-based value octet string π and a private key integer s – as in $w = Pepkgp(\pi, s)$. Each PEPKGP is designed to make it infeasible for an adversary who chooses w to compute any significant number of pairs of values $\{\pi_i, s_i\}$ that corresponding to plausible passwords, such that each $Pepkgp(\pi_i, s_i)$ produces the same w. The number of $\{\pi_i, s_i\}$ pairs that an adversary can compute for a single w typically represents the number of password guesses that the adversary can make in one run of a PKAS. In the optimal case, only one pair $\{\pi_1, s_1\}$ is known and no alternate pairs can be computed, which may limit the adversary to making just one password guess (π_1) in each run of the scheme.

The application and deployment scenario determines whether a specific number of guesses represents a significant threat.

4.4.4 Secret value derivation primitives (SVDP) model

For the purposes of a PKAS, an SVDP has the following form:

Input:

— Domain parameters

— The party's own private key

— The other party's public key or password-entangled public key

For specific cases only:

— The party's own low-grade secret π

Output: The derived shared secret value, or "invalid"

The SVDP applies cryptographic operations to the input to generate a shared secret value. It outputs the shared secret value as output or it may output "invalid" according to the result.

4.4.5 Password verification data generation primitives (PVDGP) model

A PVDGP has the following form:

Input:

— Domain parameters

— A low-grade secret π, typically derived from a password

Output: Shared secret password verification data and other related secret values associated with the password.

The PVDGP applies cryptographic operations to the input password-based value π to generate and output password verification data and other related values. The value π may also incorporate salt, identifiers, or other values. (See D.2.1.15.) A Server party in an APKAS uses password verification data to verify an associated Client party's knowledge of π. Password verification data values are stored and used by a Server and are considered shared secrets between the Client and Server.

A PVDGP may use a *password-limited private key*, where the sole source of randomness in the key may be limited to that of a password. Password verification data may include *password-limited public keys* that are computed from a password-limited private key using a public-key generation or related exponential method. Password-limited public keys generally need to be kept secret; they should not be confused with true public keys that are designed to be revealed to the public. Both parties share responsibility for maintaining the secrecy of all password verification data.

The PVDGP may also output other secret values associated with the password that shall be kept secret by the Client and are used in APKAS key agreement operations.

4.5 Schemes

This standard focuses on password-authenticated schemes, to specifically leverage the strengths and address the weaknesses of low-grade secrets. The principal difference of a password-authenticated scheme, as compared to a key agreement scheme (KAS) (as defined in IEEE Std 1363-2000), is that one or more of the prearranged secrets that authenticate the parties may be *low-grade* secrets, such as passwords. As such, when passwords are used for mutual authentication, they are typically treated as shared secrets, in contrast with static public and private keys.

The value π is used to refer to a password-based value in descriptions of these methods.

The following types of schemes are defined in this standard:

— Balanced password-authenticated key agreement schemes (BPKAS), in which two parties that share π use ephemeral private keys with π to mutually derive an ephemeral shared key.

— Augmented password-authenticated key agreement schemes (APKAS), in which two parties use ephemeral private keys and corresponding password values, where one party has π and the other party has a one-way function π, to mutually derive an ephemeral shared key.

— Password-authenticated key retrieval schemes (PKRS), in which a Client party establishes one or more static keys using π and the assistance of a Server party that has secret data associated with π.

Of the two kinds of PKAS, balanced and augmented, an APKAS has the distinction that the Server's verification data is a one-way function of the Client's password, so that a thief that knows the verification data cannot masquerade directly as the Client without first cracking the password.

A PKRS differs from a PKAS in that a PKRS establishes *static* keys for the Client that are not necessarily known or derivable by the Server, whereas a PKAS establishes *ephemeral* keys that are shared by both parties. A PKRS may be used in multi-Server systems to limit vulnerabilities due to one or more compromised Servers.

Schemes in this standard (as in IEEE Std 1363-2000) are presented in a general form based on certain primitives and additional methods, such as key derivation methods. For example, a BPKAS is based on a PKGP, an SVDP, and a KDF.

Schemes also include key management operations, such as selecting a private key or obtaining another party's public key. For proper security, a party may need to be assured of the validity of the keys and domain parameters. Generation of domain parameters and keys needs to be performed properly, and in some cases, acceptability tests also need to be performed. Issues of proper key management that may be essential for security and yet are outside the scope of this standard are addressed for informational purposes in Annex D.

The specification of a scheme consists of the following information:

— *Scheme options*, such as choices for primitives and additional methods

— One or more *operations*, depending on the scheme, expressed as a series of *steps*

— *Conformance region recommendations* for implementations conforming with the scheme (see Annex B for more on conformance)

Scheme specifications are functional specifications, not interface specifications. The format of inputs and outputs and the procedure by which an implementation of a scheme is invoked are outside the scope of this standard. See Annex E for more information on input and output formats.

4.6 Additional methods

This standard specifies the following additional methods:

— Hash functions, which are used in many primitives and schemes

— Mask generation functions (MGF), which are used in many primitives and schemes

— Key confirmation functions (KCF), which are used in password authenticated key agreement schemes

— Multiplier value creation functions (MVCF), which are used in APKAS-SRP6.

The specified additional methods are strongly recommended for use in the schemes. The use of an inadequate KCF may compromise the security of the scheme in which it is used. Therefore, a thorough

security analysis should be performed for any implementation that does not use the recommended additional methods for a particular scheme.

4.7 Table summary

This subclause gives a summary of all the schemes in this standard, together with the primitives and additional methods that are invoked within a scheme.

Table 1—Summary of techniques in the standard

Scheme name	Primitives	Additional methods
Balanced password-authenticated key agreement schemes		
{DL,EC}BPKAS-PPK-CLIENT, {DL,EC}BPKAS-PPK-SERVER	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-PAK, and {DL,EC}SVDP-PAK2	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}BPKAS-PAK-CLIENT	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-PAK, and {DL,EC}SVDP-PAK1-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}BPKAS-PAK-SERVER	({DL,EC}REDP-1 or REDP-2), {DL,EC}PKGP-DH, and {DL,EC}SVDP-PAK2	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}BPKAS-SPEKE-CLIENT, {DL,EC}BPKAS-SPEKE-SERVER	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-SPEKE, and {DL,EC}SVDP-SPEKE	KDF1 or KDF2 (each uses a hash function), KCF1
Augmented password-authenticated key agreement schemes		
{DL,EC}APKAS-AMP-CLIENT	{DL,EC}PKGP-DH and {DL,EC}SVDP-AMP-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-AMP-SERVER	{DL,EC}PVDGP-AMP, {DL,EC}PEPKGP-AMP-SERVER, and {DL,EC}SVDP-AMP-SERVER	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-BSPEKE2-CLIENT	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-SPEKE, and {DL,EC}SVDP-SPEKE	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-BSPEKE2-SERVER	{DL,EC}PVDGP-BSPEKE2, ({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-SPEKE, and {DL,EC}SVDP-SPEKE	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-WSPEKE-CLIENT	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-SPEKE, and {DL,EC}SVDP-WSPEKE-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-WSPEKE-SERVER	{DL,EC}PVDGP-BSPEKE2, ({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-SPEKE, and {DL,EC}SVDP-WSPEKE-SERVER	KDF1 or KDF2 (each uses a hash function), KCF1

Table 1— Summary of techniques in the standard *(continued)*

Scheme name	Primitives	Additional methods
{DL,EC}APKAS-PAKZ-CLIENT	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-PAK, and {DL,EC}SVDP-PAK1-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
{DL,EC}APKAS-PAKZ-SERVER	{DL,EC}PVDGP-PAKZ, ({DL,EC}REDP-1 or REDP-2), {DL,EC}PKGP-DH, and {DL,EC}SVDP-PAK2	KDF1 or KDF2 (each uses a hash function), KCF1
DLAPKAS-SRP3-CLIENT	DLPVDGP3-SRP3, DLPKGP-SRP-CLIENT, and DLSVDP-SRP3-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
DLAPKAS-SRP3-SERVER	DLPVDGP-SRP3, DLPEPKGP-SRP3-SERVER, and DLSVDP-SRP3-SERVER	KDF1 or KDF2 (each uses a hash function), KCF1
ECAPKAS-SRP5-CLIENT	ECPVDGP-SRP5, ECPKGP-DH, and ECSVDP-SRP5-CLIENT	KDF1 or KDF2 (each uses a hash function)
ECAPKAS-SRP5-SERVER	ECPVDGP-SRP5, ECPEPKGP-SRP5-SERVER, and ECSVDP-SRP5-SERVER	KDF1 or KDF2 (each uses a hash function), KCF1
Augmented password-authenticated key agreement schemes (continued)		
DLAPKAS-SRP6-CLIENT	DLPVDGP-SRP6, DLPKGP-SRP-CLIENT, and DLSVDP-SRP6-CLIENT	KDF1 or KDF2 (each uses a hash function), KCF1
DLAPKAS-SRP6-SERVER	DLPVDGP-SRP6, DLPEPKGP-SRP6-SERVER, and DLSVDP-SRP6-SERVER	KDF1 or KDF2 (each uses a hash function), KCF1
Password-authenticated Key Retrieval Schemes		
{DL,EC}PKRS-1-CLIENT	({DL,EC}REDP-1 or REDP-2), {DL,EC}PEPKGP-1, and {DL,EC}KRBP-1, KRUP-1	KDF1 or KDF2 (each uses a hash function)
{DL,EC}PKRS-1-SERVER	{DL,EC}KRPP-1	

5. Mathematical conventions

For most of the mathematical conventions, including description of notation, representation of bit strings, octet strings, finite field elements, and data type conversions, the reader is referred to IEEE Std 1363-2000, Clause 5. For convenient reference, an outline of this clause is reproduced here.

Outline of IEEE Std 1363-2000, Clause 5, as amended by IEEE Std 1363a-2004:

5.1 Mathematical notation

5.2 Bit strings and octet strings

5.3 Finite fields

- 5.3.1 Prime finite fields
- 5.3.2 Characteristic two finite fields

5.4 Elliptic curves and points

5.5 Data type conversion

- 5.5.1 Converting between integers and bit strings (I2BSP and BS2IP)
- 5.5.2 Converting between bit strings and octet strings (BS2OSP and OS2BSP)
- 5.5.3 Converting between integers and octet strings (I2OSP and OS2IP)

5.5.4 Converting between finite field elements and octet strings (FE2OSP and OS2FEP)

5.5.5 Converting finite field elements to integers (FE2IP)

5.5.6 Converting between elliptic curve points and octet strings

Exceptions and additions to these conventions are noted here.

5.1 Mathematical notation

5.1.1 Group notation

To unify the presentation of methods that work in multiple settings, this document provides generalized descriptions for methods that may use either the *DL setting* (6.1) or the *EC setting* (7.1). In these generalized descriptions, the convention is to use lowercase letters to represent DL group elements, and to generally use the same notation as that used in IEEE Std 1363-2000 in the DL setting.

Although group operations and group elements are instantiated differently in the DL and EC settings, the following * and ^ operators are used to describe operations in both settings:

— e_1*e_2 denotes the group operator applied to group elements e_1 and e_2, and

— *e*^*n* denotes the iterated group operation *e***e**...**e* on *n* values of group element *e*.

These operators assume that *n* is an integer and *e*, e_1, and e_2 are elements of an appropriate group defined by either DL or EC domain parameters. (See 6.1.1, 7.1.1, and 8.1.1 for the DL, EC, and generalized group settings, respectively.)

NOTE 1—As in IEEE Std 1363-2000, this document uses $a \times b$ to denote multiplication of field elements *a* and *b*, multiplication of integers *a* and *b*, and scalar multiplication of an EC point *b* by integer *a*. Similarly, the × symbol may be omitted when such omission (*ab*) does not cause ambiguity. This document uses the notation exp(*w*, *c*), to denote raising DL field element *w* to the power of integer *c*. The notation w^c denotes raising integer *w* to the power of integer *c*.

NOTE 2—As in IEEE Std 1363-2000, throughout this document, operations on finite field elements, group elements, and integers are used. Care needs to be exercised to distinguish between integers, field elements, and group elements, especially as operations on integers and field elements are sometimes denoted by the same symbols. See IEEE Std 1363-2000, 5.3, for more information on finite fields.

NOTE 3—Further discussion of the analogous relationship between the DL and EC settings can be found in IEEE Std 1363-2000, A.9.4.

5.2 Bit strings and octet strings

5.2.1 Single octet values

The notation *hex*(*nn*) represents a single octet with hexadecimal value *nn*, where *n* is a hexadecimal digit.

5.2.2 Empty octet string

The notation "" represents the empty octet string of length zero.

5.3 Data type conversion

5.3.1 Converting integers to field elements (I2FEP)

In performing cryptographic operations, non-negative integers sometimes need to be converted to finite field elements. The primitive that performs this operation is called the *integer to field element conversion primitive,* or I2FEP.

An integer i in the range [0, q−1] shall be converted to an element j of a finite field $GF(q)$ by the following or an equivalent procedure:

a) Convert i to an octet string of length $\lceil \log_{256}(q) \rceil$ using I2OSP.

b) Convert the resulting octet string to a field element j using OS2FEP.

c) Output j.

5.3.2 Converting group elements to secret value field elements (GE2SVFEP)

This document uses the *group element to secret value field element primitive* (GE2SVFEP) to convert group elements into field elements. This primitive is defined in both the DL and EC settings.

The GE2SVFEP function shall convert a group element e to a secret value field element by the following, or an equivalent, procedure:

a) In the DL setting:

   1) Output the field element e. (In the DL setting, group elements are field elements.)

b) In the EC setting:

   1) If e is the identity element (the point at infinity ○), output field element 0 and stop.

   2) Set field element x_e = the x-coordinate of EC point e.

   3) Output field element x_e.

NOTE 1—There is no corresponding inverse function (like "SVFE2GEP") because there is a loss of information in the conversion process from a group element to a field element in the EC setting; the y coordinate is omitted. The GE2SVFEP function is used only for the specific case of deriving the final output of an SVDP and helps to unify the generalized description of primitives that work in either a DL or EC setting.

NOTE 2—The conversion for the EC group identity element is arbitrary and may duplicate the result of a non-identity group element at (0,y). Methods that abort for unacceptable values generally do so before invoking GE2SVFEP or may reject all 0 results under the assumption that the probability of obtaining a valid 0 result is negligible.

5.3.3 Converting group elements to octet strings (GE2OSP-X)

This document uses the Group Element to Octet String Secret primitive (GE2OSP-X) to convert group elements into octet strings. This primitive is defined in both the DL and EC settings.

The GE2OSP-X function shall convert a group element e to an octet string by the following, or an equivalent, procedure:

a) In the DL setting:

1) Compute octet string o_e = FE2OSP(e) (In the DL setting, group elements are field elements.)
2) Output octet string o_e

b) In the EC setting:

1) Compute octet string o_e = EC2OSP-X(e) (See IEEE Std 1363a-2004, 5.5.6.3.)
2) Output octet string o_e

NOTE 1—There is no corresponding inverse function (like "OS2GEP") because there is a loss of information in the conversion process from a group element to an octet string in the EC setting; the y coordinate is omitted.

NOTE 2—In the EC setting, GE2OSP-X(e) is *not* equivalent to FE2OSP(GE2SVFEP(e)), due to the initial format octet provided by EC2OSP-X.

6. The DL setting

This clause specifies the DL setting for the family of cryptographic primitives based on the discrete logarithm problem over multiplicative groups in finite fields, also known as the *DL family*. The reader is referred to IEEE Std 1363-2000, Clause 6, for more information, an outline of which is presented here.

Outline of IEEE Std 1363-2000, Clause 6:

6.1 The DL setting
- 6.1.1 Notation
- 6.1.2 DL domain parameters

6.2 Primitives
- 6.2.1 DLSVDP-DH
- 6.2.2 DLSVDP-DHC

6.1 The DL setting

6.1.1 Notation

The notation used in the DL setting as defined in IEEE Std 1363-2000, 6.1.1, applies to this document.

6.1.2 DL domain parameters

The definition of *DL domain parameters* from IEEE Std 1363a-2004, 6.1.2, applies to this document, with some modifications noted here.

See 8.1.2 for the generalized definition of domain parameters for both the DL and EC settings. In the DL setting, the *parent group* is the multiplicative group of all ($k \times r$) non-zero field elements in $GF(q)$ and r divides ($q - 1$). For most methods in the DL setting, the group generated by g is referred to as the *desired group,* and *public keys* are generally described as group elements. See 8.1.2.1 for the exceptions of SRP3 and SRP6, which operate in the parent group, and where public keys may also be described as DL field elements.

6.1.3 DL key pairs

The definitions of *DL key pair*, *DL private key*, *DL public key*, *valid DL private key*, and *valid DL public key* for most of the methods defined in this document are in IEEE Std 1363-2000, 6.1, and apply to this document. See 8.1.3 for the generalized definition of key pairs.

7. The EC setting

This clause specifies the EC setting for the family of cryptographic primitives based on the discrete logarithm problem over elliptic curve groups in finite fields, also known as the *EC family*. The reader is referred to IEEE Std 1363-2000, Clause 7, for more information, an outline of which is presented here.

Outline of IEEE Std 1363-2000, Clause 7:

7.1 The EC setting

7.1.1 Notation

When EC operations are specified in terms of elliptic curve points, the same notation is used as in IEEE Std 1363-2000. That is, $P + Q$ denotes point addition of points P and Q, and $n \times P$ (or simply $n\,P$, when omission of $\times$ does not cause ambiguity) denotes scalar point multiplication of point P by integer n.

7.1.2 EC domain parameters

The definition of *EC domain parameters* from IEEE Std 1363a-2004, 7.1.2, applies to this document.

See 8.1.2 for the generalized definition of domain parameters for both the DL and EC settings. In the EC setting, the *parent group* is the group of points on the elliptic curve E defined by the two EC domain parameters coefficients a and b [elements of $GF(q)$], the parent group order r divides $\#E$ (the number of points on E), and cofactor $k = \#E / r$. For all methods in the EC setting, the desired group is the group of r group elements generated by g, where g is a synonym for the EC generator point G, and *public keys* are generally described as group elements.

7.1.3 EC key pairs

The definitions of *EC key pair*, *EC private key*, *EC public key*, *valid EC private key*, and *valid EC public key* are in IEEE Std 1363-2000, 7.1, and apply to this document. See 8.1.3 for the generalized definition of key pairs.

8. Primitives

This clause specifies cryptographic primitives for use with schemes in multiple families of underlying groups. Many of the primitives in this clause are defined in a general manner for both the DL setting (6.1) and the EC setting (7.1).

8.1 Notation and definitions

8.1.1 The generalized group setting

A generalized group setting notation and definitions are described in this clause for the primitives in Clause 8 and the schemes in Clause 9 and Clause 10. These methods may use either DL or EC settings, where * denotes the group operator and ^ denotes exponentiation for the appropriate group, as described in 5.1.1.

8.1.2 Generalized domain parameters

A generalized definition of the domain parameters q, k, r, and g for both the DL and EC settings is presented here. Further detail for the DL and EC settings is presented in 6.1.2 and 7.1.2, respectively.

Domain parameters are used in every primitive and scheme and are an implicit component of every key. A set of domain parameters in either the DL or EC setting specifies the following:

— A field $GF(q)$, where q is a positive odd prime integer p, 2^m for some positive integer m, or p^m for an odd prime p and some integer $m \geq 2$

— A large prime integer r designating the order of the *desired group*

— A *parent group* of $(k \times r)$ group elements, where k is designated the cofactor

— A group element g of order r that generates the desired group (g is a *generator* of the desired group)

If $q = 2^m$, the set of domain parameters also specifies a representation for the elements of $GF(q)$ to be used by the conversion primitives (see 5.3 and IEEE Std 1363-2000, 5.3.2). If EC key validation is to be performed, or if the SVDP-DHC or SVDP-MQVC primitive is to be applied then it shall also be the case that GCD(k, r) = 1 (i.e., r does not divide k; see IEEE Std 1363a-2004, A.1.1). For all methods other than SRP3 and SRP6, the group generated by g is referred to as the *desired group*. Members of the desired group may be referred to as *group elements*, and *public keys* are generally described as group elements.

8.1.2.1 Domain parameters for SRP3 and SRP6

For the SRP3 and SRP6 methods, the *desired group* is the parent group, which is generated by an additional domain parameter element g_{q-1} of order $(k \times r)$, instead of the element g of order r. The SRP3 and SRP6 methods are described only for this modified DL setting.

NOTE—The definition of *valid {DL,EC} domain parameters* is in IEEE Std 1363-2000, 6.1.2 (for DL) and 7.1.2 (for EC), and generally applies to this document, with the exception that SRP3 and SRP6 require the additional domain parameter g_{q-1}. See D.2.2.4.2.

8.1.3 Key pairs

The definitions of *key pair*, *private key*, *public key*, *valid private key*, and *valid public key* for most of the methods defined in this document are in IEEE Std 1363-2000, 6.1 (DL) and 7.1 (EC), and apply to this document. For convenience, these definitions are restated here.

For all methods other than the SRP3 and SRP6 methods:

— A *valid private key* (designated as s) is an integer in the range [1, r−1]

— A *valid public key* is a group element of order r in the desired group generated by g

— The public key w that corresponds to s is computed as $w = g^{\wedge}s$

The qualified term *public-key value* indicates a value that represents a valid public key in an interaction solely between legitimate compliant parties, but may represent an invalid public key in a threat or error scenario.

8.1.3.1 Key pairs for SRP3 and SRP6

For the SRP3 and SRP6 methods:

— A *valid private key* (designated as s) is an integer in the range [1, kr−1]

— A *valid public key* is a field element in $GF(q)$ other than the additive identity element (0) or the multiplicative identity element (1)

— The public key w that corresponds to s is computed as $w = g_{q-1}{}^{\wedge}s$.

NOTE 1—The concept of *valid public key* is not particularly relevant to SRP3 and SRP6, since these methods incorporate other necessary acceptability tests for received public-key values.

NOTE 2—SRP3 and SRP6 public keys are generally described in terms of DL field elements, instead of group elements.

8.1.4 Password-entangled public key

A *password entangled public key* is a public key that is derived from a private key and a password-based value using a *password-entangled public key generation primitive* (PEPKGP). This document defines various PEPKGP functions that are designed for use by specific schemes.

8.1.5 Acceptable (password-entangled) public-key values

Some methods in this document use the concept of an *acceptable* public-key value, which may be a *password-entangled* public-key value, where the definition of *acceptable* may depend on the specific method. The requirements for an acceptable public-key value are generally looser than for a valid public key and may allow for the use of simpler or more efficient acceptability tests. For example, in some of the PAK methods, an acceptable password-entangled public-key value is any element of the parent group [of order ($k \times r$)], whereas in some of the SPEKE methods, an acceptable password-entangled public-key value generates a sufficiently large order subgroup of the parent group.

Note that the identity element and other small order elements of the parent group may be treated as special cases in these schemes, primitives, and validation techniques, and that the definition of *sufficiently large* may depend on both the domain parameters and the desired level of security to be achieved.

8.1.5.1 Public-key tests

Algorithms for determining whether key values are acceptable for specific PKAS methods are described in A.1.

General algorithms to verify that a DL public-key value is valid and to verify that a value is in the DL parent group are given in IEEE Std 1363-2000, A.16.6. General algorithms to verify that an EC public-key value is valid and to verify that a value is in the EC parent group are given in IEEE Std 1363-2000, A.16.10.

Updated discussion of further key validation issues is in IEEE Std 1363a-2004, D.5.1.6.

8.1.6 Summary of terms

This subclause lists terms used in Clause 8, Clause 9, and Clause 10. It is meant as a reference guide only; for complete definitions of the terms listed, refer to the appropriate text. Some other terms are also used occasionally; they are introduced in the text where appropriate.

π	A password-based octet string, used for authentication. π is generally derived from a password or a hashed password, and may incorporate a salt value, identifiers for one or more parties, and/or other shared data.
π_m	Password-based group element, derived from a function of π
π_R, π_T	Password-based group elements, derived from a function of π, respectively associated with received and transmitted values
b_1	A Boolean value, indicating whether cofactor multiplication is desired
C_S	A Server's key confirmation value octet string
C_C	A Client's key confirmation value octet string
e,e_1, e_2, ...	Group elements
g	A group element of order r in the desired group (part of the domain parameters for most schemes and primitives)
g_π	A group element of order r in the desired group, derived from a function of π
g_a, g_b	Group elements of order r in the desired group (parameters for REDP-2)
$GF(q)$	The Galois field of order q, which may be a prime field GF(p), a binary field $GF(2^m)$, or an odd characteristic extension field $GF(p^m)$ (see IEEE Std 1363-2000, A.1.2 and A.3.1, and IEEE Std 1363a-2004, A.17)
g_{q-1}	A field element of multiplicative order $q-1$ in $GF(q)$ (part of the DL domain parameters for the SRP3 and SRP6 schemes and primitives)
HashWC	A hash function used on a Client's public key in APKAS-AMP
HashWS	A hash function used on a Server's public key in APKAS-WSPEKE
HashW2	A hash function used on public keys in APKAS-SRP5 and APKAS-SRP6
HashKC	A hash function used in key confirmation
HashPVD	A hash function used in password verification data generation
HashRE	A hash function used in random element derivation
HashSK	A hash function used on signature private keys used in APKAS-PAKZ

H_u	An octet string representing the hash of a signature private key *u* used in APKAS-PAKZ
i_1, i_2, ...	Integer values
k	The integer cofactor value equal to the order of the parent group divided by *r* (part of the domain parameters)
K_1, K_2, ...	Shared secret keys, octet strings derived in a key agreement scheme
Kdf	A key derivation function
M	A message octet string used in APKAS-PAKZ
Mgf	A mask generation function
m_v	A field element multiplier value used in APKAS-SRP6
Mvcf	A multiplier value creation function used to compute m_v in APKAS-SRP6
o_{ID}	An octet string parameter used in APKAS-AMP and APKAS-PAKZ
o_π	A selector value octet string derived from a function of π
o_1, o_2, ...	Octet string values
oLen	Integer length in octets of the output of a hash function
o_u	An octet string representing private signature key *u* used in APKAS-PAKZ
P_1, P_2, ...	Key derivation parameter octet strings
q	The size of the field used (part of the domain parameters)
r	The prime order of the desired group for most methods in this document (part of the domain parameters)
Redp	An REDP
s	Private key, integer, corresponding to public key *w*
S_C	A signature used in APKAS-PAKZ key confirmation
Sign	A signature generation operation in signature scheme *Ss*
Ss	A signature scheme used in APKAS-PAKZ
$U_{K\pi}$	An octet string representing a hidden and password-masked private signature key *u* used in APKAS-PAKZ key confirmation
U_π	An octet string representing a password-masked private signature key *u* used in APKAS-PAKZ
u	Private key, integer, corresponding to public key *v*
u_π	A password-limited private key corresponding to v_π, derived from a function of π
uLen	The length in octets of an agreed octet string representation of a private key *u* in signature scheme *Ss*
v	Public key corresponding to private key *u*
v_π	A password-limited public key corresponding to u_π, used as password verification data
Verify	A signature verification operation in signature scheme *Ss*
w	Public key corresponding to private key *s*
w'	Another party's public key
w_C	Client's public key

w_S	Server's public key
x	Field element
y	Field element
z	Shared secret value, field element in $GF(q)$, derived by SVDPs
Z, Z_1, Z_2	Shared secret values, octet strings, in key agreement operations
z_1, z_2	Shared secret field elements
z_g	Established shared secret group element in a PKAS; established secret group element in a PKRS

8.2 Primitives

Subclause 4.4 describes the types of primitives defined in this document. This subclause defines primitives for use in both the DL and EC settings.

8.2.1 KRBP-1

{DL,EC}KRBP-1 is {discrete logarithm, elliptic curve} key retrieval blinding primitive, version 1. KRBP-1 is based on the work of Ford and Kaliski [B16]. It derives a blinded password from a password-derived generator and the Client's private key, using {DL,EC} domain parameters. This primitive is used by the scheme {DL,EC}PKRS-1-CLIENT.

Input:

— The Client's private key integer s

— The Client's password-derived generator group element g_π

— The {DL,EC} domain parameters (including g and r) associated with s and g_π

Assumptions: Private key s and associated domain parameters are valid. g_π is a generator of the desired group of order r.

Output: The computed blinded password w_C, a valid public key.

Operation: The blinded password w_C shall be computed by the following or an equivalent sequence of steps:

a) Compute group element $w_C = g_\pi \wedge s$

b) Output w_C as the blinded password

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—KRBP-1 performs the inverse function of KRUP-1.

8.2.2 KRPP-1

{DL,EC}KRPP-1 is {discrete logarithm, elliptic curve} key retrieval permutation primitive, version 1. KRPP-1 is based on the work of Ford and Kaliski [B16]. This primitive derives a permuted blinded password from a party's private key and a blinded password using {DL,EC} domain parameters. This primitive is used by the scheme {DL,EC}PKRS-1-SERVER.

Input:

— The Server's own private key integer u

— The Client's blinded password group element w_C

— The {DL,EC} domain parameters (including r) associated with the keys u and w_C

Assumptions: Private key u and associated domain parameters are valid. w_C is an element of the parent group.

Output: The computed permuted blinded password w_S, a valid public key.

Operation: The permuted blinded password w_S shall be computed by the following or an equivalent sequence of steps:

a) Compute group element $w_S = w_C \wedge u$

b) Output w_S as the permuted blinded password.

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.3 KRUP-1

{DL,EC}KRUP-1 is {discrete logarithm, elliptic curve} key retrieval unblinding primitive, version 1. KRUP-1 is based on the work of Ford and Kaliski [B16]. This primitive derives a permuted password value from both the Client's own private key and a Server's blinded permuted password value using {DL,EC} domain parameters. This primitive is used by the scheme {DL,EC}PKRS-1-CLIENT.

Input:

— The Client's own private key integer s

— The Server's permuted blinded password group element w_S

— The {DL,EC} domain parameters (including r) associated with the keys s and w_S

Assumptions: Private key s and associated domain parameters are valid. w_S is an element of the parent group.

Output: The permuted password value z_g, an element of the parent group.

Operation: The permuted password value z_g shall be computed by the following or an equivalent sequence of steps:

a) Compute group element $z_g = w_S \wedge (s^{-1} \bmod r)$

b) Output z_g as the permuted password.

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—KRUP-1 performs the inverse function of KRBP-1.

8.2.4 PEPKGP-AMP-SERVER

{DL,EC}PEPKGP-AMP-SERVER is {discrete logarithm, elliptic curve} password-entangled public-key generation primitive, version AMP for Server. PEPKGP-AMP-SERVER is based on the work of Kwon [B38]. This primitive derives a password-entangled public key from password verification data, the Server's private key, and a Client's public key, using {DL,EC} domain parameters.

This primitive is parameterized by the following choices:

— A hash function *HashWC* (see NOTE), which should be one of the hash functions in 12.1.

— An octet string o_{ID} that may provide additional input to *HashWC*.

The Client and Server shall use the same *HashWC* and o_{ID} parameters with PEPKGP-AMP-SERVER and SVDP-AMP-CLIENT.

Input:

— The Server's private key s

— The password verification data v_π, a password-limited public key

— The Client's public key w_C

— The {DL,EC} domain parameters associated with the keys s, v_π, and w_C

Assumptions: Private key s, password-limited public key v_π, and associated domain parameters are valid; w_C is an element of the parent group.

Output: The derived password-entangled public key value w_S, which is an element of the parent group

Operation: The password-entangled public key value w_S shall be computed by the following or an equivalent sequence of steps:

a) Compute o_C = GE2OSP-X(w_C)

b) Compute o_1 = *HashWC*(o_C || o_{ID})

c) Compute i_1 = OS2IP(o_1)

d) Compute $w_S = ((w_C \wedge i_1) * v_\pi) \wedge s$

e) Output w_S

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.1.6 for criteria for selecting *HashWC*.

8.2.5 PEPKGP-PAK

{DL,EC}PEPKGP-PAK is {discrete logarithm, elliptic curve} password-entangled public-key generation primitive, version PAK. It is based on the work of Boyko, MacKenzie, Patel [B8] and MacKenzie [B43]. This primitive derives a password-entangled public key from the party's private key and password value, using {DL,EC} domain parameters. This primitive is used by the schemes {DL,EC}BPKAS-PAK-CLIENT, {DL,EC}BPKAS-PPK-{CLIENT,SERVER} and {DL,EC}APKAS-PAKZ-CLIENT.

Input:

— The party's own private key s.

— The password-based mask group element π_m

— The {DL,EC} domain parameters (including g and r) associated with key s and group element π_m

Assumptions: Private key s and domain parameters are valid; group element π_m generates the desired group of order r.

Output: The derived password-entangled public key w

Operation: The password-entangled public key w shall be computed by the following or an equivalent sequence of steps:

a) Compute a group element $w = (g \wedge s) * \pi_m$

b) Output w as the password-entangled public key

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.6 PEPKGP-SPEKE

{DL,EC}PEPKGP-SPEKE is {discrete logarithm, elliptic curve} password-entangled public-key generation primitive, version SPEKE. It is based on the work of Jablon [B31]. This primitive derives a password-entangled public key from the party's private key and password value. This primitive is used in schemes BPKAS-SPEKE and APKAS-BSPEKE2.

Input:

— A private key integer s

— A password-derived generator group element g_π

— The {DL,EC} domain parameters associated with g_π and s

Assumptions: Private key s and associated input domain parameters are valid. g_π generates the desired group of order r.

Output: The derived password-entangled public key w

Operation: The public key w shall be computed by the following or an equivalent sequence of steps:

a) Compute $w = g_\pi \wedge s$

b) Output w

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.7 [DL] PEPKGP-SRP3-SERVER, PEPKGP-SRP6-SERVER

DLPEPKGP-{SRP3,SRP6}-SERVER is discrete logarithm password-entangled public-key generation primitive, version {SRP3,SRP6} for Server. This primitive derives a password-entangled public key from the Server's private key and password verification data using DL domain parameters. This primitive is used in DLAPKAS-{SRP3,SRP6}-SERVER. APKAS-SRP3 is based on the work of Wu [B57] and IETF RFC 2945 [B24]. APKAS-SRP6 is based on the work of Wu [B57] and [B56].

For SRP6 only: This primitive is parameterized by the following choices:

— A multiplier value creation function *Mvcf*, which should be MVCF-DP (see 12.5.1). The Client shall use the same *Mvcf* and its associated parameters with DLSVDP-SRP6-CLIENT.

Input:

— The Server's own private key s

— The password verification data v_π, a password-limited public key

— The DL domain parameters (including g_{q-1} and q) associated with the key s and verification data v_π

Assumptions: Private key s, password-limited public key v_π, and DL domain parameters are valid; g_{q-1} is of order $q-1$.

Output: The computed password-entangled public-key value w_S, which is a field element in $GF(q)$

Operation: The password-entangled public-key value w_S shall be computed by the following or an equivalent sequence of steps:

a) Compute field element $x = \exp(g_{q-1}, s)$

b) Compute field element m_v as follows:

1) *For SRP3 only*: Let $m_v = 1$

2) *For SRP6 only*: Compute m_v using *Mvcf*

c) Compute $w_S = (v_\pi \times m_v + x)$

d) Output w_S

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.8 [EC] PEPKGP-SRP5-SERVER

ECPEPKGP-SRP5-SERVER is elliptic curve password-entangled public-key generation primitive, version SRP5. SRP5 is based on the work of Wang [B55] and Wu [B57]. This primitive derives a password-entangled public key from the Server's private key and password verification data, using EC domain parameters. This primitive is used by the scheme ECAPKAS-SRP5-SERVER.

This primitive is parameterized by the following choice:

— A REDP function *Redp*, which should be ECREDP-1 or ECREDP-2. The Client shall use the same *Redp* parameter with SVDP-SRP5-CLIENT.

Input:

— The Server's own private key s

— The password verification data v_π, a password-limited public key

— The EC domain parameters (including g) associated with key s and public key v_π

Assumptions: Private key s, public key v_π, and associated domain parameters are valid.

Output: The derived password-entangled public key w_S

Operation: The password-entangled public key w_S shall be computed by the following or an equivalent sequence of steps:

a) Compute octet string o_1 = GE2OSP-X(v_π)

b) Compute group element $e_1 = Redp(o_1)$

c) Compute group element $w_S = (g \wedge s) * e_1$

d) Output w_S as the password-entangled public key

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.9 PKGP-DH

{DL,EC}PKGP-DH is {discrete logarithm, elliptic curve} public-key generation primitive, Diffie-Hellman version. PKGP-DH is based on the work of Diffie and Hellman [B10] and IEEE Std 1363-2000. This primitive derives a {DL,EC} public key from the party's {DL,EC} private key using {DL,EC} domain parameters. This primitive is used by the schemes {DL,EC}BPKAS-PAK-SERVER, {DL,EC}APKAS-AMP-CLIENT and ECAPKAS-SRP5-CLIENT.

Input:

— The party's own private key integer s

— The {DL,EC} domain parameters (including g and r) associated with the key s

Assumptions: Private key s and associated domain parameters are valid.

Output: The computed public key w

Operation: The public key w shall be computed by the following or an equivalent sequence of steps:

a) Compute public-key group element $w = g \wedge s$

b) Output w as the public key

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.10 [DL] PKGP-SRP-CLIENT

DLPKGP-SRP-CLIENT is discrete logarithm public-key generation primitive, version SRP for Client. It is based on the work of Wu [B57] and IETF RFC 2945 [B24]. This primitive derives a public key from the Client's private key using DL domain parameters. This primitive is used by the schemes DLAPKAS-{SRP3,SRP6}-CLIENT.

Input:

— The Client's own private key integer s

— The DL domain parameters (including q and g_{q-1}) associated with the key s

Assumptions: Private key s and DL domain parameters are valid; g_{q-1} is of order $q-1$.

Output: The computed public key w_C, which is a non-zero field element of $GF(q)$

Operation: The public key w_C shall be computed by the following or an equivalent sequence of steps:

a) Compute a DL public-key field element $w_C = \exp(g_{q-1}, s)$

b) Output w_C

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—DLPKGP-SRP-CLIENT is similar to DLPKGP-DH, except that DLPKGP-DH uses the generator g of order r, instead of g_{q-1}.

8.2.11 PVDGP-AMP

{DL,EC}PVDGP-AMP is {discrete logarithm, elliptic curve} password verification data generation primitive, version AMP. PVDGP-AMP is based on the work of Kwon [B38]. This primitive derives password verification data and associated values from a party's password value, using {DL,EC} domain parameters. This primitive derives the password-limited private key used in {DL,EC}APKAS-AMP-CLIENT and derives the password verification data used in {DL,EC}APKAS-AMP-SERVER.

This primitive is parameterized by the following choices:

— A hash function *HashPVD* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The password-based octet string π

— The {DL,EC} domain parameters (including g and r) associated with π

Assumptions: Domain parameters are valid.

Output: The password-limited private key u_π and associated password verification data consisting of password-limited public key v_π

Operation: The password-limited private key u_π and verification data v_π shall be computed by the following or an equivalent sequence of steps:

a) Compute $o_\pi = HashPVD(\pi)$

b) Compute $u_\pi = \text{OS2IP}(o_\pi) \bmod r$

c) Compute $v_\pi = g \wedge u_\pi$

d) Output u_π and verification data v_π

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.1.17 for discussion of appropriate choices for *HashPVD*.

8.2.12 PVDGP-BSPEKE2

{DL,EC}PVDGP-BSPEKE2 is {discrete logarithm, elliptic curve} password verification data generation primitive, version BSPEKE2. It is based on the work of Jablon [B27] and [B28]. This primitive derives password verification data and associated values from a party's password, using {DL,EC} domain parameters. This primitive derives the password-limited private key used in {DL,EC}APKAS-BSPEKE2-CLIENT and {DL,EC}APKAS-WSPEKE-CLIENT and derives the associated password verification data used in {DL,EC}BPKAS-BSPEKE2-SERVER and {DL,EC}BPKAS-WSPEKE-SERVER.

This primitive is parameterized by the following choices:

— A REDP function *Redp*, which should be {DL,EC}REDP-1 or {DL,EC}REDP-2

— A hash function *HashPVD* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The password-based octet string π

— The {DL,EC} domain parameters associated with π

Assumptions: Domain parameters are valid.

Output: The password-limited private key u_π and the password verification data consisting of generator element g_π of order r and password-limited public key v_π

Operation: The password-limited private key u_π and password verification data (g_π, v_π) shall be computed by the following or an equivalent sequence of steps:

a) Compute an octet string $o_\pi = HashPVD(\pi)$

b) Compute a private key integer u_π = OS2IP(o_π) mod r

c) Compute a group element $g_\pi = Redp(o_\pi)$

d) Compute a group element $v_\pi = g_\pi \wedge u_\pi$

e) Output u_π and password verification data (g_π, v_π)

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.1.17 for discussion of appropriate choices for *HashPVD*.

8.2.13 PVDGP-PAKZ

{DL,EC}PVDGP-PAKZ is {discrete logarithm, elliptic curve} password verification data generation primitive, version PAKZ. PVDGP-PAKZ is based on the work of MacKenzie [B43] and Gentry, MacKenzie, Ramzan [B18]. This primitive derives password verification data from a party's password, using {DL,EC} domain parameters. It derives the password verification data used in {DL,EC}APKAS-PAKZ-SERVER.

This primitive is parameterized by the following choices:

— A signature scheme *Ss*, which is the signature scheme option of APKAS-PAKZ that is associated with a private key *u*, a corresponding public key *v*, and an agreed format for representing *u* as a string of *uLen* octets.

— A REDP function *Redp*, which should be ECREDP-1 or ECREDP-2.

— A mask generation function *Mgf*, which is the MGF scheme option of APKAS-PAKZ.

— A hash function *HashSK*, which is the hash function scheme option of APKAS-PAKZ.

Input:

— The password-based octet string π

— The {DL,EC} domain parameters associated with π

Assumptions: Domain parameters are valid.

Output: The derived password verification data (π_m, v, U_π, H_u), consisting of password-based mask group element π_m, public key v for signature scheme *Ss*, octet string U_π of length *uLen* containing the password-masked private key associated with v, and hashed private key octet string H_u (See NOTE 2 and NOTE 3.)

Operation: The password verification data shall be computed by the following or an equivalent sequence of steps:

a) Compute password-mask group element $\pi_m = Redp(hex(01) \| \pi)$ (See NOTE 1.)

b) Compute an octet string $o_1 = hex(02) \| \pi$

c) Compute a password-mask octet string o_2 of length *uLen* as $o_2 = Mgf(o_1, uLen)$

d) Generate a key pair (u,v) for signature scheme *Ss*

e) Compute octet string o_u of length *uLen* as the agreed octet string representation of private key u

f) Compute password-masked private key octet string $U_\pi = o_u \oplus o_2$

g) Compute a hashed private key octet string $H_u = HashSK(o_u)$

h) Output (π_m, v, U_π, H_u) (See NOTE 2.)

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—Steps b) through i) of the BPKAS-PAK key agreement for Client operation and steps b) through i) of the BPKAS-PAK key agreement for Server operation are reused in APKAS-PAKZ, with the value of π_m as computed here. The computation of π_m is different in BPKAS-PAK.

NOTE 2—Output values π_m, U_π, and H_u shall be kept secret. Output value v may be publicly disclosed.

NOTE 3—The APKAS-PAKZ Server may want to store π_m^(−1) to optimize efficiency.

8.2.14 [DL] PVDGP-SRP3, PVDGP-SRP6

DLPVDGP-{SRP3,SRP6} is discrete logarithm password verification data generation primitive, version {SRP3,SRP6}. This primitive derives password verification data and an associated value from a party's password value, using DL domain parameters. This primitive derives the password-limited private key used in DLAPKAS-{SRP3,SRP6}-CLIENT and the associated password verification data used in DLAPKAS-{SRP3,SRP6}-SERVER. APKAS-SRP3 is based on the work of Wu [B57] and IETF RFC 2945 [B24]. APKAS-SRP6 is based on the work of Wu [B57] and [B56].

For SRP6 only: This primitive is parameterized by the following choices:

— A hash function *HashPVD* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The password-based octet string π

— The DL domain parameters (including q and g_{q-1}) associated with π

Assumptions: DL domain parameters are valid; g_{q-1} is of order $q-1$.

Output: The password-limited private key u_π and associated password verification data v_π, a password-limited public key

Operation: The password-limited private key u_π and verification data v_π shall be computed by the following or an equivalent sequence of steps:

a) Compute an octet string o_π as follows:

   1) *For SRP3 only*: Compute o_π = SHA-1(π)

   2) *For SRP6 only*: Compute o_π = *HashPVD*(π)

b) Compute an integer DL private key u_π = OS2IP(o_π) mod (q−1)

c) Compute the corresponding password-limited public key $v_\pi = \exp(g_{q-1}, u_\pi)$

d) Output u_π and password verification data v_π

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.1.17 for discussion of appropriate choices for *HashPVD*.

8.2.15 [EC] PVDGP-SRP5

ECPVDGP-SRP5 is elliptic curve password verification data generation primitive, version SRP5. ECPVDGP-SRP5 is based on the work of Wang [B55] and Wu [B57]. This primitive derives password verification data and associated values from a party's password value, using EC domain parameters. This primitive derives the password-limited private key used in ECAPKAS-SRP5-CLIENT and derives the password verification data used in ECAPKAS-SRP5-SERVER.

This primitive is parameterized by the following choices:

— A hash function *HashPVD* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The password-based octet string π

— The EC domain parameters (including g and r) associated with π

Assumptions: Domain parameters are valid.

Output: The password-limited private key u_π and associated password verification data consisting of password-limited public key v_π

Operation: The password-limited private key u_π and verification data v_π shall be computed by the following or an equivalent sequence of steps:

a) Compute o_π = *HashPVD*(π)

b) Compute u_π = OS2IP(o_π) mod r

c) Compute $v_\pi = g \wedge u_\pi$

d) Output u_π and verification data v_π

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.1.17 for discussion of appropriate choices for *HashPVD*.

8.2.16 [DL] REDP-1

DLREDP-1 is discrete logarithm random element derivation primitive, version 1. It is based on Jablon [B28] and the concept of using a hash function to ensure valid construction in Vaudenay [B53]. The primitive uses a hash function of a password-based input selector value to select a pseudo-random element of a group to be used in a DL PKAS, in a way that prevents collisions and obscures exponential relationships of output values (see 4.4.1). DLREDP-1 may be used to construct a generator value in the schemes DLBPKAS-SPEKE, DLAPKAS-BSPEKE2, DLAPKAS-WSPEKE, and DLPKRS-1 and to construct a mask value in the schemes DLBPKAS-PAK, DLBPKAS-PPK, and DLAPKAS-PAKZ.

This primitive is parameterized by the following choices:

— A hash function *HashRE* (see NOTE 4), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The random selector value o_π, which is an octet string derived from a password value

— The DL domain parameters (including q and k) associated with o_π

Assumptions: DL domain parameters are valid.

Output: The selected group element e, which is a generator of the desired group of order r, or "invalid."

Operation: The selected group element is computed by the following or an equivalent sequence of steps:

a) Compute an octet string $o_1 = HashRE(o_\pi)$
b) Compute a field element x = I2FEP(OS2IP(o_1) mod q)
    1) If $x = 0$, output "invalid" and stop. (See NOTE 5.)
c) Compute the field element $e = \exp(x, k)$
    1) If $e = 1$, output "invalid" and stop. (See NOTE 5.)
d) Output e as the selected group element

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—DLREDP-1 in the DL setting is analogous to ECREDP-1 in the EC setting.

NOTE 2—DLREDP-2 is described in 8.2.18 *REDP-2*.

NOTE 3—Subclause D.2.1.20 describes the security properties of REDP methods in relation to various schemes.

NOTE 4—See D.2.1.18 for discussion of appropriate choices for *HashRE*.

NOTE 5—In practice, the pseudo-random behavior of *HashRE* and suitably large values of q and r should ensure that x will never be 0 and e will never be 1.

8.2.17 [EC] REDP-1

ECREDP-1 is elliptic curve random element derivation primitive, version 1. ECREDP-1 is based on DLREDP-1, IEEE Std 1363-2000, A.11, and Wang [B55]. The primitive uses a hash function of a password-based input selector value to select a pseudo-random element of a group to be used in an EC PKAS, in a way that prevents collisions and obscures exponential relationships of output values, as

discussed in 4.4.1. ECREDP-1 may be used to construct a generator value in the schemes ECBPKAS-SPEKE, ECAPKAS-{BSPEKE2,WSPEKE}, and ECPKRS-1, to construct a mask value in the schemes ECBPKAS-{PAK,PPK}, ECAPKAS-PAKZ, and a password-based group element in the primitive ECPEPKGP-SRP5.

This primitive is parameterized by the following choices:

— A hash function *HashRE* with output length *oLen* (see NOTE 5), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

Input:

— The random selector value o_π, which is an octet string derived from a password value

— The EC domain parameters (including q, p, m, a, b and k) associated with o_π

Assumptions: Domain parameters are valid.

Output: The selected group element, which is a point e of order r on the curve defined by the domain parameters, or "invalid."

Operation: The selected group element is computed by the following or an equivalent sequence of steps:

a) Compute an octet string $o_1 = HashRE(o_\pi)$

b) Compute an integer $i_1 = \text{OS2IP}(o_1)$

c) Compute an octet string o_2 of length *oLen* using $\text{I2OSP}(i_1)$

d) Compute an octet string $o_3 = HashRE(o_2)$

e) Compute a field element $x = \text{I2FEP}(\text{OS2IP}(o_3) \bmod q)$

   1) If $x = 0$, output "invalid" and stop. (See NOTE 1.)

f) Compute integer $\mu = i_1 \bmod 2$

g) If q is even, go to step i).

h) When q is odd ($GF(p)$ or $GF(p^m)$) use the following steps adapted from IEEE Std 1363-2000, A.11.1:

   1) Set α as follows:

      i) If $(p>3)$, set $\alpha \leftarrow \exp(x, 3) + a \times x + b$.

      ii) If $(p=3)$, set $\alpha \leftarrow \exp(x, 3) + a \times \exp(x, 2) + b$.

   2) 9. If $\alpha = 0$ then output "invalid" and stop. (See NOTE 1.)

   3) Find a square root β of α or determine that none exist. (Appropriate techniques can be found for $GF(p)$ in IEEE Std 1363-2000, A.2.5, and for $GF(p^m)$ in IEEE Std 1363a-2004, A.17.1.3.)

   4) If the result of step 3) indicates that no square roots exist, then go to step j). (Otherwise the output of step 3) is a field element β such that $\exp(\beta, 2) = \alpha$.)

   5) Set $y \leftarrow \exp(p-1, \mu) \times \beta$, where the integer value $p-1$ represents a field element (the primitive square root of 1) using the polynomial basis representation conversion described in IEEE Std 1363a-2004, 5.3.3

   6) Set point $T_1 \leftarrow (x, y)$ and go to step k).

i) When q is even ($GF(2^m)$) use the following steps adapted from IEEE Std 1363-2000, A.11.2:

1) Set $\beta \leftarrow x + a + b \times \exp(x, -2)$.

2) Find a field element z for which $\exp(z, 2) + z = \beta$ or determine that none exist. (An appropriate technique can be found in IEEE Std 1363-2000, A.4.7.)

3) If the result of step 2) indicates that no solutions exist, then go to step j). (Otherwise the output of step 2) is a solution z.)

4) Set $y \leftarrow (z + \mu) \times x$, where the {0,1} values of integer μ denote field elements {0,1}.

5) Set point $T_1 \leftarrow (x, y)$ and go to step k).

j) When no point on the curve exists at x, try again:

1) Set $i_1 \leftarrow i_1 + 1$

2) Go to step c).

k) Compute the selected group element $e = k \times T_1$

1) If e is the point at infinity, output "invalid" and stop. (See NOTE 1.)

l) Output e

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—Some differences between this method and the techniques described in IEEE Std 1363-2000, A.11, are that this method does not deal with special case values of x. In practice, the pseudo-random behavior of *HashRE* and a suitably large value of q should ensure that x and α will never be 0 and that e will never be the point at infinity.

NOTE 2—ECREDP-1 in the EC setting is analogous to DLREDP-1 in the DL setting.

NOTE 3—ECREDP-2 is described in 8.2.18 *REDP-2*.

NOTE 4—D.2.1.20 describes the security properties of REDP methods in relation to various schemes.

NOTE 5—See D.2.1.18 for discussion of appropriate choices for *HashRE*.

8.2.18 REDP-2

{DL,EC}REDP-2 is {discrete logarithm, elliptic curve} random element derivation primitive, version 2. This primitive uses a hash function of a password-based input selector value to select a pseudo-random element of a group to be used in a {DL,EC} PKAS, in a way that prevents collisions and obscures exponential relationships of output values, as discussed in subclause 4.4.1. This primitive uses two independent generators of the group, as in similar methods described in Jablon [B29], MacKenzie [B41], and Scott [B49]. {DL,EC}REDP-2 may be used to construct a generator value in the schemes {DL,EC}BPKAS-SPEKE, {DL,EC}APKAS-BSPEKE2, {DL,EC}APKAS-WSPEKE, and {DL,EC}PKRS-1 and to construct a mask value in the schemes {DL,EC}BPKAS-PAK, {DL,EC}BPKAS-PPK, and {DL,EC}APKAS-PAKZ.

This primitive is parameterized by the following choices:

— A hash function *HashRE* (see NOTE 2), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1)

— Two random group elements, g_a and g_b, of order r selected from the group generated by g

Input:

— The random selector value o_π, which is an octet string derived from a password value

— {DL,EC} domain parameters (including r and g) associated with o_π

Assumptions: Domain parameters are valid. Group elements g_a and g_b are random distinct group elements of order r, such that no party knows an exponential relationship between g_a and g_b and no party knows an exponential relationship between g_b and g.

Output: The selected group element e, which is a generator of the desired group of order r, or "invalid."

Operation: The selected group element is computed by the following or an equivalent sequence of steps:

a) Compute an octet string $o_1 = HashRE(o_\pi)$

b) Compute an integer $i_2 = \text{OS2IP}(o_1) \bmod r$

   1) If $i_2 = 0$, output "invalid" and stop. (See NOTE 4.)

c) Compute the selected group element $e = g_a * (g_b \hat{} i_2)$

   1) If $e = 1$, output "invalid" and stop. (See NOTE 4.)

d) Output the selected group element e

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—D.2.1.20 describes the security properties of REDP methods in relation to various schemes.

NOTE 2—See D.2.1.17 for discussion of appropriate choices for *HashRE*.

NOTE 3—The random group elements g_a and g_b may be created by g_a = REDP-1(o_a) and g_b = REDP-1(o_b) using distinct selector values o_a and o_b. The selector values may be saved and published to permit other parties to validate that g_a and g_b were chosen appropriately.

NOTE 4—In practice, the pseudo-random behavior of *HashRE* and a suitably large r should ensure that i_2 will never be 0 and that e will never be 1.

8.2.19 SVDP-AMP-CLIENT

{DL,EC}SVDP-AMP-CLIENT is {discrete logarithm, elliptic curve} secret value derivation primitive, version AMP for Client. It is based on the work of Kwon [B38]. It may be used with the scheme {DL,EC}APKAS-AMP-CLIENT. This primitive derives a shared secret value from the Client's private key, the Client's password-limited private key u_π, and a Server's password-entangled public key. This primitive is used by the scheme {DL,EC}APKAS-AMP-CLIENT.

This primitive is parameterized by the following choices:

— A hash function *HashWC* (see NOTE), which should be one of the hash functions in 14.1.

— An octet string o_{ID} that may provide additional input to *HashWC*.

The Client and Server shall use the same *HashWC* functions and o_{ID} parameters with PEPKGP-AMP-SERVER and SVDP-AMP-CLIENT.

Input:

— The Client's private key s

— The Client's password-limited private key u_π

— The Client's public key w_C

— The Server's password-entangled public key w_S

— The {DL,EC} domain parameters (including q) associated with the values s, u_π, w_C, and w_S

Assumptions: Private keys s and u_π and associated domain parameters are valid. w_C and w_S are in the parent group.

Output: The derived shared secret value z, which is a field element of $GF(q)$

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute o_C = GE2OSP-X(w_C)

b) Compute $o_1 = HashWC(o_C \| o_{ID})$

c) Compute i_1 = OS2IP(o_1)

d) Compute $i_2 = ((s + 1) / ((s \times i_1) + u_\pi)) \bmod r$

e) Compute $z_g = w_S \wedge i_2$

f) Compute z = GE2SVFEP(z_g)

g) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.1.6 for criteria for selecting *HashWC*.

8.2.20 SVDP-AMP-SERVER

{DL,EC}SVDP-AMP-SERVER is {discrete logarithm, elliptic curve} secret value derivation primitive, version AMP for Server. This primitive derives a shared secret value from a Client's public key, the Server's private key, and the domain parameter g. This primitive is used by the scheme {DL,EC}APKAS-AMP-SERVER. APKAS-AMP is based on the work of Kwon [B38].

Input:

— The Server's own private key s

— The Client's public key w_C

— The {DL,EC} domain parameters (including q and g) associated with the keys s and w_C

Assumptions: Private key s and associated DL domain parameters are valid. w_C is in the parent group.

Output: The derived shared secret value z, which is a field element of $GF(q)$, or "invalid"

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute $z_g = (w_C * g) \wedge s$

b) If the order of z_g is unacceptably small, output "invalid" and stop. (See NOTE.)

c) Compute z = GE2SVFEP(z_g)

d) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.1.4 for how and why one should validate that z_g is not a small order element and the meaning of "unacceptably small."

8.2.21 SVDP-PAK1-CLIENT

{DL,EC}SVDP-PAK1-CLIENT is {discrete logarithm, elliptic curve} secret value derivation primitive, version PAK1 for Client. It is based on the work of Diffie and Hellman [B10] and MacKenzie [B43]. This primitive derives a shared secret value from the Client's private key and a Server's public key. It may be used with the schemes {DL,EC}BPKAS-PAK-CLIENT and {DL,EC}APKAS-PAKZ-CLIENT.

This primitive performs a similar function as IEEE 1363 {DL,EC}SVDP-DH, with the following exceptions: This primitive does not assume that the input public key is a valid public key. As a result, it may produce an output that corresponds to an element of the set of invalid public keys in the parent group, including small order elements such as the identity element.

Input:

— The Client's own private key s

— The Server's public key w_S

— The {DL,EC} domain parameters (including q, r and g) associated with the keys s and w_S

Assumptions: Private key s and associated domain parameters are valid. w_S is in the parent group.

Output: The derived shared secret value z, which is a field element of $GF(q)$

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute $z_g = w_S \wedge s$ (See NOTE 3.)

b) Compute z = GE2SVFEP(z_g)

c) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—The function that invokes SVDP-PAK1-CLIENT shall ensure that public key w_S is in the parent group. (See also {DL: 6.1, EC: 7.1}.)

NOTE 2—This primitive does not by itself address small subgroup confinement (see IEEE Std 1363-2000, D.5.1.6). Such issues are addressed as necessary by the invoking scheme.

NOTE 3—When the public key w_S is a valid public key, the output z is a valid public key. When the public key w_S is not a valid public key, this primitive returns the computed result. The similar primitive SVDP-DH (IEEE Std 1363-2000) performs the same computation, but is specified as assuming that input public keys are valid, which may be interpreted as permitting an implementation to output either an error or an undefined value when the input key is not a valid public key. SVDP-PAK1-CLIENT is compatible with an implementation of SVDP-DH that performs the specified mathematical operation on public-key input values that are in the parent group but are not valid public keys. When SVDP-PAK1-CLIENT is used in an application that never invokes it with an invalid public key, it is compatible with any implementation of SVDP-DH. SVDP-PAK1-CLIENT is also compatible with any implementation of SVDP-DHC (IEEE Std 1363-2000) in the mode where compatibility with SVDP-DH is selected. However, although SVDP-DHC and SVDP-PAK1-CLIENT will compute the same output values when the input is a valid public key, they may produce different output values when the input is not a valid public key.

8.2.22 SVDP-PAK2

{DL,EC}SVDP-PAK2 is {discrete logarithm, elliptic curve} secret value derivation primitive, version PAK2. It is based on the work of Boyko, MacKenzie, Patel [B8] and MacKenzie [B43]. This primitive derives a shared secret value from the party's private key, a password-based value, and another party's password-entangled public key. This primitive is used by the scheme {DL,EC}BPKAS-PAK.

Input:

— The party's own private key s

— The password-based mask group element π_m

— The other party's password-entangled public key w'

— The {DL,EC} domain parameters (including q and r) associated with the keys s and w' and group element π_m

Assumptions: Private key s and domain parameters are valid. w' is in the parent group. π_m is a generator of the desired group of order r.

Output: The derived shared secret value z, which is a field element of $GF(q)$.

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute a group element $z_g = (w' * \pi_m\text{^}(-1))\ \text{^}\ s$

b) Compute field element z = GE2SVFEP(z_g)

c) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

8.2.23 SVDP-SPEKE

{DL,EC}SVDP-SPEKE is {discrete logarithm, elliptic curve} secret value derivation primitive, version SPEKE. It is based on the work of Jablon [B28] and [B31]. It is defined for use with either the DL or EC family. This primitive derives a shared secret value from the party's private key and another party's password-entangled public key. This primitive is used in BPKAS-SPEKE and APKAS-BSPEKE2. It optionally performs cofactor multiplication.

Input:

— The party's own {DL,EC} private key integer s

— The other party's {DL,EC} password-entangled public key w'

— The {DL,EC} domain parameters (including q) associated with the keys s and w'

— A Boolean value b_1 that indicates whether cofactor multiplication is desired

Assumptions: Private key s and associated domain parameters are valid. Keys s and w' are associated with the same domain parameters. Both parties agree to use the same indication of whether cofactor multiplication is desired. w' is an element of the parent group.

Output: The derived shared secret value z, which is a field element of $GF(q)$

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) If b_1 indicates that cofactor multiplication **is not** desired,

   1) Compute group element $z_g = w' \wedge s$

b) If b_1 indicates that cofactor multiplication **is** desired,

   1) Compute group element $z_g = w' \wedge (k\ s)$

c) Compute z = GE2SVFEP(z_g)

d) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—ECSVDP-DH is defined to output "error" in the case of an identity element result, which is not be possible in this case where w' is a valid public key and s is valid. DLSVDP-DH is not defined for the case of an invalid public key, and never outputs "error" or "invalid."

NOTE 2—SVDP-SPEKE scheme may be replaced by either SVDP-DH or SVDP-DHC as described in the NOTES associated with BPKAS-SPEKE and APKAS-BSPEKE2.

8.2.24 [DL] SVDP-SRP3-CLIENT, SVDP-SRP6-CLIENT

DLSVDP-{SRP3,SRP6}-CLIENT is discrete logarithm secret value derivation primitive, version {SRP3,SRP6} for Client. This primitive derives a shared secret value from the Client's private key, a Server's password-entangled public key, and password-based values, where the keys have the same DL domain parameters. This primitive is used by the scheme DLAPKAS-{SRP3,SRP6}-CLIENT. APKAS-SRP3 is based on the work of Wu [B24] and [B57]. APKAS-SRP6 is based on the work of Wu [B56] and [B57].

For SRP6 only: This primitive is parameterized by the following choices:

— A hash function *HashW2* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1). The Server shall use the same *HashW2* parameter with DLSVDP-SRP6-SERVER.

— A multiplier value creation function *Mvcf*, which should be MVCF-DP (see 12.5.1). The Server shall use the same *Mvcf* and its associated parameters with DLPEPKGP-SRP6-SERVER.

Input:

— The Client's own private key s

— The password-limited private key u_π

— The password verification data v_π, a password-limited public key

— *For SRP6 only*: The Client's own DL public key w_C

— The Server's DL password-entangled public key w_S

— The DL domain parameters (including q and g_{q-1}) associated with the keys s, u_π, v_π, w_C and w_S

Assumptions: Private keys s and u_π, password-limited public key v_π, and DL domain parameters are valid. w_S and w_C are elements of the parent group.

Output: The derived shared secret value z, which is a field element in $GF(q)$

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute octet string $o_2 = \text{FE2OSP}(w_S)$

b) Compute integer i_2 as follows:

1) *For SRP3 only*:

i) Compute octet string $o_3 = \text{SHA-1}(o_2)$

ii) Compute integer $i_1 = \text{OS2IP}(o_3)$

iii) Compute $i_2 = \lfloor i_1 / 2^{128} \rfloor$

2) *For SRP6 only*:

i) Compute octet string $o_1 = \text{FE2OSP}(w_C)$

ii) Compute octet string $o_3 = \textit{HashW2}(o_1 \| o_2)$

iii) Compute $i_2 = \text{OS2IP}(o_3)$

c) Compute field element m_v as follows:

1) *For SRP3 only*: Let $m_v = 1$

2) *For SRP6 only*: Compute m_v using *Mvcf*

d) Compute $z = \exp((w_S - v_\pi \times m_v), (s + i_2 \times u_\pi))$

e) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.4.3 for criteria for selecting *HashW2*.

8.2.25 [DL] SVDP-SRP3-SERVER, SVDP-SRP6-SERVER

DLSVDP-{SRP3,SRP6}-SERVER is discrete logarithm secret value derivation primitive, version {SRP3,SRP6} for Server. This primitive derives a shared secret value from password verification data, the Server's private key and password-entangled public key, and a Client's public key. This primitive is used by the scheme DLAPKAS-{SRP3,SRP6}-SERVER. APKAS-SRP3 is based on the work of Wu [B24] and [B57]. APKAS-SRP6 is based on the work of Wu [B56] and [B57].

For SRP6 only: This primitive is parameterized by the following choices:

— A hash function *HashW2* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1). The Client shall use the same *HashW2* parameter with DLSVDP-SRP6-CLIENT.

Input:

— The Server's own private key s

— The password verification data v_π, a password-limited public key

— The Client's DL public key w_C

— The Server's own DL password-entangled public key w_S

— The DL domain parameters (including g_{q-1} and q) associated with s, v_π, w_C, and w_S

Assumptions: Private key s, password-limited public key v_π, and DL domain parameters are valid. w_C and w_S are elements of the parent group.

Output: The derived shared secret value z, which is an element of the parent group

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute octet string o_2 = FE2OSP(w_S)

b) Compute integer i_2 as follows:

1) *For SRP3 only*:

i) Compute octet string o_3 = SHA-1(o_2)

ii) Compute integer i_1 = OS2IP(o_3)

iii) Compute $i_2 = \lfloor i_1 / 2^{128} \rfloor$

2) *For SRP6 only*:

i) Compute octet string o_1 = FE2OSP(w_C)

ii) Compute octet string o_3 = *HashW2*($o_1 \| o_2$)

iii) Compute i_2 = OS2IP(o_3)

c) Compute z = exp((($w_C \times$ exp(v_π, i_2)), s)

d) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.4.3 for criteria for selecting *HashW2*.

8.2.26 [EC] SVDP-SRP5-CLIENT

ECSVDP-SRP5-CLIENT is elliptic curve secret value derivation primitive, version SRP5 for Client. It is based on the work of Wang [B55] and Wu [B57]. This primitive derives a shared secret value from the Client's private key, password-based values, and a Server's password-entangled public key. This primitive is used by ECAPKAS-SRP5-CLIENT.

This primitive is parameterized by the following choices:

— A REDP function *Redp*, which should be ECREDP-1 or ECREDP-2. The Server shall use the same *Redp* parameter with ECPEPKGP-SRP5-SERVER.

— A hash function *HashW2* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1). The Server shall use the same *HashW2* parameter with SVDP-SRP5-SERVER.

Input:

— The party's own private key s

— The password-limited private key u_π and password-limited public key v_π computed from ECPVDGP-SRP5(π)

— The Client's own public key w_C

— The Server's password-entangled public key w_S

— The EC domain parameters (including q) associated with the keys s, u_π, v_π, w_C and w_S

Assumptions: Private keys s and u_π, public key v_π, and domain parameters are valid. w_C and w_S are elements of the parent group.

Output: The derived shared secret value z, which is a field element in $GF(q)$

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute o_1 = GE2OSP-X(w_C)
b) Compute o_2 = GE2OSP-X(w_S)
c) Compute o_3 = *HashW2*(o_1 || o_2)
d) Compute i_2 = OS2IP(o_3)
e) Compute o_4 = GE2OSP-X(v_π)
f) Compute group element e_1 = *Redp*(o_4)
g) Compute $e_2 = w_S * (e_1\text{^}(-1))$
h) Compute $z_g = e_2 \text{ ^ } (s + (i_2\, u_\pi))$
i) Compute z = GE2SVFEP(z_g)
j) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.5.1 for criteria for selecting *HashW2*.

8.2.27 [EC] SVDP-SRP5-SERVER

ECSVDP-SRP5-SERVER is elliptic curve secret value derivation primitive, version SRP5 for Server. It is based on the work of Wang [B55] and Wu [B57]. This primitive derives a shared secret value from the Server's private key, the Server's password-entangled public key, password verification data, and a Client's public key. It is used with the scheme ECAPKAS-SRP5-SERVER.

This primitive is parameterized by the following choice:

— A hash function *HashW2* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1). The Client shall use the same *HashW2* parameter with SVDP-SRP5-CLIENT.

Input:

— The Server's own private key s
— The password verification data v_π, a password-limited public key
— The Client's public key w_C
— The Server's own password-entangled public key w_S
— The EC domain parameters (including q) associated with s, v_π w_C, and w_S

Assumptions: Private key s, public key v_π, and domain parameters are valid. w_C and w_S are elements of the parent group.

Output: The derived shared secret value *z*, which is an element of the parent group

Operation: The shared secret value *z* shall be computed by the following or an equivalent sequence of steps:

a) Compute o_1 = GE2OSP-X(w_C)

b) Compute o_2 = GE2OSP-X(w_S)

c) Compute o_3 = *HashW2*(o_1 || o_2)

d) Compute i_2 = OS2IP(o_3)

e) Compute $z_g = (w_C * (v_\pi \wedge i_2)) \wedge s$

f) Compute z = GE2SVFEP(z_g)

g) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.5.1 for criteria for selecting *HashW2*.

8.2.28 SVDP-WSPEKE-CLIENT, SVDP-WSPEKE-SERVER

{DL,EC}SVDP-WSPEKE-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} secret value derivation primitive, version WSPEKE for {Client, Server}. It is based on the work of Wu [B57] and Jablon [B27], [B28], [B30]. It is defined for use in either the DL or EC setting. SVDP-WSPEKE-CLIENT derives a shared secret value from a password-based value, the Client's private key, and a Server's password-entangled public key. SVDP-WSPEKE-SERVER derives a shared secret value from password verification data, the Server's private key, the Server's password-entangled public key, and a Client's password-entangled public key. This primitive is used by the scheme APKAS-WSPEKE-{CLIENT,SERVER}.

This primitive is parameterized by the following choices:

— A hash function *HashWS* (see NOTE), which should be one of the hash functions in 12.1 or MGF1 (see 12.2.1). The Client and Server shall use the same *HashWS* parameters with SVDP-WSPEKE-CLIENT and SVDP-WSPEKE-SERVER.

Input:

— The party's own private key *s*

— *For CLIENT only:* The password-limited private key u_π

— *For SERVER only:* The password verification data v_π, a password-limited public key derived using {DL,EC}PVDGP-BSPEKE2(π)

— *For SERVER only:* The Client's password-entangled public key w_C

— The Server's password-entangled public key w_S

— The domain parameters (including *q*) associated with *s*, u_π, v_π, w_C and w_S

Assumptions: Private keys *s* and u_π, public key v_π, and domain parameters are valid. w_C and w_S are elements of the parent group.

Output: The derived shared secret value *z*, which is an element of the parent group

Operation: The shared secret value z shall be computed by the following or an equivalent sequence of steps:

a) Compute o_1 = GE2OSP-X(w_S)

b) Compute o_2 = *HashWS*(o_1)

c) Compute i_1 = OS2IP(o_2)

d) Compute group element z_g as follows:

1) *For CLIENT only*: Compute $z_g = w_S \wedge (s + u_\pi \times i_1)$

2) *For SERVER only*: Compute $z_g = (w_C * (v_\pi \wedge i_1)) \wedge s$

e) Compute z = GE2SVFEP(z_g)

f) Output z

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE—See D.2.2.2.3 for criteria for selecting *HashWS*.

9. Password-authenticated key agreement schemes

The general model for a password-authenticated key agreement scheme (PKAS) is given in 9.1. Specific schemes and their allowable options are given in 9.2 through 9.10.

9.1 General model

In a PKAS, each party combines its own password-based value and private key(s) with the other party's public key(s) to come up with one or more secret keys. Other (public or private) information known to both parties may also enter the scheme as *key derivation parameters*. If the parties use the corresponding passwords, keys, and identical key derivation parameters, and the scheme is executed correctly, the parties will arrive at the same secret keys. A PKAS can allow two parties to derive shared secret keys without any prior shared secret other than password-based values, with confidence that no party will be able to successfully participate in the protocol without using the password-based values.

The schemes in this clause define *key agreement operations* to allow two parties, designated *Client* and *Server*, to derive shared secret keys (see NOTE 2 in 9.1.1). Each PKAS also defines *key confirmation operations*, for each party to confirm that the other party has successfully derived the shared keys (see D.2.1.2). Domain parameter and key pair generation for the key agreement schemes are specified further in connection with the DL and EC families (Clause 6 and Clause 7, respectively). Further security considerations for the schemes in this clause are given in D.2.1 and D.2.2, and, for key agreement schemes in general, in IEEE Std 1363-2000, D.5.1 .

These PKAS methods provide the following security properties:

— Successful use of scheme to agree on a shared key is limited to parties that know the password value (π) or password-verification data that corresponds to π.

— Knowledge of the password-entangled public keys and any other public keys exchanged between participants does not allow a third party to verify guesses for π, or to determine the shared keys.

— A participant that does not know π gets the opportunity to verify at most one (or at most a very small number) of guesses for π for each interaction with the other.

9.1.1 PKAS types

The two types of PKAS described in this standard include:

— Balanced password-authenticated key agreement schemes (BPKAS), in which a Client and a Server use a shared password value π and ephemeral private keys to mutually derive ephemeral shared keys, where the shared keys are established if and only if they use the same value for π.

— Augmented password-authenticated key agreement schemes (APKAS), in which a Client has a password value π and a Server has corresponding password verification data derived from a one-way function of π. The two parties use ephemeral private keys and their corresponding password-based values to mutually derive ephemeral shared keys, where the shared keys are established if and only if they use the corresponding values for π and verification data. An APKAS further prevents an attacker who has stolen the Server's verification data from masquerading as the Client without first performing a successful brute-force password attack on the verification data.

NOTE 1—See C.2 for a rationale for why multiple schemes are included.

NOTE 2—The *Client* and *Server* designations should be considered arbitrary labels associated with actions defined by the scheme for each party; the parties need not conform to client/server roles in any other respects. For example, these designations are aligned with a typical APKAS or PKRS client/server application, but arbitrary for a peer-to-peer BPKAS application.

9.1.2 PKAS operations

Some significant differences between a PKAS and a KAS are noted as follows. Unless specifically noted here, all other comments and definitions in IEEE Std 1363-2000, 9.1, regarding a KAS are applicable to a PKAS.

A password-authenticated key agreement operation has the following form for all the schemes in this standard:

a) Establish a set of valid domain parameters.

b) Establish one or more password-based values, associated with the domain parameters.

c) Select one or more valid private keys for the operation, associated with the domain parameters.

d) Use password-entangled public-key generation (PEPKG) and/or public-key generation (PKGP) operations to create one or more public keys corresponding to the private keys of step c). Password-entangled public keys further incorporate a password-based value of step b).

e) Make the [password-entangled] public keys of step d) available to another party.

f) Obtain one or more other party's purported [password-entangled] public keys for the operation. (See NOTE 3.)

g) *(Optional)* Depending on the cryptographic operations in step d), choose an appropriate method to validate the [password-entangled] public keys and the domain parameters. If any validation fails, output "invalid" and stop. (See NOTE 6.)

h) Apply certain cryptographic operations (secret value derivation) to the private and [password-entangled] public keys to produce a shared secret value.

i) For each shared secret key to be agreed on, establish or agree on key derivation parameters, and derive a shared secret key from the shared secret value and the key derivation parameters using a KDF. (See NOTE 9.)

NOTE 1—(*Explicit public-key creation.*) In contrast with a KAS (IEEE Std 1363-2000, 9.1), a PKAS description includes explicit steps for creating one or more public keys (which may be *password-entangled* public keys), steps for

making them available to another party, and steps for receiving such keys from another party. The explicit steps and primitives for public-key creation are needed as there is considerable variation in the construction of public-keys (and password-entangled public-keys) in different schemes.

NOTE 2—(*Password entanglement.*) At least one of the public keys created by or received in a PKAS is a *password-entangled* public key, which further incorporates a password-based value, in order to provide the password-authenticated benefit of the PKAS.

NOTE 3—(*Key confirmation.*) Parties that engage in a PKAS will generally require key confirmation to the same extent and for the same reasons as those that engage in an ordinary key agreement scheme, as discussed in IEEE Std 1363-2000, 9.1 (NOTE 1), and IEEE Std 1363-2000, D.5.1.3. This document further discusses key confirmation issues in D.2.1.2.

NOTE 4—(*Repeated use of a key pair.*) A PKAS generally creates at least one ephemeral key pair, in which neither the private nor the public keys may be used more than once.

NOTE 5—(*Authentication of ownership.*) A PKAS authenticates each party's public key to the other based on a password, which can eliminate the need for additional authentication. In this regard, a PKAS may be particularly applicable for use in applications where there is no other reliable basis for key authentication. However, in some applications, additional authentication of one or more parties may be available or desired, and may be achieved by other methods (such as those that use a key management infrastructure or certifying authority) as discussed in IEEE Std 1363-2000, 9.1 (NOTE 3), D.3.2, and D.5.1.5.

NOTE 6—(*Key validation.*) A PKAS generally requires key validation, for the same reasons as discussed in IEEE Std 1363-2000, 9.1, D.3.3, and D.5.1.6. Methods for key validation are generally specific to each scheme, and are generally defined within the scheme, rather than within primitive functions. (See D.2.1.3.)

NOTE 7—*(Domain parameter validation.)* A PKAS generally requires domain parameter validation to the same extent and for the same reasons as discussed in IEEE Std 1363-2000, 9.1 (NOTE 4). More specific considerations are discussed in D.2.1.3.

NOTE 8—(*Order of steps.*) There may be requirements for the relative ordering of some PKAS operations to ensure the integrity of the scheme. Some PKAS require that certain steps of the scheme be completed by a party before certain results computed by that party are revealed to other parties. See D.2.1.9 for general discussion of such requirements in *unilateral commitment* PKAS.

NOTE 9—(*Limitations on use of derived keys.*) Some PKAS impose requirements for key confirmation operations to be performed before other use of derived keys, as discussed in D.2.1.9.

NOTE 10— (*Error conditions.*) The two parties may produce errors under certain conditions, such as the following:

- Private key not found in step c)
- Public key not found in step f)
- Public key not valid in step g)
- Private key or public key not supported in step h)
- Key derivation parameter not supported in step i)

Such error conditions should be detected and handled appropriately by an implementation, but the specific methods for detecting and handling them are outside of the scope of this standard.

9.2 BPKAS-PAK

{DL,EC}BPKAS-PAK-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} balanced password-authenticated key agreement scheme, version PAK for {Client, Server}. It is based on the work of Boyko, MacKenzie, Patel [B8].

Figure 1 illustrates the scheme.

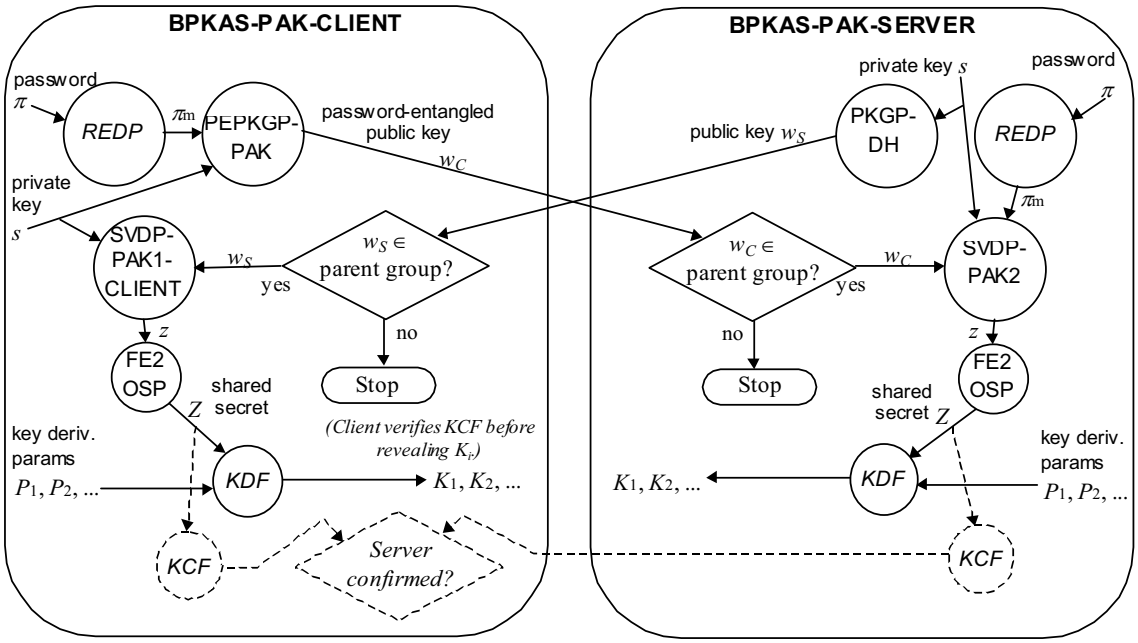

Figure 1—BPKAS-PAK key agreement operation

9.2.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for public-key generation and secret value derivation, which shall be PEPKGP-PAK, PKGP-DH, SVDP-PAK1-CLIENT, SVDP-PAK2, and their associated parameters

— A shared password-based octet string π (See NOTE 4 in 9.2.3.2.)

— A set of valid {DL,EC} domain parameters (including q and r) associated with π and keys s, w_C, and w_S

— A REDP function *Redp*, which should be {DL,EC}REDP-1 or {DL,EC}REDP-2

— A KDF *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.2.2 Key agreement operation

The following key agreement operation steps are used in both BPKAS-PAK and APKAS-PAKZ (see NOTE 5 in 9.2.3.2). A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps.

9.2.2.1 Key agreement for Client

a) Compute a mask group element $\pi_m = Redp(\pi)$

b) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

c) Compute password-entangled public key w_C using {DL,EC}PEPKGP-PAK(s, π_m)

d) Send w_C to the Server

e) Receive public-key value w_S from the Server

f) If w_S is not an element of the parent group, output "invalid" and stop.

g) Compute field element z using {DL,EC}SVDP-PAK1-CLIENT(s, w_S)

h) Compute octet string Z = FE2OSP(z)

i) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

NOTE—The Client shall confirm the Server's knowledge of shared secret Z before any derived keys are used. Key confirmation for BPKAS-PAK is described in 9.2.3. Key confirmation for APKAS-PAKZ is described in 9.7.3.

j) Output derived keys $K_1, K_2, \ldots K_t$

9.2.2.2 Key agreement for Server

a) Compute a mask group element $\pi_m = Redp(\pi)$ (See NOTE 3 in 9.2.3.2.)

b) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

c) Compute public key w_S using {DL,EC}PKGP-DH(s)

d) Send w_S to the Client

e) Receive password-entangled public-key value w_C from the Client

f) If w_C is not an element of the parent group, output "invalid" and stop.

g) Compute field element z using {DL,EC}SVDP-PAK2(s, π_m, w_C)

h) Compute octet string Z = FE2OSP(z)

i) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

j) Output derived keys $K_1, K_2, \ldots K_t$

9.2.3 Key confirmation operation

The BPKAS-PAK Client shall confirm the Server's knowledge of the shared secret Z, before the Client uses Z or any derived shared secrets K_i for other KCFs or other purposes (see NOTE 1 in 9.2.3.2). Explicit confirmation of the Client's knowledge of Z to the Server is optional.

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.2.3.1 Key confirmation for Server

a) *Mandatory*

1) Compute o_π = GE2OSP-X(π_m)

2) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

3) Send C_S to the Client

b) *Optional*

1) Receive octet string C_C from the Client

2) Compute o_π = GE2OSP-X(π_m)

3) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

4) If $C_C' \neq C_C$, output "invalid" and stop.

9.2.3.2 Key confirmation for Client

a) *Mandatory*

1) Receive octet string C_S from the Server

2) Compute o_π = GE2OSP-X(π_m)

3) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

4) If $C_S' \neq C_S$, output "invalid" and stop.

NOTE—Step a) shall occur before step b), and before any other use of shared keys K_i that are derived from the Client's key agreement operation.

b) *Optional*

1) Compute o_π = GE2OSP-X(π_m)

2) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

3) Send C_C to the Server

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—BPKAS-PAK is a unilateral commitment scheme, where the Server does not provide a commitment to the password during the key agreement operations. In using this scheme, the Client needs to verify the Server's proof of knowledge of the agreed key before revealing any information derived from the agreed key. See D.2.1.9 for discussion of the limitations on the use of unilateral commitment schemes in application protocols.

NOTE 2—This scheme is designed to prevent any adverse effects from receiving and using an invalid public key; if not caught during the key agreement operation, any such problems will result in a subsequent key confirmation failure.

NOTE 3—The BPKAS-PAK Server may choose to store the value of π_m as precomputed password-verification data to reduce computation in Server key agreement operations.

NOTE 4—The BPKAS-PAK Server may want to store π_m^(−1) to optimize efficiency.

NOTE 5—Steps b) through i) of the BPKAS-PAK key agreement for Client operation and steps b) through i) of the BPKAS-PAK key agreement for Server operation are reused in APKAS-PAKZ, using a different computation of π_m.

9.3 BPKAS-PPK

{DL,EC}BPKAS-PAK-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} balanced password-authenticated key agreement scheme, version PPK for {Client, Server}. It is based on the work of Boyko, MacKenzie, Patel [B8] and MacKenzie [B43].

Figure 2 illustrates the scheme.

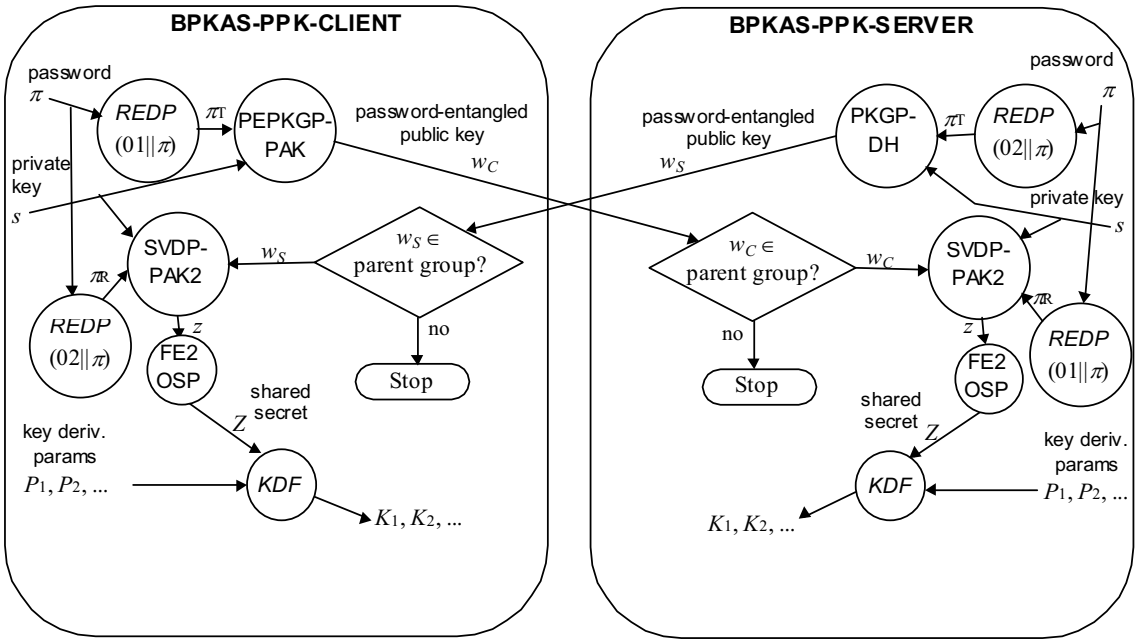

Figure 2—BPKAS-PPK key agreement operation

9.3.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for public-key generation and secret value derivation, which shall be PEPKGP-PAK, SVDP-PAK2, and their associated parameters

— A shared password-based octet string π (See NOTE 4 in 9.3.3.2.)

— A set of valid {DL,EC} domain parameters (including q and r) associated with π and keys s, w, and w'

— A REDP function *Redp*, which should be {DL,EC}REDP-1 or {DL,EC}REDP-2

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, \ldots\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.3.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_l, shall be generated by each party by performing the following or an equivalent sequence of steps:

a) For CLIENT only:

   1) Compute a first mask group element $\pi_T = Redp(\ hex(01) \parallel \pi\ )$

   2) Compute a second mask group element $\pi_R = Redp(\ hex(02) \parallel \pi\ )$

b) For SERVER only:

   1) Compute a first mask group element $\pi_T = Redp(\ hex(02) \parallel \pi\ )$

2) Compute a second mask group element $\pi_R = Redp(\ hex(01) \parallel \pi\ )$

c) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

d) Compute group element w using {DL,EC}PEPKGP-PAK(s, π_T)

e) Send w to the other party, to be used by the other party (as w') in key agreement operation

f) Receive password-entangled public-key value w' from the other party, which was created (as w) in other party's step d)

g) If w' is not in the parent group, output "invalid" and stop.

h) Compute field element z using {DL,EC}SVDP-PAK2(s, π_R, w')

i) Compute octet string Z = FE2OSP(z)

j) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

k) Output derived keys K_1, K_2, ... K_t

9.3.3 Key confirmation operation

It is optional in BPKAS-PPK for either the Client or the Server to confirm the other party's knowledge of the shared secret Z, before the party uses Z or any derived shared secrets K_i for other purposes (see NOTE 1 in 9.3.3.2).

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.3.3.1 Key confirmation for Server

a) *Optional*

1) Compute o_T = GE2OSP-X(π_T)

2) Compute $C_S = Kcf(hex(03), w', w, Z, o_T)$

3) Send C_S to the Client

b) *Optional*

1) Receive octet string C_C from the Client

2) Compute o_R = GE2OSP-X(π_R)

3) Compute $C_C' = Kcf(hex(04), w', w, Z, o_R)$

4) If $C_C' \neq C_C$, output "invalid" and stop.

9.3.3.2 Key confirmation for Client

a) *Optional*

1) Receive octet string C_S from the Server

2) Compute o_R = GE2OSP-X(π_R)

3) Compute $C_S' = Kcf(hex(03), w, w', Z, o_R)$

4) If $C_S' \neq C_S$, output "invalid" and stop.

b) *Optional*

1) Compute o_T = GE2OSP-X(π_T)
2) Compute $C_C = Kcf(hex(04), w, w', Z, o_T)$
3) Send C_C to the Server

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—BPKAS-PPK is a bilateral commitment scheme, where both parties provide a commitment to the password during the key agreement operations (see D.2.1.9).

NOTE 2—This scheme is designed to prevent any adverse effects from receiving and using an invalid public key; if not caught during the key agreement operation, any such problems will result in a subsequent key confirmation failure.

NOTE 3—Note that π_T for the Client is the same as π_R for the Server, and π_T for the Server is the same as π_R for the Client.

NOTE 4—The BPKAS-PPK Server may want to store π_T, π_R and π_R^(−1) to optimize efficiency.

9.4 BPKAS-SPEKE

{DL,EC}BPKAS-SPEKE-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} balanced password-authenticated key agreement scheme, version SPEKE for {Client, Server}. It is based on the work of Jablon [B28] and [B31].

Figure 3 illustrates the scheme.

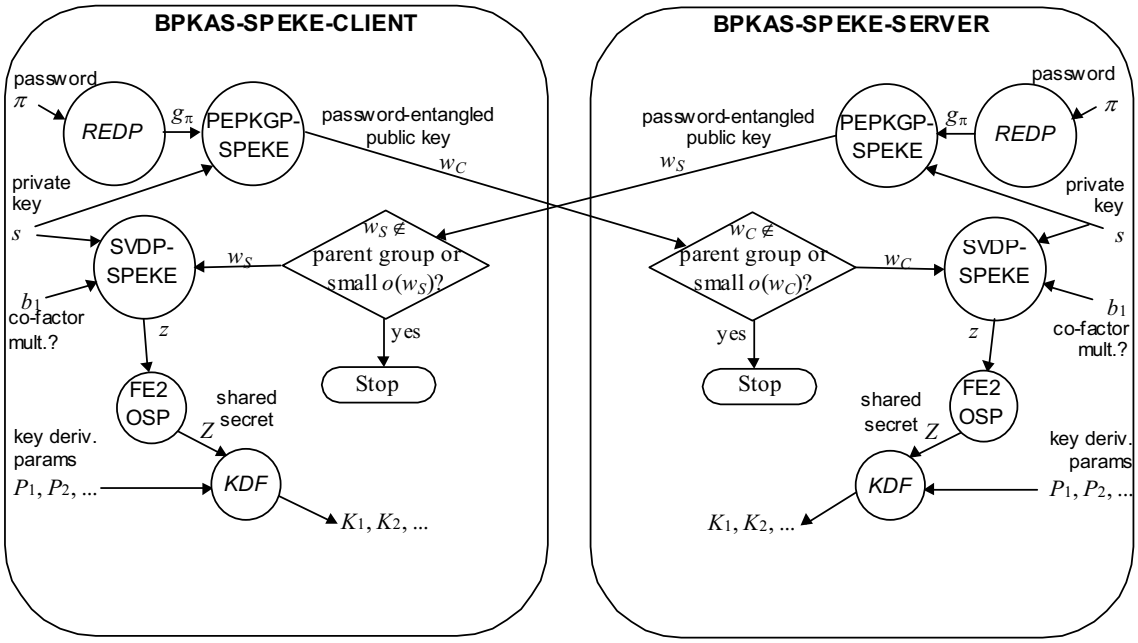

Figure 3—BPKAS-SPEKE key agreement operation

9.4.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for public-key generation and secret value derivation, which shall be PEPKGP-SPEKE, SVDP-SPEKE, and their associated parameters

— A shared password-based octet string π

— A set of valid {DL,EC} domain parameters (including q and r) associated with π and keys s, w, and w'

— A REDP function *Redp*, which should be {DL,EC}REDP-1 or {DL,EC}REDP-2

— A Boolean value b_1 that indicates whether cofactor multiplication is desired

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.4.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps:

a) Compute a generator group element $g_\pi = Redp(\pi)$

b) Obtain private key s, a random integer in the range $[1, r-1]$ (See D.2.1.16.)

c) Compute public key w using {DL,EC}PEPKGP-SPEKE(s, g_π)

d) Send w to the other party (to be received as w' in other party's key agreement operation)

e) Receive public-key value w' from the other party (sent as w in other party's key agreement operation)

   1) If w' is not an element of the parent group, output "invalid" and stop. (See NOTE 7 in 9.4.3.2.)

   2) If the order of w' is unacceptably small, output "invalid" and stop. (See NOTE 7 in 9.4.3.2.)

   3) *(Optional)* If w' is not a valid public key, output "invalid" and stop. (See NOTE 8 in 9.4.3.2.)

f) Compute field element z using {DL,EC}SVDP-SPEKE(s, w', b_1) (See NOTE 8 in 9.4.3.2.)

g) Compute octet string Z = FE2OSP(z)

h) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

i) Output derived keys K_1, K_2, ... K_t

9.4.3 Key confirmation operation

It is optional in BPKAS-SPEKE-{CLIENT,SERVER} for the {Client, Server} to confirm the other party's knowledge of the shared secret Z, before the {Client, Server} uses Z or any derived shared secrets K_i for other purposes (see NOTE 1 in 9.4.3.2).

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.4.3.1 Key confirmation for Server

a) *Optional*

1) Compute o_π = GE2OSP-X(g_π)

2) Compute $C_S = Kcf(hex(03), w', w, Z, o_\pi)$

3) Send C_S to the Client

b) *Optional*

1) Receive octet string C_C from the Client

2) Compute o_π = GE2OSP-X(g_π)

3) Compute $C_C' = Kcf(hex(04), w', w, Z, o_\pi)$

4) If $C_C' \neq C_C$, output "invalid" and stop.

9.4.3.2 Key confirmation for Client

a) *Optional*

1) Receive octet string C_S from the Server

2) Compute o_π = GE2OSP-X(g_π)

3) Compute $C_S' = Kcf(hex(03), w, w', Z, o_\pi)$

4) If $C_S' \neq C_S$, output "invalid" and stop.

b) *Optional*

1) Compute o_π = GE2OSP-X(g_π)

2) Compute $C_C = Kcf(hex(04), w, w', Z, o_\pi)$

3) Send C_C to the Server

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—BPKAS-SPEKE is a bilateral commitment scheme, where both parties provide a commitment to the password during the key agreement operations (see D.2.1.9).

NOTE 2—In BPKAS-SPEKE, the key agreement operation of Client and Server is identical. The Client's password-entangled public key w corresponds to the Server's received value w', and the Server's password-entangled public key w corresponds to the Client's received value w'.

NOTE 3—Either party may subsequently use a zero knowledge proof of knowledge to prove to the other party knowledge of the negotiated key K, which implies knowledge of the password. Such proofs can reliably use common techniques, such as those based on hash functions or symmetric encryption functions, since K has large entropy.

NOTE 4—This scheme provides the flexibility to use any of the IEEE 1363 DH/DHC primitives, either with or without cofactor multiplication, to permit use of a variety of DH/DHC implementations.

NOTE 5—If there is no prior agreement on which party is to be the Client or Server, one way to distinguish roles is to arbitrarily designate the party that generates the larger value for its w as the Server.

NOTE 6—This scheme, with appropriate restrictions on the scheme options and inputs, represent a specific use of, and may be compatible with, the techniques in IEEE Std 1363-2000 DLKAS-DH1 and ECKAS-DH1, ANSI X9.42 (see IEEE Std 1363-2000, [B8]) (in the DL case), and ANSI X9.63 (see IEEE Std 1363-2000, [B12]) (in the EC case).

NOTE 7—The Client and Server may test the received public-key value w' and abort if it is not a valid public key. However, such a test is more than sufficient for compliance. The acceptable values are the elements of the parent group that generate a sufficiently large group. See D.2.2.2.1 for discussion of validating acceptable received public-key values for the SPEKE schemes and the meaning of "sufficiently large."

NOTE 8—When b_1 indicates that cofactor multiplication *is not* desired, SVDP-SPEKE may be replaced by SVDP-DHC (see IEEE Std 1363a-2004, 6.2.2 and 7.2.2) with an indication that compatibility with SVDP-DH *is* desired, or, when the optional key agreement step is used to abort if w' is not a valid public key, SVDP-SPEKE may be replaced by SVDP-DH (see IEEE Std 1363a-2004, 6.2.1 and 7.2.1). (SVDP-DH is defined only for valid public keys.) When b_1 indicates that cofactor multiplication *is* desired, SVDP-SPEKE may be replaced by SVDP-DHC with an indication that compatibility with {DL,EC}SVDP-DH *is not* desired. In either case when SVDP-DHC is used as a replacement for SVDP-SPEKE, the GCD of domain parameters k and r shall be equal to 1, and when the output from SVDP-DHC is "invalid public key," the scheme should output "invalid" and stop.

NOTE 9—BPKAS-SPEKE is comparable to Key Agreement Mechanism 1 in ISO/IEC 11770-4:2006 [B26]. However, differences in conversion primitives prevent interoperability between these methods, randomly, with non-negligible probability, and key confirmation operations are not interoperable in EC settings. See D.2.1.23 for more discussion of differences between these standards.

9.5 APKAS-AMP

{DL,EC}APKAS-AMP-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} augmented password-authenticated key agreement scheme, version AMP for {Client, Server}. It is based on the work of Kwon [B38]. The Server uses a password-limited public key v_π as password verification data that was derived using PVDGP-AMP with the input value π, which is a password-based octet string used by the Client.

Figure 4 illustrates the scheme.

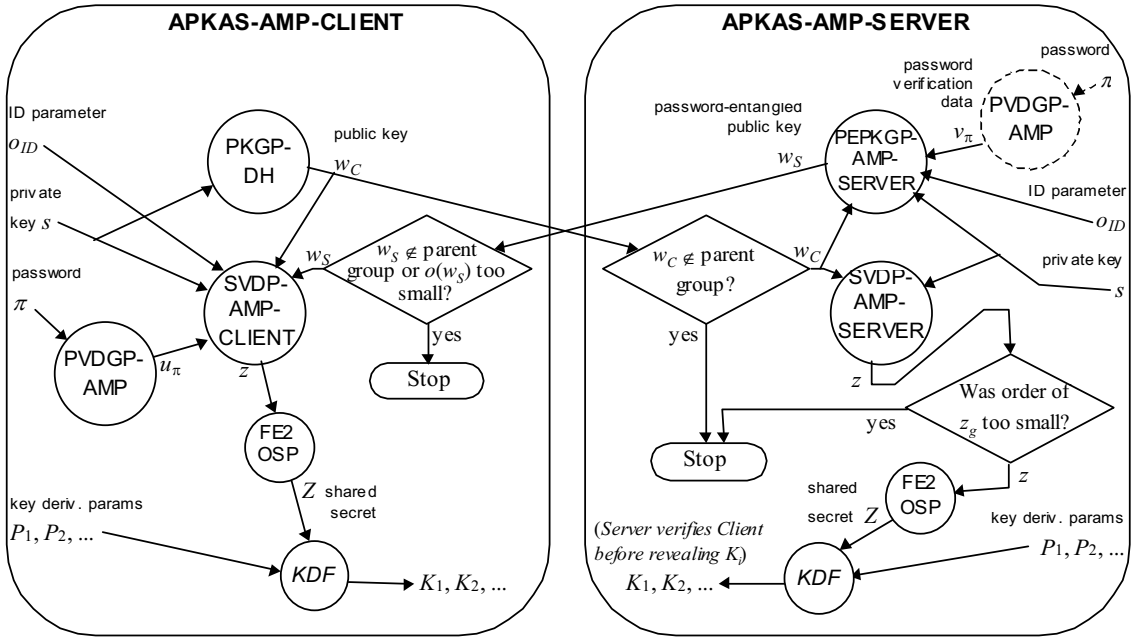

Figure 4—APKAS-AMP key agreement operation

9.5.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

For CLIENT only:

— A password-based octet string π

For SERVER only:

— A password-limited public key v_π that was generated using {DL,EC}PVDGP-AMP with the same parameter and input values as used in the Client's key agreement operation.

For both CLIENT and SERVER:

— Primitives for password verification data generation, public-key generation, and secret value derivation, which shall be PVDGP-AMP, PKGP-DH, PEPKGP-AMP-SERVER, SVDP-AMP-CLIENT, and SVDP-AMP-SERVER, and their associated parameters. The AMP Client and Server shall use the same *HashWC* and o_{ID} parameters with PEPKGP-AMP-SERVER and SVDP-AMP-CLIENT.

— A set of valid {DL,EC} domain parameters (including q and r) associated with π and v_π and keys s, w_C, and w_S. (See NOTE 4 in 9.5.3.2.)

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.5.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps.

9.5.2.1 Key agreement for Client

a) Obtain private key s, a random integer in the range $[1, r-1]$ (See D.2.1.16.)

b) Compute public key w_C = {DL,EC}PKGP-DH(s)

c) Send w_C to the Server

NOTE—Step c) shall occur before step d), since the Server uses w_C to compute w_S.

d) Receive password-entangled public-key value w_S from the Server

   1) If w_S is not in the parent group, output "invalid" and stop. (See NOTE 2 in 9.5.3.2.)

   2) If the order of w_S is unacceptably small, output "invalid" and stop. (See NOTE 2 in 9.5.3.2.)

   3) *(Optional)* If w_S is not a valid public key, output "invalid" and stop. (See NOTE 5 in 9.5.3.2.)

e) Compute password-limited private key u_π using the steps described in {DL,EC}PVDGP-AMP(π)

f) Compute field element z using {DL,EC}SVDP-AMP-CLIENT(s, u_π, w_C, w_S)

g) Compute octet string Z = FE2OSP(z)

h) For each key derivation parameter P_i, derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

i) Output derived keys $K_1, K_2, \ldots K_t$

9.5.2.2 Key agreement for Server

a) Obtain private key s, a random integer in the range $[1, r-1]$ (See D.2.1.16.)

b) Receive public-key value w_C from the Client

1) If w_C is not an element of the parent group, output "invalid" and stop. (See NOTE 2 in 9.5.3.2.)

2) *(Optional)* If w_C is not a valid public key, output "invalid" and stop. (See NOTE 4 in 9.5.3.2.)

c) Generate password-entangled public key value w_S using {DL,EC}PEPKGP-AMP-SERVER(s, v_π, w_C)

d) Send w_S to the Client

e) Compute field element z using {DL,EC}SVDP-AMP-SERVER(s, w_C)

1) If the SVDP function in step 5 outputs "invalid," output "invalid" and stop. (See NOTE 2 in 9.5.3.2.)

f) Compute octet string Z = FE2OSP(z)

g) For each key derivation parameter P_i, derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

NOTE—The Server shall confirm the Client's knowledge of shared secret Z before any derived keys are used. Key confirmation is described in 9.5.3.

h) Output derived keys $K_1, K_2, \ldots K_t$

9.5.3 Key confirmation operation

The APKAS-AMP Server shall confirm the Client's knowledge of the shared secret Z, before the Server uses Z or any derived shared secrets K_i for other KCFs or other purposes (see NOTE 1 in 9.5.3.2). Explicit confirmation of the Server's knowledge of Z to the Client is optional.

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.5.3.1 Key confirmation for Server

a) *Mandatory*

1) Receive octet string C_C from the Client

2) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, "")$

3) If $C_C' \neq C_C$, output "invalid" and stop.

NOTE—Step a) shall occur before step b), and before any other use of shared keys K_i that are derived from the Server's key agreement operation.

b) *Optional*

1) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, "")$

2) Send C_S to the Client

9.5.3.2 Key confirmation for Client

a) *Mandatory*

1) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, "")$

2) Send C_C to the Server

b) *Optional*

1) Receive octet string C_S from the Server

2) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, "")$

3) If $C_S' \neq C_S$, output "invalid" and stop.

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—APKAS-AMP is a unilateral commitment scheme, where the Client does not provide a commitment to the password during the key agreement operations. In using this scheme, the Server needs to verify the Client's proof of knowledge of the agreed key before revealing any information derived from the agreed key. See D.2.1.9 for discussion of the limitations on the use of unilateral commitment schemes in application protocols.

NOTE 2—The required public-key validation during key agreement [Client steps d1) and d2), and Server step b1)] may be implemented in a variety of ways. For example, in the DL settings recommended in NOTE 3, it is sufficient for the Client to ensure that w_S is an integer in the range [2, q−2] and for the Server to ensure that w_S is an integer in the range [1,q−1]. See D.2.2.1 for discussion of the security considerations for AMP, including the reasons for these steps and the meaning of "unacceptably small" order.

NOTE 3—For DL settings, it is recommended that q−1 have no factors in the range [3,r−1]. For EC settings, it is recommended that k be either 1, 2, or 4. See D.2.2.1.1 for how domain parameter choice is related to public-key validation, performance, security, and alignment with other standards.

NOTE 4—When using domain parameters as recommended in NOTE 3, the Server does not need to ensure that w_C is a valid public key in step b2). The Server may perform this step without adversely affecting the method, although this may require significant added computation, such as when using a DL $GF(p)$ setting with a "safe prime" p. When using domain parameters other than as recommended in NOTE 3, step b2), or some other further validation may be necessary to prevent attack. See D.2.2.1 for discussion of how public-key validation is related to domain parameter choice, performance, and security in APKAS-AMP.

NOTE 5—Other than ensuring that the order of w_S is acceptably large in step d2), the Client does not need to ensure that w_S is a valid public key. However, the Client may perform the stricter validity check in optional step d3) without adversely affecting the method.

NOTE 6—APKAS-AMP is significantly different than the comparable Key Agreement Mechanism 3 in ISO/IEC 11770-4:2006 [B26], such that these methods will not interoperate. See D.2.1.23 for more discussion of differences between these standards.

9.6 APKAS-BSPEKE2

{DL,EC}APKAS-BSPEKE2-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} augmented password-authenticated key agreement scheme, version BSPEKE2 for {Client, Server}. It is based on the work of Jablon [B27] and [B28]. This scheme is a password-authenticated instantiation of {DL,EC}KAS-DH2 that creates two shared secret values z_1 and z_2, where z_1 is established using a password-derived generator, as in BPKAS-SPEKE, and z_2 is established using a password-derived exponent for the Client. The Client uses a password-based octet string π and the Server uses password verification data generated by PVDGP-BSPEKE2 using π.

Figure 5 illustrates the scheme.

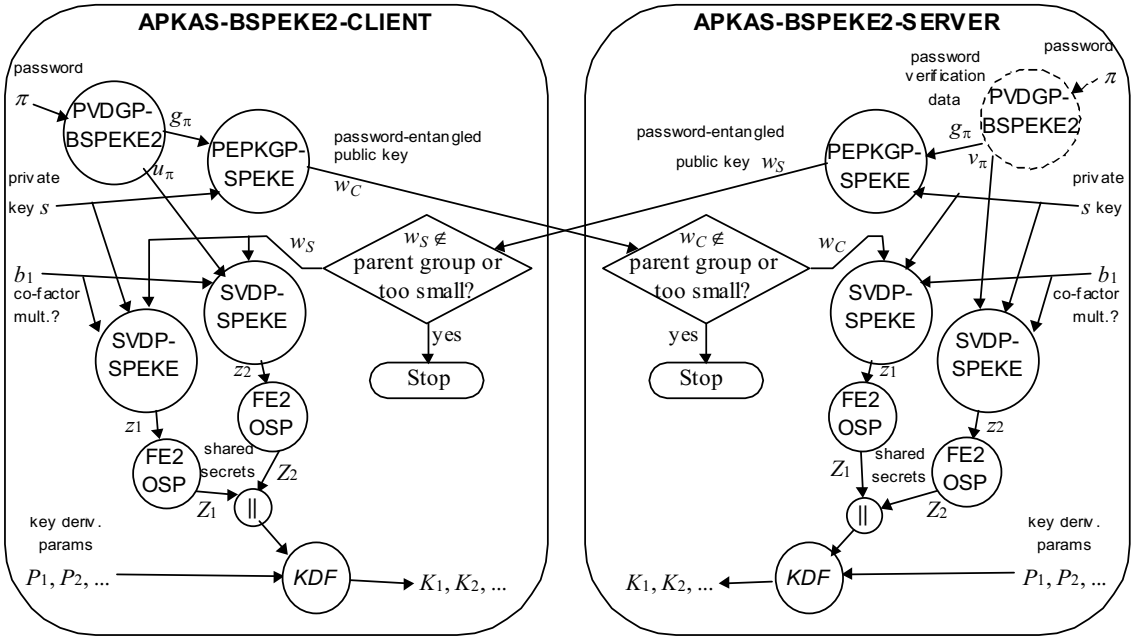

Figure 5—APKAS-BSPEKE2 key agreement operation

9.6.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

For CLIENT only:

— A password-based octet string π

For SERVER only:

— A password verification generator element g_π and password-limited public key v_π that are both derived from {DL,EC}PVDGP-BSPEKE2 using Client's value for π

For both CLIENT and SERVER:

— Primitives for password verification data generation, public-key generation, and secret value derivation, which shall be PVDGP-BSPEKE2, PEPKGP-SPEKE, and SVDP-SPEKE, and their associated parameters

— A set of valid {DL,EC} domain parameters (including q, g, and r) associated with π, v_π, and g_π and keys s, w_C, and w_S.

— A Boolean value b_1 that indicates whether cofactor multiplication is desired

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.6.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps.

9.6.2.1 Key agreement for Client

a) Compute generator element g_π and password-limited private key u_π using {DL,EC}PVDGP-BSPEKE2(π)

b) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

c) Compute password-entangled key w_C using {DL,EC}PEPKGP-SPEKE(s, g_π)

d) Send w_C to the Server

e) Receive password-entangled public-key value w_S from the Server

   1) If w_S is not an element of the parent group, output "invalid" and stop. (See NOTE 3 in 9.6.3.2.)

   2) If the order of w_S is unacceptably small, output "invalid" and stop. (See NOTE 3 in 9.6.3.2.)

   3) *(Optional)* If w_S is not a valid public key, output "invalid" and stop. (See NOTE 4 in 9.6.3.2.)

f) Compute field element z_1 using {DL,EC}SVDP-SPEKE(s, w_S, b_1) (See NOTE 4 in 9.6.3.2.)

g) Compute field element z_2 using {DL,EC}SVDP-SPEKE(u_π, w_S, b_1) (See NOTE 4 in 9.6.3.2.)

h) Compute octet string Z_1 = FE2OSP(z_1)

i) Compute octet string Z_2 = FE2OSP(z_2)

j) Let $Z = Z_1 \| Z_2$

k) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

l) Output derived keys K_1, K_2, ... K_t

9.6.2.2 Key agreement for Server

a) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

b) Compute w_S using {DL,EC}PEPKGP-SPEKE(s, g_π)

c) Send w_S to the Client

d) Receive password-entangled public-key value w_C from the Client

   1) If w_C is not an element of the parent group, output "invalid" and stop. (See NOTE 3 in 9.6.3.2.)

   2) If the order of w_C is unacceptably small, output "invalid" and stop. (See NOTE 3 in 9.6.3.2.)

   3) *(Optional)* If w_C is not a valid public key, output "invalid" and stop. (See NOTE 4 in 9.6.3.2.)

e) Compute field element z_1 using {DL,EC}SVDP-SPEKE(s, w_C, b_1) (See NOTE 4 in 9.6.3.2.)

f) Compute field element z_2 using {DL,EC}SVDP-SPEKE(s, v_π, b_1) (See NOTE 4 in 9.6.3.2.)

g) Compute octet string Z_1 = FE2OSP(z_1)

h) Compute octet string Z_2 = FE2OSP(z_2)

i) Compute $Z = Z_1 \| Z_2$

j) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

k) Output derived keys K_1, K_2, ... K_t

9.6.3 Key confirmation operation

It is optional in APKAS-BSPEKE2 for either the Client or the Server to confirm the other party's knowledge of the shared secret Z, before the party uses Z or any derived shared secrets K_i for other purposes (see NOTE 1 in 9.6.3.2).

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.6.3.1 Key confirmation for Server

a) *Optional*

1) Compute o_π = GE2OSP-X(g_π)

2) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

3) Send C_S to the Client

b) *Optional*

1) Receive octet string C_C from the Client

2) Compute o_π = GE2OSP-X(g_π)

3) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

4) If $C_C' \neq C_C$, output "invalid" and stop.

9.6.3.2 Key confirmation for Client

a) *Optional*

1) Receive octet string C_S from the Server

2) Compute o_π = GE2OSP-X(g_π)

3) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

4) If $C_S' \neq C_S$, output "invalid" and stop.

b) *Optional*

1) Compute o_π = GE2OSP-X(g_π)

2) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

3) Send C_C to the Server

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—APKAS-BSPEKE2 is a bilateral commitment scheme, where both parties provide a commitment to the password (see D.2.1.9).

NOTE 2—These schemes, with appropriate restrictions on the scheme options and inputs, represent a specific use of, and may be compatible with, the techniques in IEEE Std 1363 DLKAS-DH2 and ECKAS-DH2, ANSI X9.42 (see IEEE Std 1363-2000, [B8]) (in the DL case), and ANSI X9.63 (see IEEE Std 1363-2000, [B12]) (in the EC case).

NOTE 3—The Client and Server may test the received public-key value (w_S or w_C) and abort if it is not a valid public key. However, such a test is more than sufficient for compliance. The acceptable values are the elements of the parent group that generate a sufficiently large group. See D.2.2.2.1 for discussion of validating acceptable received public-key values for the SPEKE schemes, and the meaning of "sufficiently large."

NOTE 4—When b_1 indicates that cofactor multiplication *is not* desired, SVDP-SPEKE may be replaced by SVDP-DHC (see IEEE Std 1363a-2004, 6.2.2 and 7.2.2) with an indication that compatibility with SVDP-DH *is* desired, or, when the optional key agreement step is used to abort if w_C or w_S is not a valid public key [Server step d3), Client step e3)], SVDP-SPEKE may be replaced by SVDP-DH (see IEEE Std 1363a-2004, 6.2.1 and 7.2.1). (SVDP-DH is defined only for valid public keys.) When b_1 indicates that cofactor multiplication *is* desired, SVDP-SPEKE may be replaced by SVDP-DHC with an indication that compatibility with {DL,EC}SVDP-DH *is not* desired. In either case when SVDP-DHC is used as a replacement for SVDP-SPEKE, the GCD of domain parameters k and r shall be equal to 1, and when the output from SVDP-DHC is "invalid public key," the scheme should output "invalid" and stop.

9.7 APKAS-PAKZ

{DL,EC}APKAS-PAKZ-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} augmented password-authenticated key agreement scheme, version PAKZ for {Client, Server}. It is based on the work of MacKenzie [B43] and Gentry, MacKenzie, Ramzan [B18].

Figure 6 illustrates the key agreement operation, and Figure 7 illustrates the key confirmation operation of the scheme.

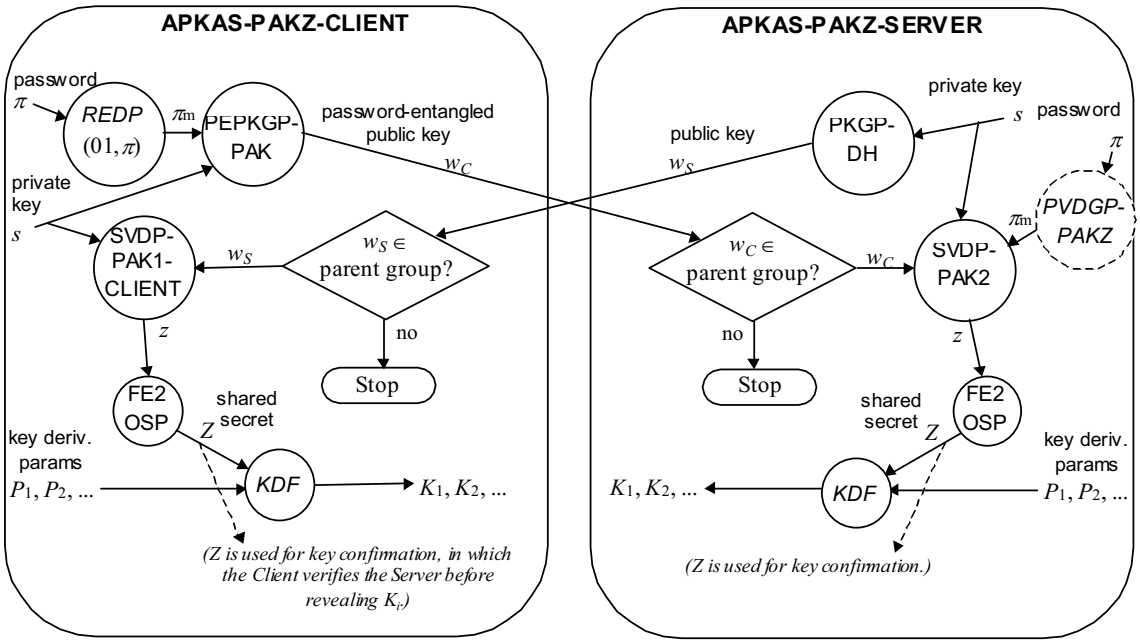

Figure 6—APKAS-PAKZ key agreement operation

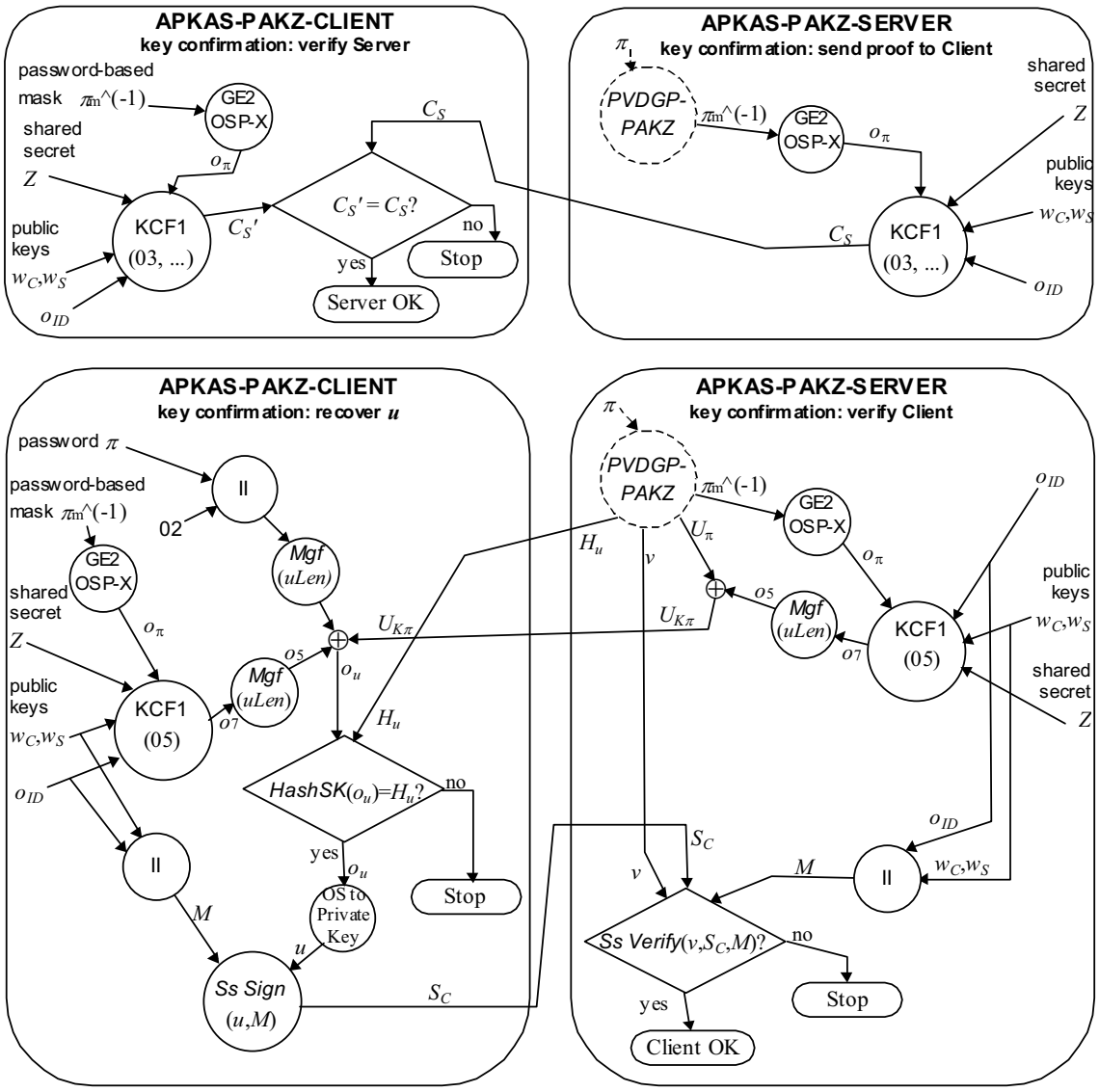

Figure 7—APKAS-PAKZ key confirmation operation

9.7.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for password verification data generation, public-key generation and secret value derivation, which shall be PVDGP-PAKZ, PEPKGP-PAK, PKGP-DH, SVDP-PAK1-CLIENT, and SVDP-PAK2, and their associated parameters

— A set of valid {DL,EC} domain parameters (including k, q, r, and g), associated with the values π (see NOTE 7 in 9.7.3.2) and π_m and keys s, w_C, and w_S as defined below

— A signature scheme *Ss* associated with a private key u and corresponding public key v, which should be selected from the schemes specified in IEEE Std 1363a-2004, Clause 10, which defines or is otherwise associated with:

  - A signature generation operation *Sign*(*u*,*M*) that outputs a signature S_C of a message octet string *M*,
  - A signature verification operation *Verify*(*v*,S_C,*M*) that outputs "valid" or "invalid,"
  - A format for representing *Ss* private key *u* as a fixed-length string of *uLen* octets, which should be a format described in 12.4 that defines *uLen* in accordance with associated signature scheme options and parameters (see NOTE 5 in 9.7.3.2), and
  - If *Ss* is a DL or EC scheme, *Ss* is associated with the same domain parameters (including *k*, *q*, *r*, and *g*).

- A mask generation function *Mgf*, which should be MGF1 (see 12.2.1). The same *Mgf* parameter shall be used with {DL,EC}PVDGP-PAKZ.
- A hash function *HashSK*, which should be chosen from the hash functions in 14.1. The same *HashSK* parameter shall be used with {DL,EC}PVDGP-PAKZ.
- A message parameter octet string o_{ID} (See NOTE 7 in 9.7.3.2.)
- A key derivation function *Kdf*, which should be KDF1 or KDF2
- One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys
- A key confirmation function *Kcf*, which should be KCF1

For CLIENT only:

- A password-based octet string π

For SERVER only:

- Password verification data values that were derived from {DL,EC}PVDGP-PAKZ(π), using the Client's values for π and mask generation function *Mgf*, including:
- A password-based mask group element π_m (see NOTE 4 in 9.7.3.2),
- The public key *v* associated with the signature scheme *Ss*,
- An octet string U_π of length *uLen* containing the password-masked value of o_u, where o_u is the agreed octet string representation of the private key *u* that corresponds to *v*, and
- The hashed private key octet string H_u containing *HashSK*(o_u).

9.7.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps:

a) *For CLIENT only*: Compute a mask group element π_m using step a) of {DL,EC}PVDGP-PAKZ(π)

b) Perform the *Key agreement operation* steps b) through i) for Client and steps b) through i) for Server as specified for {DL,EC}BPKAS-PAK-{CLIENT,SERVER} in 9.2.2 to derive keys K_1, K_2, ... K_t

NOTE—The Client shall confirm the Server's knowledge of shared secret *Z* before any derived keys are used. Key confirmation for APKAS-PAKZ is described in 9.7.3.

c) Output derived keys K_1, K_2, ... K_t

9.7.3 Key confirmation operation

The APKAS-PAKZ Client shall confirm the Server's knowledge of the shared secret Z, recover private key u, and verify that u is correct, before the Client uses Z or any derived shared secrets K_i for other KCFs or other purposes (see NOTE 1 in 9.7.3.2). The APKAS-PAKZ Server shall confirm the Client's knowledge of π, as this provides the augmented advantage of APKAS-PAKZ over BPKAS-PAK.

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.7.3.1 Key confirmation for Client

a) *(Mandatory)* The Client verifies that the Server knows the verification data corresponding to π as follows:

   1) Receive octet string C_S from the Server
   2) Compute o_π = GE2OSP-X(π_m^(−1))
   3) Compute $C_S' = Kcf(hex(03) \| o_{ID}, w_C, w_S, Z, o_\pi)$
   4) If $C_S' \neq C_S$, output "invalid" and stop.

NOTE 1—Step a) shall be performed before the Client uses Z or any derived shared secrets K_i for other purposes.

b) *(Mandatory)* The Client recovers private signature key u and verifies that it is correct as follows:

   1) Receive octet string $U_{K\pi}$, a hidden and password-masked private signature key, and hashed private key octet string H_u from the Server
   2) Compute $o_7 = Kcf(hex(05) \| o_{ID}, w_C, w_S, Z, o_\pi)$
   3) Compute $o_5 = Mgf(o_7, uLen)$
   4) Compute an octet string representation of the signature private key $o_u = U_{K\pi} \oplus o_5 \oplus Mgf(hex(02) \| \pi, uLen)$
   5) If $HashSK(o_u) \neq H_u$, output "invalid" and stop.
   6) Compute signature private key u from the agreed octet string representation o_u
   7) *(Optional)* If u is not a valid private key for the *Sign* operation, output "invalid" and stop. (See NOTE 6 in 9.7.3.2.)

NOTE 2—Step b5) shall be performed before the Client reveals information derived from u.

c) *(Mandatory)* The Client proves knowledge of π to the Server as follows:

   1) Compute $M = o_{ID} \| GE2OSP\text{-}X(w_C) \| GE2OSP\text{-}X(w_S)$
   2) Compute signature $S_C = Sign(u, M)$
   3) Send S_C to the Server

9.7.3.2 Key confirmation for Server

a) *(Mandatory)* The Server proves to the Client knowledge of the verification data corresponding to π as follows:

   1) Compute o_π = GE2OSP-X(π_m^(−1))

2) Compute $C_S = Kcf(hex(03) \| o_{ID}, w_C, w_S, Z, o_\pi)$

3) Send C_S to the Client

b) *(Mandatory)* The Server verifies that the Client's knows π as follows:

1) Compute $o_7 = Kcf(hex(05) \| o_{ID}, w_C, w_S, Z, o_\pi)$

2) Compute $o_5 = Mgf(o_7, uLen)$

3) Compute $U_{K\pi} = U_\pi \oplus o_5$

4) Send $U_{K\pi}$ and H_u to the Client

5) Receive signature S_C from the Client (See NOTE 8.)

6) Compute $M = o_{ID} \| \text{GE2OSP-X}(w_C) \| \text{GE2OSP-X}(w_S)$

7) If $Verify(v, S_C, M)$ = "invalid," output "invalid" and stop.

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—APKAS-PAKZ is a unilateral commitment scheme, where the Server does not provide a commitment to the password during the key agreement operations. In using this scheme, the Client needs to verify the Server's proof of knowledge of the password-based value π_m before revealing any information derived from the output derived keys. See D.2.1.9 for discussion of the limitations on the use of unilateral commitment schemes in application protocols.

NOTE 2—In this scheme, the test for a valid {DL,EC} password-entangled public key is the same as the test for a {DL,EC} public key. That is, a public key w is presumed valid if it specifies an element of order r in the desired group defined by the {DL,EC} domain parameters.

NOTE 3—The steps for the Server to verify the Client's key are mandatory because they are needed to achieve the augmented benefit of APKAS-PAKZ.

NOTE 4—The APKAS-PAKZ Server may want to additionally store π_m^(−1) to optimize efficiency.

NOTE 5—The parties shall agree on a fixed size *uLen* for the length of the agreed octet string format of signature private keys. When using a {DL,EC} signature scheme, *uLen* depends on the scheme option domain parameter r for the recommended private key format (see 12.4.1). When using an IF signature scheme with the recommended private key format (see 12.4.2), the parties need to agree on the size of the IF modulus n associated with (u, v), and agree on a specific representation of private keys as an ordered set of integers (see IEEE Std 1363-2000, 8.1.3), in order to determine *uLen*.

NOTE 6—The check for u being a valid private key in Client key confirmation step b7) is not necessary for the security of the scheme, but it may be needed to ensure that $Sign(u, M)$ is a well-defined operation. Note, however, that IEEE Std 1363-2000 defines IF signature scheme private key operations as assuming that the private key is valid, but it does not define a validation method.

NOTE 7—To be aligned with the PAK–Z+ protocol described in Gentry, MacKenzie, and Ramzan [B18], message parameter o_{ID} should include identifiers for the client and server, and password value π should include o_{ID}.

NOTE 8—Formats for representing signatures as octet strings are given in IEEE Std 1363a-2004, E.3.

9.8 [DL] APKAS-SRP3, APKAS-SRP6

DLAPKAS-{SRP3,SRP6}-{CLIENT,SERVER} is discrete logarithm augmented password-authenticated key agreement scheme, version {SRP3,SRP6} for {Client,Server}. APKAS-SRP3 is based on the work of Wu [B57] and IETF RFC 2945 [B24]. APKAS-SRP6 is based on the work of Wu [B56] and [B57].

DLAPKAS-{SRP3,SRP6} creates a shared key between two parties using an optimized form of dual Diffie-Hellman exchange.

Figure 8 illustrates APKAS-SRP3, and Figure 9 illustrates APKAS-SRP6.

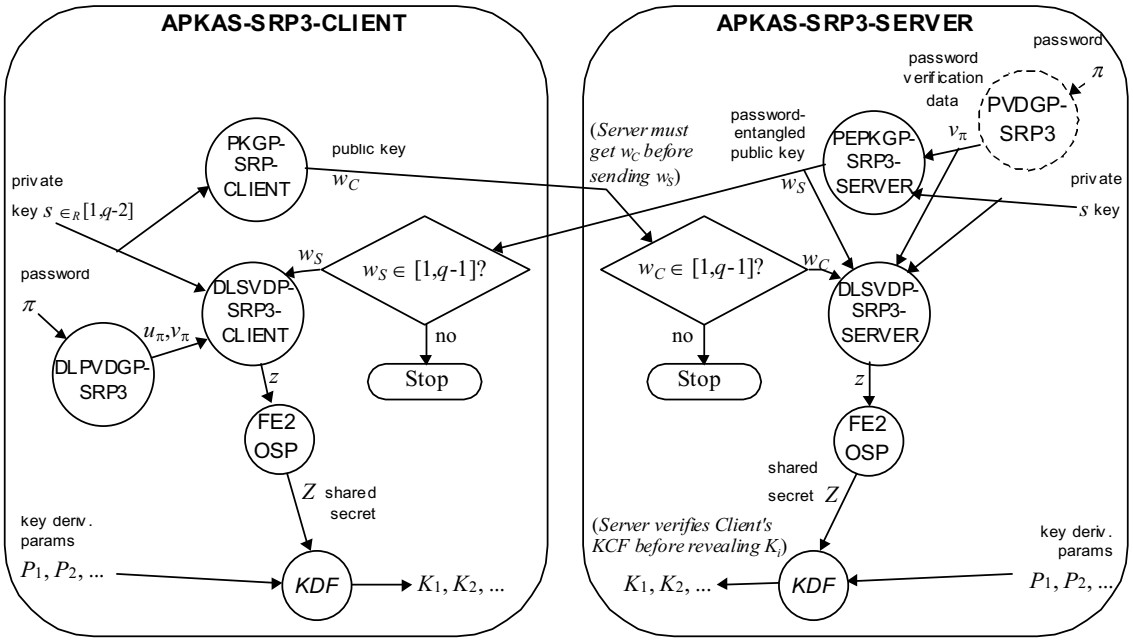

Figure 8 —APKAS-SRP3 key agreement operation

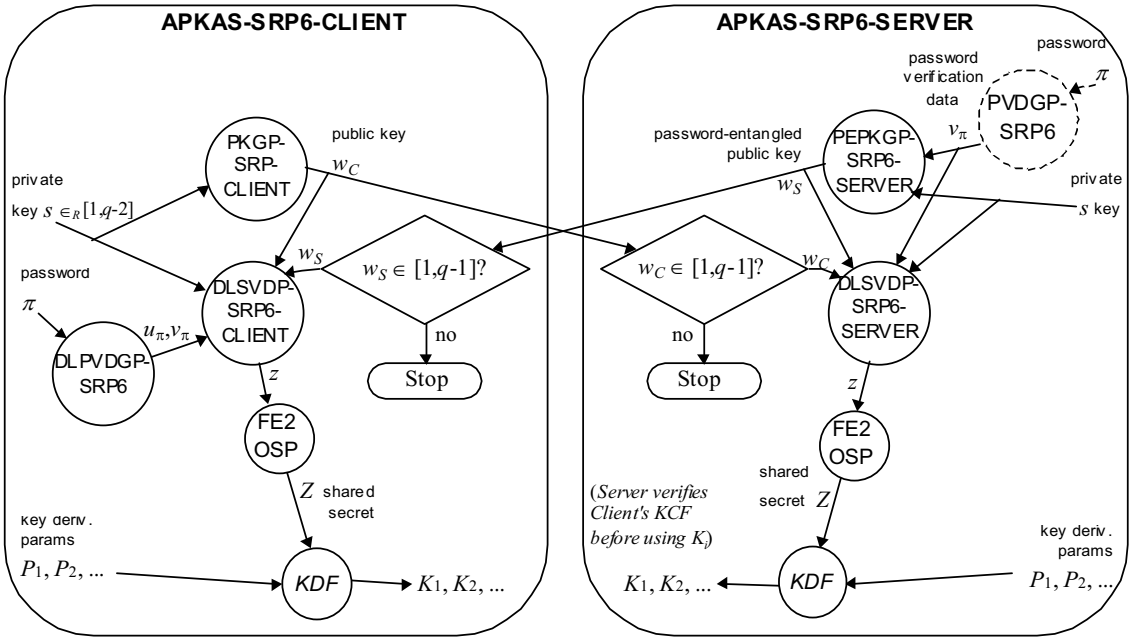

Figure 9—APKAS-SRP6 key agreement operation

9.8.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for password verification data generation, public-key generation, and secret value derivation, which shall be DLPVDGP-{SRP3,SRP6}, DLPKGP-SRP-CLIENT, DLPEPKGP-{SRP3,SRP6}-SERVER, DLSVDP-{SRP3,SRP6}-CLIENT, DLSVDP-{SRP3,SRP6}-SERVER, and their associated parameters. The SRP6 Client and Server shall use the same *HashW2* parameters with DLSVDP-SRP6-CLIENT and DLSVDP-SRP6-SERVER, and shall use the same *Mvcf* parameters with DLSVDP-SRP6-CLIENT and DLPEPKGP-SRP6-SERVER.

For CLIENT only:

— A password-based octet string π, which may also include salt and identifying information for one or both parties

For SERVER only:

— A password-limited public key v_π that was derived from Client's value for π using DLPVDGP-{SRP3,SRP6}

For both CLIENT and SERVER:

— A set of valid DL domain parameters (including g_{q-1}, q, r, and k) associated with π and v_π and keys s, w_C, and w_S

— A key derivation function *Kdf*, which should be KDF1 or KDF2 (*For SRP3*: See NOTE 5 in 9.8.3.2.)

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.8.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps:

9.8.2.1 Key agreement for Client

a) Obtain private key s, a random integer in the range $[1, q-2]$ (See D.2.1.16.)

b) Compute a DL public key w_C = DLPKGP-SRP-CLIENT(s)

c) Send public key w_C to the Server

d) Receive password-entangled public-key value w_S from the Server

   1) If w_S is not in the parent group, output "invalid" and stop. (See NOTE 7 in 9.8.3.2.)

e) Compute password-limited private u_π and password verification data v_π using DLPVDGP-{SRP3,SRP6}(π)

f) Compute agreed field element z as follows:

   1) *For SRP3 only*: Compute z using DLSVDP-SRP3-CLIENT(s, u_π, v_π, w_S)

   2) *For SRP6 only*: Compute z using DLSVDP-SRP6-CLIENT(s, u_π, v_π, w_C, w_S)

g) Compute octet string Z = FE2OSP(z)

h) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

i) Output derived keys K_1, K_2, ... K_t

9.8.2.2 Key agreement for Server

a) Obtain private key s, a random integer in the range [1, q−2] (See D.2.1.16)

b) Compute password-entangled public key w_S using DLPEPKGP-{SRP3,SRP6}-SERVER(s, v_π)

c) Receive public-key value w_C from the Client

1) If w_C is not in the parent group, output "invalid" and stop. (See NOTE 7 in 9.8.3.2.)

NOTE 1—For SRP3 only: The Server shall receive the Client's public-key value w_C before sending password-entangled public key w_S. (See NOTE 4 in 9.8.3.2.)

d) Send password-entangled public key w_S to the Client

e) Compute agreed field element z using DLSVDP-{SRP3,SRP6}-SERVER(s, v_π, w_C, w_S)

f) Compute octet string Z = FE2OSP(z)

g) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

NOTE 2—The Server shall confirm the Client's knowledge of shared secret Z before any derived keys are used. Key confirmation is described in 9.8.3.

h) Output derived keys K_1, K_2, ... K_t

9.8.3 Key confirmation operation

The APKAS-{SRP3,SRP6} Server shall confirm the Client's knowledge of the shared secret Z, before the Server uses Z or any derived shared secrets K_i for other KCFs or other purposes (see NOTE 1 in 9.8.3.2). Explicit confirmation of the Server's knowledge of Z to the Client is optional.

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.8.3.1 Key confirmation for Client

a) *Mandatory*

1) Compute octet string o_π = FE2OSP(v_π)

2) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

3) Send C_C to the Server

b) *Optional*

1) Receive octet string C_S from the Server

2) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

3) If $C_S' \neq C_S$, output "invalid" and stop.

9.8.3.2 Key confirmation for Server

a) *Mandatory*

1) Compute octet string o_π = FE2OSP(v_π)

2) Receive octet string C_C from the Client

3) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, o_\pi)$

4) If $C_C' \neq C_C$, output "invalid" and stop.

NOTE—Step a) shall occur before step b), and before any other use of shared keys K_i that are derived from the Server's key agreement operation.

b) *Optional*

1) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, o_\pi)$

2) Send C_S to the Client

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—DLAPKAS-{SRP3,SRP6} is a unilateral commitment scheme, where the Client does not provide a commitment to the password during the key agreement operations. In using this scheme, the Server needs to verify the Client's proof of knowledge of the agreed key before revealing any information derived from the agreed key. See D.2.1.9 for discussion of the limitations on the use of unilateral commitment schemes in application protocols.

NOTE 2—In contrast with most of the other methods in this document, DLAPKAS-{SRP3,SRP6} and its supporting primitives are defined for use only in the DL setting, and in terms of using field elements in $GF(q)$, rather than as multiplicative subgroup elements of order r.

NOTE 3—A limitation of SRP3 is that an adversary posing as the Server can verify two guesses for the password in a single run. This two-for-one guessing limitation can be exploited by an adversary who masquerades as the Server and verifies two guesses, π' and π'', in one run by sending w_S = exp(g_{q-1}, OS2IP(SHA-1(π'))) + exp(g_{q-1},OS2IP (SHA-1(π''))) (see Wu [B56], Scott [B51], and MacKenzie [B42]). In SRP6, the use of field element multiplier m_v in DLSVDP-SRP6-CLIENT and DLPEPKGP-SRP6-SERVER prevents this two-for-one limitation (see Wu [B56]), when m_v is not a known exponential power of g_{q-1}.

NOTE 4—A further limitation of SRP3 is that the Server shall receive the Client's public-key value w_C before sending his password-entangled public key w_S. SRP6 does not have this limitation, due to the incorporation of w_C in DLSVDP-SRP6-CLIENT, as explained in Wu [B56].

NOTE 5—The DLAPKAS-SRP3 key agreement operations, with the exception of the KDF, are compatible with IETF RFC 2945 [B24]. RFC 2945 uses different key derivation and KCFs.

NOTE 6—APKAS-SRP6 is roughly comparable to Key Agreement Mechanism 2 in ISO/IEC 11770-4:2006 [B26]. However, significant differences prevent interoperability of these methods, such as a different ordering of g_{q-1} and q in the hashed multiplier value, and different conversion primitives that prevent interoperability when using small values of g_{q-1}. See D.2.1.23 for more discussion of differences between these standards.

NOTE 7—For DLAPKAS-{SRP3,SRP6}-{CLIENT,SERVER}, an *acceptable* public-key value w_C or password-entangled public-key value w_S is any element of $GF(q)$ other than zero.

9.9 [EC] APKAS-SRP5

ECAPKAS-SRP5-{CLIENT,SERVER} is elliptic curve augmented password-authenticated key agreement scheme, version SRP5 for {Client, Server}. It is based on the work of Wang [B55] and Wu [B57]. ECAPKAS-SRP5 derives one or more keys from an optimized password-authenticated Diffie-Hellman

exchange. The Client uses a password-based octet string π and the Server uses password verification data created using an appropriate PVDGP function of the Client's password.

Figure 10 illustrates the scheme.

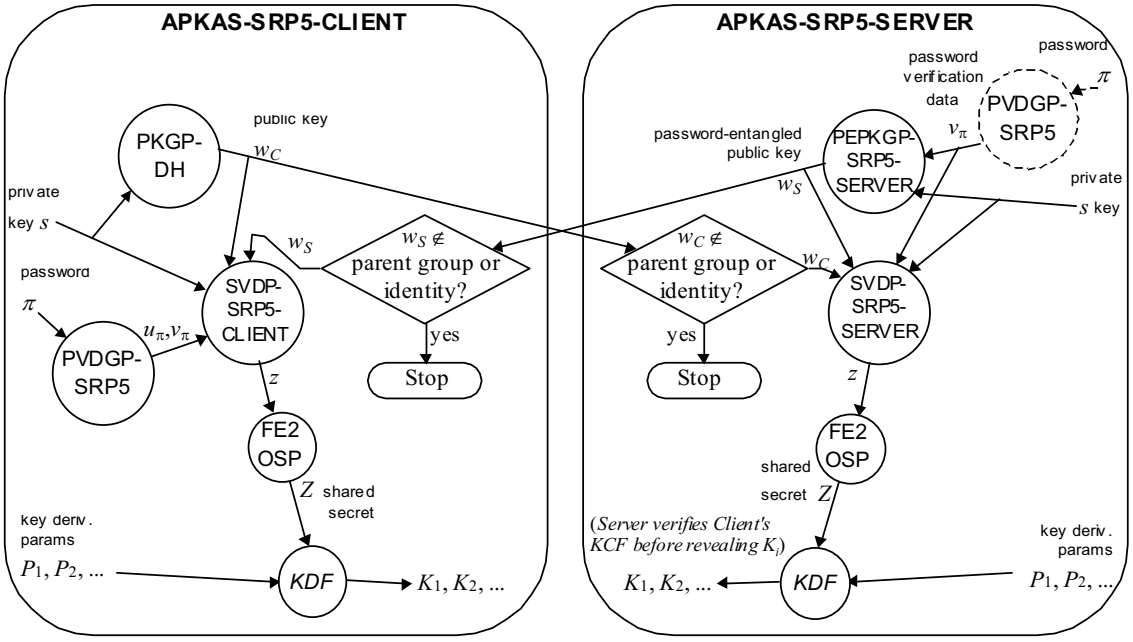

Figure 10—APKAS-SRP5 key agreement operation

9.9.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for password verification data generation, public-key generation, and secret value derivation, which shall be ECPVDGP-SRP5, ECPKGP-DH, ECPEPKGP-SRP5-SERVER, ECSVDP-SRP5-CLIENT, ECSVDP-SRP5-SERVER, and their associated parameters. The Client and Server shall use the same *HashW2* parameters with ECSVDP-SRP5-CLIENT and ECSVDP-SRP5-SERVER, and shall use the same *Redp* parameters with ECSVDP-SRP5-CLIENT and ECPEPKGP-SRP5-SERVER.

For CLIENT only:

— A password-based octet string π, which may also include salt and identifying information for one or both parties.

For SERVER only:

— A password-limited public key v_π that was generated using ECPVDGP-SRP5 with the same parameters and input values as used in the Client's key agreement operation.

For both CLIENT and SERVER:

— A set of valid EC domain parameters associated with π and v_π and keys s, w_C, and w_S.

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.9.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps.

9.9.2.1 Key agreement for Client

a) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

b) Generate a public key w_C using ECPKGP-DH(s)

c) Send w_C to the Server

d) Receive password-entangled public-key value w_S from the Server

   1) If w_S is not a non-identity element of the parent group, output "invalid" and stop.

e) Compute password-limited private key u_π, and password verification element v_π using ECPVDGP-SRP5(π)

f) Derive field element z = ECSVDP-SRP5-CLIENT(s, u_π, v_π, w_C, w_S)

g) Compute octet string Z = FE2OSP(z)

h) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

i) Output derived keys K_1, K_2, ... K_t

9.9.2.2 Key agreement for Server

a) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

b) Generate a password-entangled public key w_S using ECPEPKGP-SRP5-SERVER(s, v_π)

c) Send w_S to the Client

d) Receive public-key value w_C from the Client

   1) If w_C is not a non-identity element of the parent group, output "invalid" and stop.

e) Compute field element z using ECSVDP-SRP5-SERVER(s, v_π, w_C, w_S)

f) Compute octet string Z = FE2OSP(z)

g) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

NOTE—The Server shall confirm the Client's knowledge of shared secret Z before any derived keys are used. Key confirmation is described in 9.9.3.

h) Output derived keys K_1, K_2, ... K_t

9.9.3 Key confirmation operation

The ECAPKAS-SRP5 Server shall confirm the Client's knowledge of the shared secret Z, before the Server uses Z or any derived shared secrets K_i for other KCFs or other purposes (see NOTE 1 in 9.9.3.2). Explicit confirmation of the Server's knowledge of Z to the Client is optional.

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.9.3.1 Key confirmation for Client

a) *Mandatory*

1) Compute octet string o_π = FE2OSP(v_π)
2) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, o_\pi)$
3) Send C_C to the Server

b) *Optional*

1) Receive octet string C_S from the Server
2) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, o_\pi)$
3) If $C_S' \neq C_S$, output "invalid" and stop.

9.9.3.2 Key confirmation for Server

a) *Mandatory*

1) Compute octet string o_π = FE2OSP(v_π)
2) Receive octet string C_C from the Client
3) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, o_\pi)$
4) If $C_C' \neq C_C$, output "invalid" and stop.

NOTE—Step a) shall occur before step b), and before any other use of shared keys K_i that are derived from the Server's key agreement operation.

b) *Optional*

1) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, o_\pi)$
2) Send C_S to the Client

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—ECAPKAS-SRP5 is a unilateral commitment scheme, where the Client does not provide a commitment to the password during the key agreement operations. In using this scheme, the Server needs to verify the Client's proof of knowledge of the agreed key before revealing any information derived from the agreed key. See D.2.1.9 for discussion of the limitations on the use of unilateral commitment schemes in application protocols.

9.10 APKAS-WSPEKE

{DL,EC}APKAS-WSPEKE-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} augmented password-authenticated key agreement scheme, version WSPEKE for {Client, Server}. It is based on the work of Wu [B57], Jablon [B27], [B28], and [B30]. APKAS-WSPEKE derives one or more keys from an

optimized password-authenticated Diffie-Hellman exchange. The Client uses a password-based octet string π and the Server uses password verification data created using an appropriate PVDGP function of the Client's password.

Figure 11 illustrates the scheme.

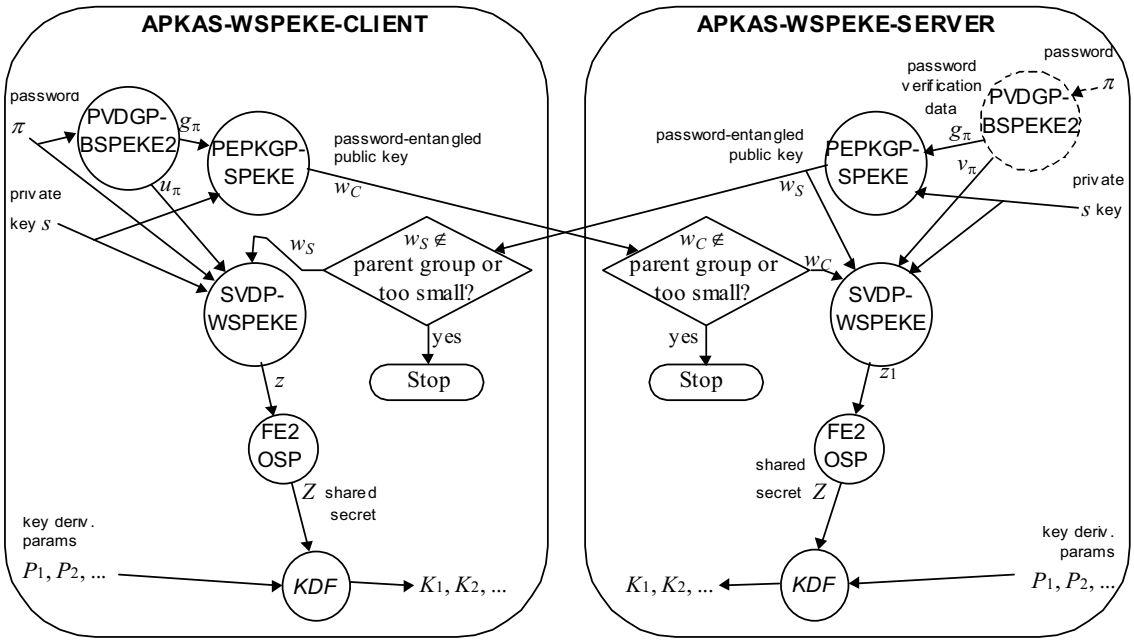

Figure 11—APKAS-WSPEKE key agreement operation

9.10.1 Scheme options

Both the Client and Server parties shall establish or otherwise agree upon the following options:

— Primitives for password verification data generation, public-key generation, and secret value derivation, which shall be PVDGP-BSPEKE2, PEPKGP-SPEKE, SVDP-WSPEKE-CLIENT, SVDP-WSPEKE-SERVER, and their associated parameters. The Client and Server shall use the same *HashWS* parameters with SVDP-WSPEKE-CLIENT and SVDP-WSPEKE-SERVER.

For CLIENT only:

— A password-based octet string π, which may also include salt and identifying information for one or both parties.

For SERVER only:

— A password verification generator element g_π and password-limited public key v_π that were both generated using {DL,EC}PVDGP-BSPEKE2 with the same parameter and input values as used in the Client's key agreement operations

For both CLIENT and SERVER:

— A set of valid {DL,EC} domain parameters associated with π, g_π, and v_π and keys s, w_C, and w_S

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, ...\}$ to be used to derive agreed keys

— A key confirmation function *Kcf*, which should be KCF1

9.10.2 Key agreement operation

A sequence of shared secret keys, K_1, K_2, ... K_t, shall be generated by each party by performing the following or an equivalent sequence of steps.

9.10.2.1 Key agreement for Client

a) Compute generator element g_π using {DL,EC}PVDGP-BSPEKE2(π)

b) Compute password-limited private key u_π using {DL,EC}PVDGP-BSPEKE2(π)

c) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

d) Generate a password-entangled public key w_C using {DL,EC}PEPKGP-SPEKE(s, g_π)

e) Send w_C to the Server

f) Receive a password-entangled public-key value w_S from the Server

   1) If w_S is not in the parent group, output "invalid" and stop.

   2) If order of w_S is unacceptably small, output "invalid" and stop. (See NOTE 2 in 9.10.3.2.)

   3) *(Optional)* If w_S is not a valid public key, output "invalid" and stop. (See NOTE 2 in 9.10.3.2.)

g) Compute field element z using {DL,EC}SVDP-WSPEKE-CLIENT(s, u_π, w_S)

h) Compute octet string Z = FE2OSP(z)

i) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

j) Output derived keys K_1, K_2, ... K_t

9.10.2.2 Key agreement for Server

a) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

b) Generate password-entangled public key w_S using {DL,EC}PEPKGP-SPEKE(s, g_π)

c) Send w_S to the Client

d) Receive a password-entangled public-key value w_C from the Client

   1) If w_C is not in the parent group, output "invalid" and stop.

   2) If order of w_C is unacceptably small, output "invalid" and stop. (See NOTE 2 in 9.10.3.2.)

   3) *(Optional)* If w_C is not a valid public key, output "invalid" and stop. (See NOTE 2 in 9.10.3.2.)

e) Compute field element z using {DL,EC}SVDP-WSPEKE-SERVER(s, v_π, w_C, w_S)

f) Compute octet string Z = FE2OSP(z)

g) For each key derivation parameter P_i , derive a shared secret key K_i from the shared secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

h) Output derived keys K_1, K_2, ... K_t

9.10.3 Key confirmation operation

It is optional in APKAS-WSPEKE for either the Client or the Server to confirm the other party's knowledge of the shared secret Z, before the party uses Z or any derived shared secrets K_i for other purposes (see NOTE 1 in 9.10.3.2).

Key confirmation may be achieved using the following or an equivalent sequence of steps.

9.10.3.1 Key confirmation for Server

a) *Optional*

1) Compute o_π = GE2OSP-X(g_π)
2) Compute $C_S = Kcf(hex(03), w_C, w_S, Z, o_\pi)$
3) Send C_S to the Client

b) *Optional*

1) Receive octet string C_C from the Client
2) Compute o_π = GE2OSP-X(g_π)
3) Compute $C_C' = Kcf(hex(04), w_C, w_S, Z, o_\pi)$
4) If $C_C' \neq C_C$, output "invalid" and stop.

9.10.3.2 Key confirmation for Client

a) *Optional*

1) Receive octet string C_S, a key confirmation message, from the Server
2) Compute o_π = GE2OSP-X(g_π)
3) Compute $C_S' = Kcf(hex(03), w_C, w_S, Z, o_\pi)$
4) If $C_S' \neq C_S$, output "invalid" and stop.

b) *Optional*

1) Compute o_π = GE2OSP-X(g_π)
2) Compute $C_C = Kcf(hex(04), w_C, w_S, Z, o_\pi)$
3) Send C_C to the Server

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—APKAS-WSPEKE is a bilateral commitment scheme, where both parties provide a commitment to the password during the key agreement operations (see D.2.1.9).

NOTE 2—The Client and Server may test the received public-key value (w_S or w_C) and abort if it is not a valid public key. However, such a test is more than sufficient for compliance. The acceptable values are the elements of the parent group that generate a sufficiently large group. See D.2.2.2.1 for discussion of validating acceptable received public-key values for the SPEKE schemes, and the meaning of "sufficiently large."

10. Password-authenticated key retrieval schemes

The general model for a PKRS is given in 10.1. A specific scheme and its allowable options are given in 10.2.

10.1 General model

A PKRS establishes one or more static keys for one party, the *Client*, that has a password value π, in an exchange of messages with (at least one) other party, the *Server*, that has secret data that has been associated with π. Because the Client typically has prior knowledge of the (re)established static keys, these keys are said to be retrieved by the scheme.

To successfully perform the PKRS, the Client needs to know only the value π, and the Server needs to know only the stored data associated with the value π. The Server does not necessarily need to know either π or any resulting static key retrieved by the Client. The Client interacts with the Server using *key establishment operations* to retrieve the keys associated with the password.

A PKRS differs from a PKAS in two ways:

— A PKRS establishes *static* keys, whereas a PKAS establishes *ephemeral* keys.

— A PKRS establishes keys for the Client that are not necessarily known or derivable by the Server, whereas a PKAS establishes keys that are shared by both parties.

A PKRS (such as PKRS-1, defined in 10.2) may be used in multi-server systems (Ford and Kaliski [B16], Jablon [B29]) where a Client uses a password to retrieve a key from key shares that have been distributed and stored with two or more Servers.

Since there is only one PKRS in this standard, the general form of a PKRS, similar to those for APKAS and BPKAS, is not presented here.

10.2 PKRS-1

{DL,EC}PKRS-1-{CLIENT,SERVER} is {discrete logarithm, elliptic curve} password-authenticated key retrieval scheme, version 1 for {Client,Server}. It is based on the work of Ford and Kaliski [B16] and Jablon [B29] (see NOTE 1 in 10.2.2.2). Key establishment operations compute values to be exchanged between the Client and Server, and compute retrieved static keys for the Client derived from both the Client's password value π and the Server's associated private key.

The scheme prevents both a bogus Client (that does not have π), and a third-party observer (of the values exchanged between a legitimate Client and Server), from being able to derive the password, the Server's associated key, or the retrieved keys, even with an off-line brute-force password attack. In addition, the Server's associated key does not, in itself, contain sufficient information to permit the Server to determine the Client's password or retrieved keys.

PKRS-1 does not describe key confirmation operations, although such operations are generally required in an invoking application (see NOTE 6 in 10.2.2.2).

PKRS-1 may be used in application protocols where a Client interacts with two or more Servers, to reduce vulnerabilities due to one or more compromised Servers. See D.2.2.3.4 for further discussion of applications of PKRS-1.

Figure 12 illustrates the scheme.

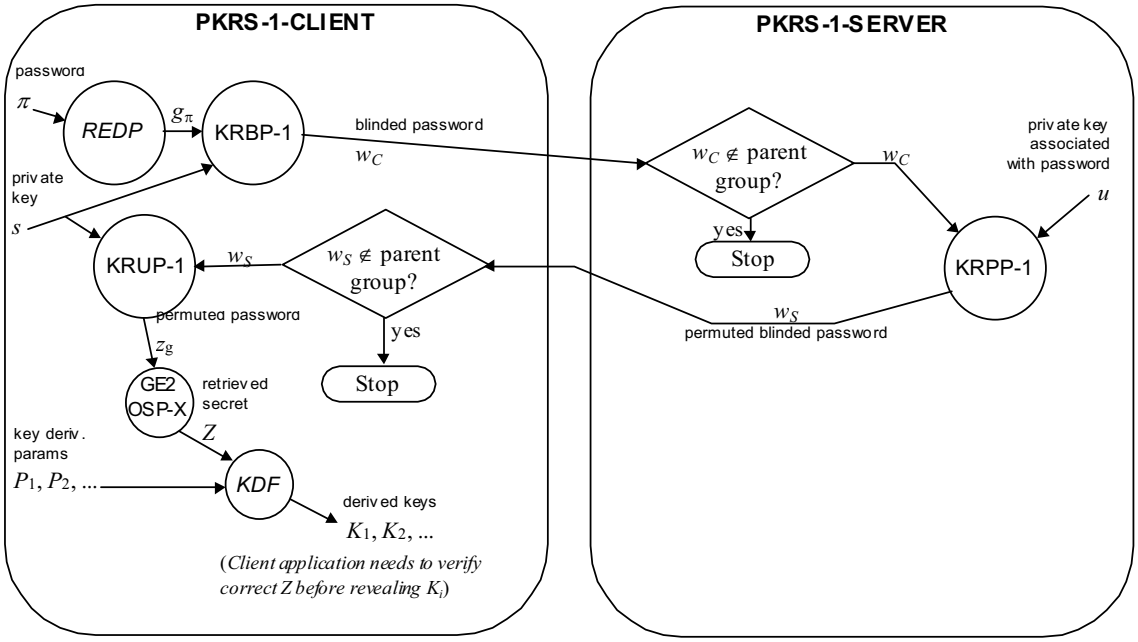

Figure 12—PKRS-1 key establishment operation

10.2.1 Scheme options

Both the Client and Server shall establish or otherwise agree upon the following options:

For CLIENT only:

— A password-based octet string π, known by Client, which may also include salt and identifying information for one or both parties

— A REDP function *Redp*, which should be {DL,EC}REDP-1 or {DL,EC}REDP-2

— A key derivation function *Kdf*, which should be KDF1 or KDF2

— One or more key derivation parameter octet strings $\{P_1, P_2, \ldots\}$ to be used to derive agreed keys

For SERVER only:

— A static private key u known by Server that corresponds to the value of π known by Client

For both CLIENT and SERVER:

— Primitives for blinding, unblinding, and permutation functions, which shall be KRBP-1, KRUP-1 and KRPP-1, and their associated parameters

— A set of valid {DL,EC} domain parameters (including q and r) associated with π, u, s, w_C, and w_S. (See NOTE 5 in 10.2.2.2.)

10.2.2 Key establishment operation

A sequence of secret keys, K_1, K_2, ... K_t, shall be generated by the Client, with the assistance of the Server, by performing the following or an equivalent sequence of steps.

10.2.2.1 Key establishment for Client

a) Compute a generator group element $g_\pi = Redp(\pi)$

b) Obtain private key s, a random integer in the range [1, r–1] (See D.2.1.16.)

c) Compute blinded password group element w_C using {DL,EC}KRBP-1(s, g_π)

d) Send blinded password w_C to the Server

NOTE 1—Step d) shall occur before step e), since the Server uses w_C to compute w_S.

e) Receive permuted blinded password value w_S from the Server

1) If w_S is not in the parent group, output "invalid" and stop. (See NOTE 2 in 10.2.2.2.)

2) *(Optional)* If w_S is not a valid public key, output "invalid" and stop. (See NOTE 3 in 10.2.2.2.)

f) Compute permuted password z_g = {DL,EC}KRUP-1(s, w_S)

NOTE 2—The application needs to confirm that the value of z_g is correct before any information about z_g is revealed to other parties. (See NOTE 6 in 10.2.2.2.)

g) Compute octet string Z = GE2OSP-X(z_g)

h) For each key derivation parameter P_i , derive a secret key K_i from the secret octet string Z and P_i using $K_i = Kdf(Z, P_i)$.

i) Output derived keys K_1, K_2, ... K_t

10.2.2.2 Key establishment for Server

a) Receive blinded password value w_C from the Client

1) If w_C is not in the parent group, output "invalid" and stop. (See NOTE 4.)

2) *(Optional)* If w_C is not a valid public key, output "invalid" and stop. (See NOTE 5.)

b) Compute permuted blinded password group element w_S = {DL,EC}KRPP-1(u, w_C)

c) Send permuted blinded password w_S to the Client

Conformance region recommendation: A conformance region should include limitations for any input values as discussed in Annex B.

NOTE 1—There are differences between PKRS-1 and the methods in the referenced papers, Ford and Kaliski [B16] and Jablon [B29], which both describe multi-server application protocols that use an underlying PKRS. The referenced methods use specific DL $GF(p)$ settings, whereas PKRS-1 is generalized for those and other DL and EC settings. When used with identical domain parameters, PKRS-1 is compatible with, but not identical to, the comparable Client and Server key establishment operations in the referenced methods; PKRS-1 explicitly rejects values of w_C or w_S that are not in the parent group (see NOTE 2 and NOTE 4), since the underlying primitives are not generally defined for such values.

NOTE 2—The Client key establishment operation aborts when the received value w_S is not in the parent group in order to make the step of invoking KRUP-1 a well-defined operation. See D.2.2.3.2 for further discussion.

NOTE 3—A value for w_S that is merely a *valid* public key does not ensure a *correct* value for z_g (see NOTE 6). Thus, merely making the Client abort when w_S is an invalid public key is insufficient to ensure that z_g is correct. Rejecting invalid public keys is permitted, since the correct value for w_S will always be a valid public key.

NOTE 4—The Server aborts when the received value w_C is not in the parent group in order to make the step of invoking KRPP-1 a well-defined operation. See D.2.2.3.3 for further discussion.

NOTE 5—The Server's need to validate w_C during key establishment depends on the domain parameters. It is optional for the Server to abort when the received value w_C is determined to be an invalid public key. This step is unnecessary and may require significant added computation when using certain recommended settings, such as DL $GF(p)$ with a safe prime p. However, this step or some alternative further validation may be necessary to prevent Pohlig-Hellman decomposition attack on the Server's private key u in other settings. See D.2.2.3.3 and D.2.2.3.1 for discussion of validating acceptable values for w_C and the related issues of recommended domain parameter selection, performance, and security.

NOTE 6—PKRS-1 does not describe key confirmation operations. Nevertheless, any application that uses PKRS-1-CLIENT needs to confirm that any established keys are correct before revealing any information about them to other parties to prevent disclosure of π (see D.2.2.3.4).

NOTE 7—*(Domain parameter validation.)* Since the underlying primitives generally assume that the domain parameters (if any) are valid, the result of a primitive may be undefined otherwise. Consequently, it is recommended that the sender validate domain parameters, if any, in step 1, unless the risk of operating on invalid domain parameters is mitigated by other means, as discussed further in IEEE Std 1363-2000, D.3.3.

NOTE 8—*(Error conditions.)* The Client's and Server's steps may produce errors under certain conditions, such as the following:

— Domain parameters not valid in Client or Server
— Client finds received permuted blinded password to be not valid
— Server finds received blinded password to be not valid
— Client finds retrieved permuted password to be not valid
— The Server finds no stored private key associated with the blinded password received from the Client

Such error conditions should be detected and handled appropriately by an implementation, but specific methods for detecting and handling them are outside of the scope of this standard.

NOTE 9—PKRS-1 is comparable to Key Retrieval Mechanism 1 in ISO/IEC 11770-4:2006 [B26]. However, differences in conversion primitives prevent interoperability between these methods in EC settings, and prevent interoperability randomly, with non-negligible probability, in DL settings. See D.2.1.23 for more discussion of differences between these standards.

11. Key derivation functions

11.1 KDF1

This function is parameterized by the choice of a hash function, which shall be SHA-1 (12.1.1) or RIPEMD-160 (12.1.2). (See description in IEEE Std 1363-2000, 13.1.)

11.2 KDF2

This function is parameterized by the choice of a hash function, which shall be one of the hash functions in 12.1. (See description in IEEE Std 1363a-2004, 13.2.)

12. Auxiliary techniques

12.1 Hash functions

A hash function is a building block for many techniques described in Clause 6 through Clause 12. A hash function takes a variable-length octet string as input and outputs a fixed-length octet string. The length of the input to a hash function is usually unrestricted or constrained by a very large number. The output depends solely on the input—a hash function is deterministic.

12.1.1 SHA-1

Hash function SHA-1 outputs an octet string of length 20. (See description in IEEE Std 1363a-2004, 14.1.1.)

12.1.2 RIPEMD-160

Hash function RIPEMD-160 outputs an octet string of length 20. (See description in IEEE Std 1363a-2004, 14.1.2.)

12.1.3 SHA-256

Hash function SHA-256 outputs an octet string of length 32. (See description in IEEE Std 1363a-2004, 14.1.3.)

12.1.4 SHA-384

Hash function SHA-384 outputs an octet string of length 48. (See description in IEEE Std 1363a-2004, 14.1.4.)

12.1.5 SHA-512

Hash function SHA-512 outputs an octet string of length 64. (See description in IEEE Std 1363a-2004, 14.1.5.)

12.2 Mask generation functions

A mask generation function (MGF) is a building block for many techniques described in Clause 6 through Clause 9. An MGF takes as input an octet string and the desired length of the output, and outputs an octet string of that length. The lengths of both the input to and the output of an MGF are usually unrestricted or constrained by a very large number. The output depends solely on the input—an MGF is deterministic.

12.2.1 MGF1

This function is parameterized by the choice of a hash function, which shall be one of the hash functions in 12.1, and accepts an input value that specifies the length in octets of the desired output.

MGF1 is used as a hash function in several methods. (See description in IEEE Std 1363a-2004, 14.2.1.)

12.3 Key confirmation functions

A key confirmation function (KCF) is used in a PKAS to create a confirmation message that proves one parties' knowledge of the agreed shared secret key and related data to the other party. The confirmed data input to the KCF may include the public values exchanged during the key agreement operation and pre-shared password-based data.

NOTE—The choice of a KCF for confirming a key within a scheme should be made in light of the KDF that is used to derive the key. Ideally, the design of the KDF and KCF methods should ensure that all the KDF and KCF functions will output computationally independent results. Otherwise, an application might expose a KCF result that compromises the desired secrecy of a KDF result, or vice versa. To prevent this problem it is recommended that KCF1 should be used with either KDF1 or KDF2.

12.3.1 KCF1

KCF1 is key confirmation function, version 1. It is based on the work of MacKenzie [B43].

This function is parameterized by the following choice:

— A hash function *HashKC*, which should be one of the hash functions in 12.1

Input:

— Key Derivation Parameter octet string P

— Client public key w_C

— Server public key w_S

— Shared key octet string Z

— Shared password value octet string o_π

— A set of valid DL or EC domain parameters (including q)

Output: key confirmation octet string o

Operation:

a) Compute o_C = GE2OSP-X(w_C)

b) Compute o_S = GE2OSP-X(w_S)

c) Compute $o = HashKC(P \| o_C \| o_S \| Z \| o_\pi)$

d) Output o

NOTE—The sizes of P (typically one octet), Z, and o_π are determined by the invoking scheme. The sizes of o_C and o_S are the number of significant octets in domain parameter q.

12.4 Converting between private keys and their octet string representations

When a private key needs to be represented as an octet string, it may be done as defined in this subclause.

12.4.1 Converting between DL/EC private keys and octet strings

A {DL,EC} private key (see IEEE Std 1363-2000, 6.1.3 and 7.1.3) is an integer in the range [1, r–1], where r denotes the order of the desired group in the {DL,EC} domain parameters. Let $uLen = \lceil \log_{256} r \rceil$ denote the length of r in octets. The private key may be formatted as an string of *uLen* octets by using the primitive I2OSP.

The private key may be recovered from this formatted octet string representation by using the primitive OS2IP.

12.4.2 Converting between IF private keys and octet strings (IFU2OSP and IFOS2UP)

An IF private key may be represented as an ordered pair, triple, or quintuple of integers, designated by (n, d), (p, q, d), or (p, q, d_1, d_2, c), where each integer is in the range [1, n–1], where n is the modulus in the corresponding IF public key, as described in IEEE Std 1363-2000, 8.1.3. IFU2OSP converts a specific multi-integer representation of an IF private key for a specific size of n into a string of *uLen* octets as follows:

For each integer component of a private key, let $f_i(x) = \text{I2OSP}(x, j_i)$ describe the function for converting integer components into octet string components of length j_i. Let $j_2 = \lceil \log_{256} n \rceil$ denote the lengths (in octets) of the octet string components for n and d, and $j_1 = \lceil j_2/2 \rceil$ denote the lengths of the octet string components for p, q, d_1, d_2, and c. The octet string representation of a private key is formed by concatenating the octet string components, as follows:

— For an integer pair: $f_2(n) \,||\, f_2(d)$, where $uLen = 2j_2$

— For an integer triple: $f_1(p) \,||\, f_1(q) \,||\, f_2(d)$, where $ulen = 2j_1 + j_2$

— For an integer quintuple: $f_1(p) \,||\, f_1(q) \,||\, f_1(d_1) \,||\, f_1(d_2) \,||\, f_1(c)$, where $ulen = 5j_1$

To parse an IFU2OSP-encoded octet string associated with an agreed multi-integer representation of an IF private key, IFOS2UP splits the octet string into the agreed two, three, or five octet string components, with $f_2(n)$ and $f_2(d)$ each of length j_2, and $f_1(p)$, $f_1(q)$, $f_1(d_1)$, $f_1(d_2)$, and $f_1(c)$ each of length j_1, and converts them to integers using OS2IP. Note that it is essential that each of the octet string components be of the specified length, even if it means that they have leading zero octets.

NOTE—When representing integer triples or quintuples, this format assumes that p and q are each less than $2^{8\times\lceil\lceil\log_{256} n\rceil/2\rceil}$. This assumption is true if p and q are of the same size (that is, if $\lfloor \log_2 p \rfloor = \lfloor \log_2 q \rfloor$).

12.5 Multiplier value creation functions for APKAS-SRP6

A multiplier value creation function (MVCF) creates a multiplier value field element that is shared and used by DLSVDP-SRP6-CLIENT when invoked by an APKAS-SRP6 Client and DLPEPKGP-SRP6-SERVER when invoked by the corresponding APKAS-SRP6 Server. The function may take input values that are shared by the Client and Server and outputs a field element that depends solely on the input. The MVCF is designed to prevent a malicious party from selecting and forcing others to use domain parameters with an associated multiplier value m_v that are designed to enable a 2-for-1 guessing attack.

NOTE—See D.2.2.4.4 for discussion of the requirements for *Mvcf.*

12.5.1 MVCF-DP

MVCF-DP is multiplier value creation function version DP. The multiplier value is created as a hash function of the domain parameters shared by the Client and Server.

Input:

— The DL domain parameters (including g_{q-1} and q) of the invoking primitive.

Output: A field element multiplier value m_v.

The multiplier value is computed as follows:

a) Compute octet string o_1 of length $\lceil \log_{256}(q+1) \rceil$ using I2OSP(q)

b) Compute octet string o_2 = FE2OSP(g_{q-1})

c) Compute octet string o_3 = SHA-1($o_1 \| o_2$) (See 12.1.1.)

d) Compute m_v = I2FEP(OS2IP(o_3) mod q)

Annex A

(informative)

Number-theoretic background

A.1 Generation and validation of parameters and keys

A.1.1 Algorithm for generating verifiably pseudo-random DL domain parameters *p* and *r* (prime case)

Several standard methods exist that generate verifiably pseudo-random (see NOTE 5) primes, suitable for use as domain parameters for a DL prime field (see NOTE 1). This method describes how to use such a method (*pGen*) to produce domain parameter primes p and r and associated *SEED* and *counter* output values, with extra steps to ensure that $k/2$ is prime (see NOTE 3) and/or that $(r-1)/2$ is prime (see NOTE 9). *SEED* and *counter* can later be used to verify p and r using A.1.2.

Input:

— An integer L that specifies the size of desired output p in bits ($2^{L-1} < p < 2^{L}$)

— An integer M that specifies the size of desired output r in bits ($2^{M-1} < r < 2^{M}$)

— An indication of whether $k/2$ shall be prime, where $p = kr+1$

— An indication of whether $(r-1)/2$ shall be prime

— A method *pGen*(L', M') that produces a random octet string *SEED* and associated pseudo-random values p, r, and *counter*, where p is an L'-bit prime, r is an M'-bit prime representing the order of a desired subgroup of *GF*(p), and *counter* is an integer that may be used to verify the pseudo-random construction of p and r (see NOTE 1)

Output: Prime integers p and r and associated values for octet string *SEED* and integer *counter*.

The method is defined as follows:

a) Use *pGen*(L, M) to generate p, r, *SEED* and *counter*.

b) If it is indicated that $k/2$ shall be prime, perform a robust primality test (see NOTE 4) on $(p-1)/2r$, and if it is not prime, go to step a).

c) If it is indicated that $(r-1)/2$ shall be prime, perform a robust primality test on $(r-1)/2$, and if it is not prime, go to step a).

d) Output the values p, r, *SEED* and *counter*.

NOTE 1—The FIPS PUB 186-2 method in [B14] Appendix 2.2 generates "kosherized" primes for DSA, and similar methods are described in ANSI X9.30.1:1997 [B1], ANSI X9.42:2003 [B2] 7.1, IETF RFC 2631 [B22] 2.2.1, and the proposed FIPS Draft 186-3 [B15], A.1.1.2. (See IEEE Std 1363-2000, D.4.1.4, for more information.) The FIPS PUB 186-2 method is limited to 1024 bit $p = kr+1$ with 160 bit subgroup order r (see NOTE 2) and does not impose further constraints on the value of k. The X9.42 and RFC 2631 method can produce arbitrary sized p and r, and it can further ensure that $k=2$ (by setting $L = M+1$).

NOTE 2—In FIPS PUB 186-2 [B14], the letter "q" is used to denote the desired subgroup order r.

NOTE 3—When $k = 2$, or when $k>r$ and $k/2$ is prime, the only small order field elements of $GF(p)$ are the elements 1 and $p-1$, which may simplify testing for acceptable public keys in various key agreement and key retrieval schemes.

NOTE 4—Robust primality testing methods are described in IEEE Std 1363-2000, A.15.3, and FIPS PUB 186-2 [B14].

NOTE 5—Meaning of *verifiably pseudo-random.* Dependent on the strength of the selected hash function and the length and source entropy of *SEED*, it would be ideal if the generated values of p and r (in the absence of SEED and counter values) were black-box-indistinguishable from randomly (or pseudo-randomly) generated values of p and r that meet the particular size, divisibility, and primality constraints. Strictly speaking, however, this goal is not attainable; unless *SEED* is chosen deterministically (which would defeat randomness), there is no way to enforce that the *first* chosen value of *SEED* that yields acceptable values of p and r is the one that is submitted for verification. Repeated attempts could be made to generate values of p and r that meet certain additional (perhaps undisclosed) properties. Notably, the countermeasure to the effectiveness of such attempts is the judicious use of the (presumably computationally irreversible) hash function within the generation and verification algorithms. This is intended to make it computationally infeasible to attain highly sparse "trapdoor" properties in p and/or r (see NOTE 7). This eliminates reliance on detection of perhaps unknown trapdoor mechanisms by the verifier.

NOTE 6—It is not assumed that the generation source is implicitly trusted by all relying parties. Relying parties do not necessarily have to individually implement verification, but rather can trust an assignee, such as a particular certification authority, to handle verification.

NOTE 7—The circumstances relative to what constitutes an effective trapdoor can be dependent on the specific application of p and r. For example, the method in Appendix 2.2 of FIPS PUB 186-2 effectively thwarts an attack of Vaudenay [B53] against DSA that is based on constructing r such that two messages hash to the same value modulo r. This attack is different from the case of trapdoors designed to shortcut the discrete log problem itself for a system-wide p, such as pointed out by Lenstra and Haber [B40] relative to the number field sieve, which is also effectively thwarted by these methods.

NOTE 8—Algorithms to construct verifiably pseudo-random elliptic curves are described in IEEE Std 1363-2000 A.12.4 (prime case) and A.12.6 (binary case), FIPS PUB 186-2 [B14], and ANSI X9.62:1998 [B3].

NOTE 9—One of the suggested ways to reduce the threat of SDHP attack in PKRS-1 as described in Brown and Gallant [B9] is to ensure that the value of $(r-1)/2$ is prime (see D.2.2.3.6).

NOTE 10— A standard *pGen* method may be modified to produce a faster method for generating primes with extra constraints, by incorporating additional tests between appropriate steps. One may need to be careful to ensure that a modified generation method is compatible with a desired standard verification method.

A.1.2 Algorithm for verifying pseudo-random DL domain parameters *p* and *r* (prime case)

This subclause describes a method to verify that prime DL domain parameters p and r were generated by a pseudo-random function of specific *SEED* and *counter* values. This is compatible with the generation method described in A.1.1 (see NOTES).

Input:

— An integer p'

— An integer r'

— An octet string *SEED* of *seedLen* octets

— An integer *counter*

— An indication of whether $k/2$ shall be prime, where $p' = kr'+1$

— An indication of whether $(r'-1)/2$ shall be prime

— A method *pVer*(p, r, *SEED*, *counter*) that verifies whether a pseudo-randomly generated DL field prime p and subgroup order prime r were generated by the values *SEED* and *counter*. (See NOTE 2.)

Output: The message "verified" or "failed."

a) Use *pVer*(*p*, *r*) to verify whether *p* and *r* are generated by *SEED* and *counter*, and if the result is not "verified," output "failed" and stop.

b) If it is indicated that $(r-1)/2$ shall be prime, perform a robust primality test (see NOTE 4) on $(r-1)/2$, and if it is not prime, output "failed" and stop.

c) If it is indicated that $k/2$ shall be prime, perform a robust primality test on $(p-1)/2r$, and if it is not prime, output "failed" and stop.

d) Output "verified."

NOTE 1—See A.1.1 for the corresponding domain parameter generation algorithm.

NOTE 2—Standard descriptions of *pVer* methods are in ANSI X9.30.1:1997 [B1], ANSI X9.42:2003 [B2] 7.2, IETF RFC 2631 [B22] 2.2.2, and FIPS Draft 186-3 [B15], A.1.1.1 and A.1.1.3.

NOTE 3—Algorithms to verify pseudo-random elliptic curves are described in IEEE Std 1363-2000 A.12.5 (prime case) and A.12.7 (binary case), FIPS PUB 186-2 [B14], and ANSI X9.62:1998 [B3].

NOTE 4—Robust primality testing methods are described in IEEE Std 1363-2000, A.15.3, and FIPS PUB 186-2 [B14].

A.1.3 Algorithm for constructing a field element of multiplicative order *q*–1

Input: DL domain parameters, including q, m, and p, where $q = p$, $q = 2^m$, or $q = p^m$, and p is prime

Output: A field element e of multiplicative order $q-1$

a) Let $e = 1$ (the field multiplicative identity element)

b) Let $e = e + 1$ (using field addition)

c) Verify whether e has multiplicative order $q-1$ (see A.1.4).

d) If the result of A.1.4 is not "true," go to step b).

e) Output e and stop.

NOTE—See Algorithm 4.80 of Menezes, van Oorschot, Vanstone [B45] for how to find a generator of a cyclic group.

A.1.4 Algorithm for verifying whether a field element has multiplicative order *q*–1

This algorithm verifies whether a field element e has multiplicative order $q-1$. This algorithm deals only with domain parameters for which the factorization of the group order is known or can be easily determined through primality testing (see NOTE).

Input: DL domain parameters, including q, m, and p, where $q = p$, $q = 2^m$, or $q = p^m$, and p is prime; a field element e

Output: "True" if the field element e is determined to be of order $q-1$, and either "false" or "unknown" otherwise (see NOTE).

a) Method 1, for when q is prime ($GF(p)$):

   1) Set integers t and r such that $p-1 = 2^t \times r$, where r is odd, and t is positive.

   2) If r equals 1, skip this step. Otherwise, test r for primality. If the result is "composite," output "unknown" and stop.

3) If $\exp(e, (p-1)/r)$ is the identity element, output "false" and stop.

4) If $\exp(e, (p-1)/2)$ is the identity element, output "false" and stop.

5) Output "true."

b) Method 2, for when $q = 2^m$ ($GF(2^m)$):

1) If e is the identity element, output "false" and stop.

2) Test 2^m-1 for primality. If the result is "composite," output "unknown" and stop.

3) Otherwise, output "true." (The group order is a Mersenne prime, making all non-identity elements primitive. Examples: m = 521, 607, 1279, 2203, 2281.)

c) Method 3, for when $q = p^m$, $p > 2$, $p-1 = 2^t$ ($GF(p^m)$, p is a Fermat prime):

1) Test $(q-1)/(p-1)$ for primality. If the result is "composite," output "unknown" and stop.

2) If $\exp(e, (p-1))$ is the identity element, output "false" and stop.

3) If $\exp(e, (q-1)/2)$ is the identity element, output "false" and stop.

4) Output "true."

d) Method 4, for when $q = p^m$, $p > 2$ ($GF(p^m)$, all other p):

1) Set integers t and r such that $p-1 = 2^t \times r$, where r is odd, and t is positive.

2) Test r for primality. If the result is "composite," output "unknown" and stop.

3) Test $(q-1)/(p-1)$ for primality. If the result is "composite," output "unknown" and stop.

4) If $\exp(e, (p-1))$ is the identity element, output "false" and stop.

5) If $\exp(e, (q-1)/2)$ is the identity element, output "false" and stop.

6) If $\exp(e, (q-1)/r)$ is the identity element, output "false" and stop.

7) Output "true."

e) Method 5, for when the prime factorization of $q-1 = {p_1}^{i_1} {p_2}^{i_2} \ldots {p_n}^{i_n}$ is known:

1) For j from 1 to n do the following:

i) If $\exp(e, (q-1) / p_j)$ is the identity element, output "false" and stop.

2) Output "true."

NOTE—These methods output "true" when e is determined to be of order $q-1$, "false" when e is determined to not be of order $q-1$, and "unknown" when the method does not determine whether e is of order $q-1$. For cases where a method outputs "unknown," there may be other methods that can establish a definitive "true" / "false" result for e.

A.1.5 Algorithm for verifying that a group element is a valid public key

Input: Valid DL or EC domain parameters (including group order r and cofactor k); an element w of the parent group

Output: "True" if w is a valid public key, and "false" otherwise.

a) For DL settings:

1) Use the algorithm for verifying that a DL public key is valid in IEEE Std 1363-2000, A.16.6, and IEEE Std 1363a-2004, A.17.5, for odd characteristic extension fields.

b) For EC settings:

1) Use the algorithm verifying that an EC public key is valid in IEEE Std 1363-2000, A.16.10, and IEEE Std 1363a-2004, A.17.5, for odd characteristic extension fields.

A.1.6 Algorithm for verifying the minimal order of a group element

The following algorithm verifies that the order of a group element meets or exceeds a specified integer value b. This method assumes that the full factorization of cofactor k is known.

Input: Valid DL or EC domain parameters (including group order r and cofactor k); an element w of the parent group; an integer b, where $1 < b \leq r$.

Output: "True" if the order of w is determined to be greater than or equal to b, and "false" if the order of w less than b or if the exact relationship cannot be determined by this method.

a) Let integers $\{p_1, p_2, ..., p_n\}$ represent all instances of the prime factors of k that are less than b

b) Compute $t := p_1 \times p_2 \times ... \times p_n$ (See NOTE 1.)

c) If (w ^ t) = the identity element, output "false," otherwise output "true." (See NOTE 2.)

NOTE 1—When t is larger than r, this method is inefficient and should not be used. Instead, one may verify that w is a valid public key, a more stringent condition that is easier to verify.

NOTE 2—When t is small, explicit exponentiation can be avoided by other equivalent methods that test to see if w belongs to the set of all known small order elements. For example, when using $GF(p)$ with $t = 2$, w is a valid public key, and thus of order $r \geq b$, if and only if it is in the range $[2, p–2]$. Two ways to ensure that $t = 2$ in a $GF(p)$ setting are by choosing $k = 2$, or by choosing $k/2$ as a large prime.

A.1.7 Algorithm for validating an element of the parent group

Input: Valid DL or EC domain parameters (including cofactor k); a value w, purported to be an element of the parent group

Output: "True" if w represents an element of the parent group, otherwise "false."

a) For DL settings:

1) Verify whether w is in the parent group (in $GF(q)^*$) using the method in IEEE Std 1363-2000, A.16.6, and IEEE Std 1363a-2004, A.17.5, for odd characteristic extension fields.

2) If the result of step 1) is "false," output "false," otherwise, output "true."

b) For EC settings:

1) If w is the identity element, output "true" and stop.

2) Verify whether w is a non-identity element in the parent group (on the curve E) using the method in IEEE Std 1363-2000, A.16.10, and IEEE Std 1363a-2004, A.17.5, for odd characteristic extension fields.

3) If the result of step 2) is "false," output "false" and stop, otherwise, output "true."

Annex B

(normative)

Conformance

Implementers should use consistent language for claiming conformance with parts of this standard. The language should clearly and completely define any constraints on the supported variations of each implemented method, including sets of acceptable choices or ranges of values for each of the following:

— Primitive inputs

— Scheme options

— Random private keys obtained within a scheme

— Values received within a scheme from other parties

General guidelines and detailed examples of conformance claims for several methods are provided in IEEE Std 1363-2000, Annex B. Additional specific recommendations and examples of conformance claims for password-based techniques are provided here, in subclauses that are numbered as an extension of the prior standard.

B.1 Conformance requirements

B.1.1 Conformance requirements for password-based techniques

Table B.1—Required subclauses for conformance with primitives

Primitive	Subclause
DLREDP-1	4.4, 8.2, 8.2.16
DLPVDGP-{SRP3,SRP6}	4.4, 8.2, 8.2.14
DLPEPKGP-{SRP3,SRP6}-SERVER	4.4, 8.2, 8.2.7
DLPKGP-SRP-CLIENT	4.4, 8.2, 8.2.10
DLSVDP-{SRP3,SRP6}-CLIENT	4.4, 8.2, 8.2.24
DLSVDP-{SRP3,SRP6}-SERVER	4.4, 8.2, 8.2.25
ECREDP-1	4.4, 8.2, 8.2.17
{DL,EC}REDP-2	4.4, 8.2, 8.2.18
{DL,EC}PVDGP-AMP	4.4, 8.2, 8.2.11
{DL,EC}PVDGP-BSPEKE2	4.4, 8.2, 8.2.12
{DL,EC}PVDGP-PAKZ	4.4, 8.2, 8.2.13
{DL,EC}PVDGP-SRP5	4.4, 8.2, 8.2.15
{DL,EC}PEPKGP-AMP-SERVER	4.4, 8.2, 8.2.4
{DL,EC}PEPKGP-PAK	4.4, 8.2, 8.2.5
{DL,EC}PEPKGP-SPEKE	4.4, 8.2, 8.2.6
ECPEPKGP-SRP5-SERVER	4.4, 8.2, 8.2.8
{DL,EC}PKGP-DH	4.4, 8.2, 8.2.9
{DL,EC}KRBP-1	4.4, 8.2, 8.2.1
{DL,EC}KRPP-1	4.4, 8.2, 8.2.2
{DL,EC}KRUP-1	4.4, 8.2, 8.2.3
{DL,EC}SVDP-AMP-CLIENT	4.4, 8.2, 8.2.19
{DL,EC}SVDP-AMP-SERVER	4.4, 8.2, 8.2.20

Table B.1—Required subclauses for conformance with primitives *(continued)*

Primitive	Subclause
{DL,EC}SVDP-PAK1-CLIENT	4.4, 8.2, 8.2.21
{DL,EC}SVDP-PAK2	4.4, 8.2, 8.2.22
{DL,EC}SVDP-SPEKE	4.4, 8.2, 8.2.23
ECSVDP-SRP5-CLIENT	4.4, 8.2, 8.2.26
ECSVDP-SRP5-SERVER	4.4, 8.2, 8.2.27
{DL,EC}SVDP-WSPEKE-{CLIENT,SERVER}	4.4, 8.2, 8.2.28

Table B.2—Required subclauses for conformance with schemes

Scheme	Operation	Subclause
{DL,EC}BPKAS-PAK	Key agreement	4.5, 9.1, 9.2.1, 9.2.2
	Key confirmation	4.5, 9.1, 9.2.1, 9.2.3
{DL,EC}BPKAS-PPK	Key agreement	4.5, 9.1, 9.3.1, 9.3.2
	Key confirmation	4.5, 9.1, 9.3.1, 9.3.3
{DL,EC}BPKAS-SPEKE	Key agreement	4.5, 9.1, 9.4.1, 9.4.2
	Key confirmation	4.5, 9.1, 9.4.1, 9.4.3
{DL,EC}APKAS-AMP	Key agreement	4.5, 9.1, 9.5.1, 9.5.2
	Key confirmation	4.5, 9.1, 9.5.1, 9.5.3
{DL,EC}APKAS-BSPEKE2	Key agreement	4.5, 9.1, 9.6.1, 9.6.2
	Key confirmation	4.5, 9.1, 9.6.1, 9.6.3
{DL,EC}APKAS-PAKZ	Key agreement	4.5, 9.1, 9.7.1, 9.7.2
	Key confirmation	4.5, 9.1, 9.7.1, 9.7.3
DLAPKAS-{SRP3,SRP6}	Key agreement	4.5, 9.1, 9.8.1, 9.8.2
	Key confirmation	4.5, 9.1, 9.8.1, 9.8.3
ECAPKAS-SRP5	Key agreement	4.5, 9.1, 9.9.1, 9.9.2
	Key confirmation	4.5, 9.1, 9.9.1, 9.9.3
{DL,EC}APKAS-WSPEKE	Key agreement	4.5, 9.1, 9.10.1, 9.10.2
	Key confirmation	4.5, 9.1, 9.10.1, 9.10.3
{DL,EC}PKRS-1	Key agreement	4.5, 10.1, 10.2.1, 10.2.2

B.1.1.1 Conformance region recommendations for primitives

For any primitives defined in this document, a conformance region should include:

— *For input domain parameters:* At least one valid set of domain parameters.

— *For input private keys:* All valid private keys for each set of domain parameters.

— *For input public keys and group elements:* All values conforming to the stated assumptions for each set of domain parameters.

— *For other input options:* May be preset by the implementation or given as an input flag.

B.1.1.2 Conformance region recommendations for schemes

For any scheme options defined in this document, a conformance region should include:

— *For domain parameters:* At least one valid set of domain parameters.

— *For password-based octet strings:* A suitable range of acceptable values.

— *For password verification data:* All values defined by the appropriate PVDGP.

— *For function choices:* At least one of the specified functions.

— *For security parameters:* At least one value.

— *For parameter octet strings:* A range of values.

— *For static private keys:* At least one private key for each set of domain parameters.

Where any scheme obtains a random ephemeral private key, a conformance region should include the full range of private keys for each set of domain parameters (see D.2.1.16).

Where any scheme receives a value from another party, a conformance region *should not* include limitations on what the received values can be (see NOTE).

For each function choice of a scheme, the conformance region should further (recursively) include values for any parameterized choices of the function. For example, MGF1 needs a *Hash*, and REDP-2 needs a *Hash* and g_a, g_b values.

NOTE—Schemes defined in this document are designed to operate on arbitrary received values that are potentially created by an adversary. When created by a legitimate party, these values will represent valid objects, including (password-entangled) public keys or (permuted) blinded passwords, but they may not represent valid objects when created by an adversary. The schemes prevent potential attacks in various ways: some schemes check for and abort after detecting invalid values; others operate consistently on both valid and invalid values. In any case, a conformance region should not impose constraints on these received values to ensure that any conformant implementation operates on *all* values in the manner specified by the scheme.

B.2 Examples

This subclause gives some examples of claims of conformance with the primitives and scheme operations in the standard.

B.2.1 DLPVDGP-SRP6

A module claiming conformance to DLPVDGP-SRP6 under one set of reasonable conditions might document its conformance as follows:

"Conforms with IEEE Std 1363.2 DLPVDGP-SRP6 / MGF1(SHA-1, 256) over the region where q has 2048 bits, r has 256 bits, and password value π has at most 128 octets."

An indented form that highlights the structure of this statement is as follows:

"Conforms with IEEE Std 1363.2 DLPVDGP-SRP6 /
MGF1(SHA-1, 256)
over the region where
q has 2048 bits,
r has 256 bits, and
password value π has at most 128 octets."

B.2.2 ECAPKAS-PAKZ-SERVER

A module claiming conformance to the ECAPKAS-PAKZ-SERVER key agreement and key confirmation operations, under one set of reasonable conditions, might document its conformance as follows:

"Conforms with IEEE Std 1363.2 ECAPKAS-PAKZ-SERVER / ECPVDGP-PAKZ / ECREDP-2(MGF1 (SHA-1, 160), any valid g_a and g_b) / ECSSA(ECSP-DSA(...), EMSA1) / MGF1(SHA-1, 160) / SHA-1 /

KDF1(SHA-1) / KCF1(SHA-1) key agreement and key confirmation operations over the region where ..."

An indented form that highlights the structure of this statement is as follows:

"Conforms with IEEE Std 1363.2 ECAPKAS-PAKZ-SERVER /
ECPVDGP-PAKZ /
ECREDP-2(
MGF1(SHA-1, 160) /
any valid g_a and g_b
) /
ECSSA(
ECSP-DSA(...),
EMSA1
) /
MGF1(SHA-1, 160) /

SHA-1 /
KDF1(SHA-1) /
KCF1(SHA-1)
key agreement and key confirmation operations over the region where ..."

Annex C

(informative)

Rationale

This annex is presented in the form of questions and answers.

Much of the rationale discussed in IEEE Std 1363-2000, Annex C (*), is applicable to the methods in this document. The additional subclauses presented in this document are numbered as an extension of this prior standard.

Relevant portions of IEEE Std 1363-2000, Annex C (*), are listed here:

C.1 General

C.1.1 Why are there three families of cryptographic techniques?

C.1.2 Why are primitives and schemes separated?

C.1.4 Why are constraints on key sizes not specified?

C.1.6 Why are key derivation functions needed?

C.1.7 Why are data formats for input/output, keys, and domain parameters not normative?

C.2 Keys and domain parameters

C.2.1 Why have three types of fields for the DL and EC families? *

C.2.2 Why allow multiple representations for $GF(2^m)$?

C.3 Schemes

C.3.2 For the DL and EC families, why have the "compatibility" option for the DHC and MQVC primitives?

C.4 Additional methods *

C.4.1 Why were the KDF1 and KDF2 key derivation functions selected? *

C.4.2 Why was the MGF1 mask generation function selected? *

C.4.3 Why have SHA-1, RIPEMD-160, SHA-256, SHA-384, and SHA-512? *

C.4.5 Why was the MAC1 message authentication code selected? *

NOTE—The asterisk (*) denotes areas amended in IEEE Std 1363a-2004.

C.1 General

C.1.1 Password-based techniques

C.1.1.1 Why are there specific methods for passwords?

Passwords are in widespread use as authenticators of human presence, and thus constitute an important factor in authentication. However, as they are typically low-grade secrets, they often cannot be responsibly used to directly derive secret keys or private keys for use in ordinary cryptographic techniques. It is thus

natural to search for methods that optimize the use and protection of passwords. This search has resulted in cryptographic techniques that can, in a network setting, derive authenticated high-grade keys from this particularly sensitive class of authentication data.

C.1.1.2 Why are password methods that use public-key cryptography needed?

Many widely used password-based network authentication systems use a password as a symmetric key or as an authentication key in a keyed message authentication code. Examples in the latter category include systems that create an authentication response code based on a one-way hash function of a password and a random challenge. As a class, such systems are all vulnerable to unconstrained guessing attacks given access to the corresponding cipher text or message authentication code. This specification is focused on systems that inherently prevent unconstrained guessing attack on password-derived data. Research indicates that public-key cryptography is somehow fundamental in providing this extra protection.

C.1.1.3 Why are password-only methods needed?

In limiting the focus to methods that use only a password, this specification addresses an important need that has not been addressed by other high-level cryptography standards. Many systems require a previously secured channel to transmit passwords, hashed passwords, or other sensitive data to appropriate parties for verification. In such systems, channel security is typically based on some other factor, such as physical security, high-grade secrets, and/or a method for authenticating the other party. Other standards efforts (e.g., IEEE Std 1363-2000) cover such methods, but they have generally not covered password-only methods for securing a channel. Password-only methods can be used as an alternative when other factors are not available or reliable, and password-only methods can be used in combination with other methods to provide an added basis for security. By limiting the scope of this document to password-only methods, we can better focus on the presentation of a few core methods that may be used for developing other standards for application-specific or hybrid protocols.

C.1.1.4 Why are there multiple password-based techniques within the same class and family?

The known methods for password-authenticated key negotiation have relative advantages and disadvantages, with differences in functionality, speed, efficiency, message size, number of messages, implementation size and complexity, availability and scope of cryptanalytic research, patent coverage, and use in other standards and products. C.2 discusses issues with specific methods. The rationale for the presence of methods in this document includes the following:

— Some methods provide unilateral commitment to a password-based value, while other methods provide bilateral commitment.

— Some methods are associated with proofs of security.

— Some methods provide enhanced performance or have functional optimizations for specific applications.

— Multiple comparable methods have been used in products.

The goal of this standard is not to restrict users to a single choice, but rather to provide a framework that allows selection of methods appropriate for particular applications. Since implementers of this standard need not implement it in its entirety, they have the option to choose the techniques that best suit their needs.

C.1.1.5 How were the decisions made regarding the selection of individual password-based methods?

The main purpose for the current version of the standard is to specify password-based techniques in a common way. The selection of schemes in the current version of the standard is based on the desire to meet a range of requirements as stated previously, in consideration of the relative advantages and limitations of proposed methods. Some further reasons for the selection of individual schemes can be found in C.2.

It is anticipated that the Working Group will provide a forum for discussing additional techniques to be included in an addendum document to supplement the techniques already covered in this standard, and it is intended that the two documents will be merged during future revisions.

C.1.1.6 Is provable security considered as a selection criteria?

Security analysis is used in the rationale for method selection. But there is no requirement that a selected method be accompanied by a proof that the method has equivalent security to any other particular method or mathematical problem. IEEE Std 1363A-2004, D.1, further discusses the benefits and limitations of security analysis.

C.1.1.7 Is the extra benefit of the augmented model always needed?

Whether the extra benefit of an augmented (APKAS) model over the balanced model (BPKAS) model is significant depends on the application.

In some applications for retrieving password-encrypted credentials, the extra benefit of the augmented model may not be realized Perlman and Kaufman [B48]. Recall that the extra benefit of an augmented model is that an adversary who steals the password verification data needs to also guess the password to successfully masquerade as the user. Now, consider an application that uses an augmented method only to create a secure channel to retrieve password-encrypted credentials that are stored along with the password verification data. Clearly, a thief that steals the verification data also has access to the encrypted credentials, and thus has no need to try to masquerade as the user to get those credentials. In this case, the APKAS method provides no useful benefit over a BPKAS method.

The case described above presumes that the host stores encrypted credentials and password verification data using the same mechanism and level of care. But in scenarios where credential storage is more secure than verification data storage, then there may indeed be a realizable added benefit of the augmented model.

In other applications there may be functional reasons for preferring a balanced method, such as where two parties share a common secret, or where neither party can store precomputed verification data. While it is easy to substitute an augmented method for a balanced method by computing the verification data on-demand, the application may not want to incur the cost of this added computation. Whether these selection factors are important depends on the key agreement model and the relative implementation and deployment costs of augmented vs. balanced methods for the desired applications.

C.1.1.8 Why are random element derivation methods needed?

Multiple schemes need to derive group elements from passwords. But for security reasons some of these schemes require that password-derived elements meet a special constraint: The elements associated with different passwords shall not be related by any exponential relationship that might be known by an adversary. An appropriate password-to-random-element encoding method provides a systematic way of meeting this constraint and thwarting such attacks.

In the recommended approach, the implementer specifies a method to transform passwords into integers, and then uses an approved REDP to transform integers into pseudo-random group elements. The properties of the integer-to-element derivation methods assure that for any reasonable password-to-integer function that an implementer might construct, the derived group elements will have the desired property.

It is highly recommended to use these methods when implementing the schemes specified in the standard. Further discussion of security properties of message encoding methods is in Annex D of IEEE Std 1363-2000.

C.2 Schemes and primitives

C.2.1 Why have two random element derivation primitives (REDP-1 and REDP-2)?

The two REDPs REDP-1 and REDP-2 allow for different performance characteristics and trade-offs in certain methods, depending on input and domain parameters and the acceptability of security assumptions. REDP-2 may be faster than REDP-1 in cases where the cofactor k is larger than the integer resulting from the *hash* function in REDP-1, which depends on a security parameter. In other cases, particularly when using a small cofactor k, REDP-1 is faster than REDP-2.

For the SPEKE methods, it is generally desirable that password-entangled public keys provide zero knowledge of the password to an adversary. Whether this goal is achieved may depend on different assumptions when using REDP-1 and REDP-2, where each method is parameterized for high performance. REDP-1 permits use of *short exponents* in DL groups, for example, where security parameters $c = 160$ and $d = 1024$, $k = 2$, $\lceil \log_2 r \rceil = d$, and private key $u \in [1, 2^c - 1]$. This practice relies on the *short exponent assumption* (see D.2.1.4) that $g^{\wedge}u$ does not reveal information about g, even though u is from a smaller range than $[1, r-1]$. In contrast, REDP-2 permits use of DL groups where $k >> r$, which allows private keys to be chosen uniformly from $[1, r-1]$ while still preserving reasonable efficiency. Here the presumption of zero knowledge of the password being available from the password entangled public keys (alone) relies solely on the assumption of uniform distribution of the random private keys in $[1, r-1]$, and does not rely on the presumed difficulty of any mathematical problem.

Further discussion of techniques like REDP-2 that use two independent generators in password-authenticated methods can be found in Jablon [B29], MacKenzie [B41], and IETF RFC 2631 [B22] .

C.2.2 Why have both key agreement and key retrieval classes of password-authenticated schemes (PKAS and KRS)?

Both PKAS and KRS methods may be used in many applications. PKAS are preferable for some applications because they derive a fresh mutually generated key between two parties, in contrast with key retrieval schemes that (generally) retrieve static or unilaterally chosen keys. However, some key retrieval schemes are especially suitable for applications that split and distribute multiple shares of password verification data to reduce vulnerability of attack when less than a specified number of parties have been compromised.

C.2.3 Why have two classes of key agreement schemes (balanced and augmented)?

Including both BPKASs and APKASs allows one to choose between the (typically greater) efficiency of a balanced method vs. the increased functionality of an augmented method.

The balanced schemes may be appropriate for two parties that share a common password, whereas the augmented schemes, which generally require more computation, may be appropriate for cases where one

party securely stores persistent password verification data for the other. (See C.1.1.7 for more discussion of use scenarios.)

C.2.4 Why have multiple balanced PKAS?

C.2.4.1 Why have PAK?

PAK has been deployed in product. There is a security reduction argument for PAK in the random oracle model based on the computational Diffie-Hellman assumption (see D.2.2.6.2). PAK is a unilateral commitment scheme, where the Server does not provide a commitment to the password during key agreement. PAK is computationally more efficient than PPK (see D.2.1.9).

C.2.4.2 Why have PPK?

PPK has been deployed in product. There is a security reduction argument for PPK in the random oracle model based on the computational Diffie-Hellman assumption (see D.2.2.6.2). PPK is a bilateral commitment scheme, where both parties provide a commitment to the password during key agreement, and thus does not require key confirmation or special ordering of messages as does PAK.

C.2.4.3 Why have SPEKE?

SPEKE has been deployed in products. A security reduction argument for SPEKE based on a variant of the decision Diffie-Hellman assumption due to MacKenzie is discussed in D.2.2.2.4.

C.2.5 Why have multiple augmented PKAS?

C.2.5.1 Why have AMP?

AMP is most efficient.

C.2.5.2 Why have BSPEKE2?

BSPEKE2 has been used in products and has been published for a longer time than WSPEKE.

C.2.5.3 Why have PAKZ?

PAKZ is associated with a security reduction argument to the Computational Diffie-Hellman (CDH) assumption in the random oracle model.

C.2.5.4 Why have DLSRP3?

SRP3 is being standardized elsewhere and has been deployed. SRP3 is based on IETF RFC 2945 [B24], and has been used in some early SRP-based protocols like SRP Telnet.

C.2.5.5 Why have DLSRP6?

SRP6 is being standardized elsewhere and has been deployed. SRP6 supersedes SRP3 and is being specified (as "SRP-6") in current proposals such as the SRP-TLS cipher suites.

C.2.5.6 Why have ECAPKAS-SRP5?

ECAPKAS-SRP5 is the EC variant of DLAPKAS-SRP6.

C.2.5.7 Why have APKAS-WSPEKE?

WSPEKE is used in product. WSPEKE is closely related to the SPEKE method, is defined for both the EC and DL setting, and uses less computation for the Client than SRP{3,5,6}. WSPEKE can use less computation than BSPEKE2.

C.2.5.8 How are the various SRP methods related?

APKAS-SRP3 is an augmented scheme defined only for the DL setting, as described in Wu [B57], and is compatible with RFC 2945 [B24]. APKAS-SRP3 has two notable limitations:

— A two-for-one guessing limitation, that permits an attacker to guess two passwords in one run of the scheme

— A message ordering limitation, that requires the Server to receive the Client's public key w_C before sending its password-entangled public key w_S.

This document also defines three methods derived from SRP3 that do not have the two-for-one guessing and message ordering limitations:

— SRP5, a refinement of SRP3 based on a generalization of Wang [B55] and defined for only the EC setting.

— SRP6, a refinement of SRP3 defined for only the DL setting.

— WSPEKE, a hybrid of both SRP3 and BSPEKE Jablon [B32].

C.2.5.9 How are BSPEKE2 and WSPEKE related?

WSPEKE is different from BSPEKE2 in the following ways:

— WSPEKE provides for increased computational efficiency over BSPEKE2.

— WSPEKE Client and Server use different SVDP functions than BSPEKE2.

— BSPEKE2 is more compatible with KAS-DH2 than WSPEKE.

Annex D

(informative)

Security considerations

Many of the security considerations described in IEEE Std 1363-2000, Annex D (*), are applicable to the methods in this document. Subclauses in this clause are numbered as an extension of Annex D of this prior standard.

Relevant portions of IEEE Std 1363-2000, Annex D (*), are listed here:

D.1 Introduction

D.2 General principles

D.3 Key management considerations

- D.3.1 Generation of domain parameters and keys
- D.3.2 Authentication of ownership *
- D.3.3 Validation of domain parameters and keys
- D.3.4 Cryptoperiod and protection lifetime of domain parameters and keys
- D.3.5 Usage restrictions
- D.3.6 Storage and distribution methods *

D.4 Family-specific considerations

- D.4.1 DL family
  - D.4.1.1 Security parameters *
  - D.4.1.2 Generation method *
  - D.4.1.3 Other considerations *
  - D.4.1.4 Notes *
- D.4.2 EC Family
  - D.4.2.1 Security parameters *
  - D.4.2.2 Generation method *
  - D.4.2.3 Other considerations *
  - D.4.2.4 Notes *

D.5 Scheme-specific considerations

- D.5.1 Key agreement schemes
  - D.5.1.1 Primitives *
  - D.5.1.2 Additional methods *
    - D.5.1.2.1 Key derivation functions *
    - D.5.1.2.2 Hash functions *
  - D.5.1.3 Key confirmation
  - D.5.1.4 Key derivation parameters

NOTE—The asterisk (*) denotes areas amended in IEEE Std 1363a-2004.

D.1 Family-specific considerations

D.1.1 DL family

D.1.1.1 Additional notes on pseudo-random prime generation

This subclause provides further guidance on pseudo-random generation and validation of primes for $GF(p)$ settings, supplementing in IEEE Std 1363-2000, D.4.1.4 NOTE 3.

Standard descriptions of prime generation methods (and corresponding verification methods) are in:

— NIST FIPS PUB 186-2 [B14], Appendix 2.2 (and FIPS Draft 186-3 [B15], A.1.1.1)

— ANSI X9.30.1:1997 [B1]

— ANSI X9.42:2003 [B2], 7.1 (and 7.2)

— IETF RFC 2631 [B22], 2.2.1 (and 2.2.2)

— FIPS Draft 186-3 [B15], A.1.1.2 (and A.1.1.3)

Depending on the scheme, additional constraints for the domain parameters may be desirable, as discussed in D.2.1.5, D.2.2.1.1, and D.2.2.3.1. In general, any of the standard probabilistic pseudo-random prime generation methods can be iterated, with extra tests, until suitable primes are found with the desired extra constraints, as described in A.1.1.

D.2 Scheme-specific considerations

D.2.1 General considerations for password authenticated schemes (PKAS and PKRS)

PKAS are used in pairs, where two parties execute a pair of complementary schemes in tandem. The key agreement operation of each party in a PKAS generally consists of public-key generation and secret value derivation. Each party generates a public key using a PKGP and makes the public key available to the other party. At least one of these keys is a *password-entangled* public key that is generated from the password and a private key using a PEPKGP. Each party obtains the other's (password-entangled) public key and uses it as input to an SVDP to derive the shared secret value.

In IEEE Std 1363-2000 key agreement schemes, a party typically needs to be provided with other assurance of the ownership of a received public key in order to authenticate the identity of the other party that shares

the derived secret key. However, in a PKAS, the scheme itself provides assurance that the creator of a password-entangled public key knows the corresponding password, and thus, the password authenticates the parties that derived the shared keys.

D.2.1.1 Comparison of APKAS with BPKAS

An APKAS is similar to a BPKAS, except for two distinctive features of an APKAS:

— The APKAS Server knows verification data that is computed as a one-way function of π, instead of knowing π directly.

— A thief that knows the APKAS verification data cannot masquerade as the Client to the Server without first determining π from the data (e.g., typically using a brute-force password attack—"cracking" the password).

The APKAS model thus provides an added benefit, but typically requires that the Server store the pre-computed verification data.

Some environments may not be able to take advantage of the extra benefits of an APKAS, such as a user-to-user application, or a client/server application with stored password verification data in a preexisting format that is incompatible with the augmented scheme. Furthermore, the threat model in some environments may not provide an opportunity to gain the added benefit of an APKAS, as compared to a similar (or underlying) BPKAS. For further discussion of augmented vs. balanced methods, see C.1.1.7.

D.2.1.2 Key confirmation in password authenticated key agreement schemes

Key confirmation provides an explicit authentication of the agreed key from one party to the other (see D.2.1.7).

Key confirmation also provides an explicit zero-knowledge proof of knowledge of a password or password verification data from one party to the other. In a BPKAS, each party proves knowledge of the same password data to the other. In an APKAS, the Client proves knowledge of the password whereas the Server proves knowledge of corresponding password verification data.

All PKAS in this document define key confirmation operations. These operations are optional in cases where the KCF may be omitted, without impacting security. These operations may be implemented using the steps defined in the scheme, or they may just as well be provided by an invoking application protocol. However, in cases where a specific ordering for key confirmation operations is needed to preserve the security of a scheme, the scheme is defined to perform key confirmation operations in the appropriate order (see D.2.1.9.1).

Other issues with key confirmation that may be applicable to PKAS, such as unknown key share attacks, are discussed in IEEE Std 1363-2000, 9.1 NOTE 1, and IEEE Std 1363-2000, D.5.1.3. In cases where a user shares the same password with multiple parties, it may be is especially important to deal with unknown key share and other identity confusion attacks, as discussed in D.2.1.10.

D.2.1.3 Domain parameter and key validation in PKA and PKR schemes

IEEE Std 1363-2000, D.3.3 and D.5.1.6, discuss issues with domain parameter and key validation in general, and specifically for key agreement schemes, respectively. These issues are generally applicable to password-authenticated key agreement and key retrieval schemes (PKAS and PKRS). The following subclauses further discuss domain parameter selection, key validation, and related performance issues that apply more specifically to PKAS and PKRS, and D.2.2 discusses such issues for particular schemes.

D.2.1.4 Using *GF*(*p*) with "safe prime" *p* and short exponent assumption

Performance in a PKAS or PKRS may be improved for some schemes in some DL settings (see NOTE 1) by using a private key that is significantly shorter than r, and relying on the *short exponent assumption*, which assumes that short exponents may be used reliably in an appropriately constrained method. This assumption is discussed in Menezes, van Oorschot, Vanstone [B45] and Gennaro [B17] with regard to using *safe prime* DL domain parameters.

Specifically, it has been noted that there are no significant known attacks when cofactor k is 2 and short private keys are chosen from the range $[1, 2^c-1]$, where c is the length in bits of the recommended minimum subgroup order r that corresponds to field order q. For example, a value of $c = 160$ might be used when $q \cong 2^{1024}$. (See NOTE 2.)

However, theoretical concerns linger. Random self-reduction (RSR) applies to the both the Diffie-Hellman problem and the discrete log problem, to relate the hardness of special cases to the general case, and Brown and Gallant [B9] argue that RSR does not apply when the special case is a negligible proportion, as when using short exponents.

NOTE 1—Short private keys are not relevant when $\log_2 r \cong \log_2 q$, as is typical in EC settings.

NOTE 2—Attacks that one should consider in trying to establish a minimum required subgroup order and a minimum required field size, and assumptions to consider in determining a precise correspondence between these values, are discussed in IEEE Std 1363a-2004, D.4.1.4 and D.4.2.4.

D.2.1.5 Small subgroup confinement attack

Some schemes include steps for verifying that the order of a group element e is not "unacceptably small," in order to defend against small subgroup attacks (see Jablon [B31] and IEEE Std 1363-2000, D.5.1.6). In a small subgroup confinement attack, the adversary selects e, modifies e, or causes a party to compute e so that it generates an unacceptably small group, in an attempt to confine a subsequently-derived secret value z to a set that is enumerable by the adversary.

Note that the trivial small subgroup $\{1\}$ is present in all settings, and that in any DL *GF*(p) setting, the integer values nq, $(nq+1)$, and $(nq-1)$ for any integer n are all generators of a group with either 1 or 2 elements.

The defense against such attack is to treat any value of e that is not of sufficiently large order as an error. Although the explicit meaning of "sufficiently large" is left as an implementation choice, and the nature of error handling is defined by the scheme, some general guidance is provided here.

One test for an acceptable value e is to ensure that e is a valid public key—an element of order r in the subgroup generated by g. A party may test e using the algorithm in A.1.5, and abort if e' is not a valid public key.

However, it may be sufficient (and more efficient) to merely ensure that e is a generator of any group of b or more elements, where the implementation determines security parameter b such that any e of order less than b is rejected. This ensures that an adversary cannot confine the legitimate party's derived secret z to a set with less than b elements, thus ensuring that the adversary using a random password value π has no greater than a $1/b$ probability of negotiating shared key z in each run (see NOTE 1). When a security factor b has been determined, and when all prime factors of cofactor k are known, one may first verify that e is in the parent group using A.1.7, and then also verify that the order of w is greater than or equal to b using A.1.6, to determine that w is acceptable.

One way to simplify addressing small subgroup confinement in a DL *GF*(p) setting is to choose domain parameters such that there are no non-trivial factors of $(q-1)/2$ smaller than r. This guarantees that the only

two unacceptably small order elements of the parent group are 1 and $q-1$. In this case, any value in the range $[2, q-2]$ is a sufficiently large order element of the parent group. See A.1.1 for an algorithm to construct DL domain parameters where cofactor k is 2, or $k/2$ is a large prime, each of which insures that all factors of $(q-1)/2$ are sufficiently large.

NOTE 1—Essentially, b indicates the desired level of resistance to online guessing attack. One can securely choose $b = r$. However, values of $b < r$ may be acceptable depending on the scheme, the expected randomness of the password, and the constraints on password guessing that may be imposed by the application. It generally does not make sense for b to be greater than r, since the order of the group establishes a lower bound on the probability of guessing the derived key by brute force. The level of resistance to attack is also limited by the domain parameters that determine the difficulty of discrete log attacks. Roughly speaking, there is no additional benefit in making b more than twice as large as a symmetric key that has a strength equivalent to the domain parameters.

NOTE 2—See D.2.2.1 and D.2.2.1.1 for discussion of key validation in the AMP and SPEKE schemes respectively.

D.2.1.6 Pohlig-Hellman decomposition attack

In many schemes, Server key establishment operations compute values using group operations on a public key (w_C) received from the Client and a Server's private key (s). In schemes where the direct result of a group operation (e.g., $w_S = w_C \hat{} s$) is revealed to the Client, one concern is the threat of a *Pohlig-Hellman decomposition attack* where an adversary poses as a Client, submits an invalid w_C, and examines the result to determine information about s. A specific attack is described in Wan and Wang [B54] that applies to a variant of AMP as described in Kwon [B34]. The concern of Pohlig-Hellman decomposition is discussed in van Oorschot and Wiener [B52], and the general threat applies to both APKAS-AMP and to PKRS-1.

To address this threat, the Server may generally

— Validate w_C and/or

— Use domain parameters that limit such information disclosure.

Such attack is completely prevented when the Server checks to ensure that w_C is a valid public key before using it. Using this approach, when w_C is not a valid public key, the Server should abort or take other appropriate action as defined by the scheme.

However, depending on the setting and application, the cost of testing for a valid public key may be undesired. To eliminate this cost, the domain parameters may be selected specifically to ensure that any invalid public keys that are elements of the parent group do not pose this threat. The decomposition attack is also prevented when using the following examples of domain parameters, where the order of the received w_C can never have too many small factors.

For one example, in a setting with cofactor $k = 1$, common in EC settings, all non-trivial small subgroups are eliminated. Other examples are DL $GF(p)$ settings with $k = 2$ (where p is a "safe prime"), or with $k/2$ prime, as described for PKRS-1 in Ford and Kaliski [B16] and Jablon [B29]. Both of these settings ensure that the only factor of the order of the parent group smaller than r is the single factor 2. In these DL cases, a Pohlig-Hellman decomposition attack yields only the lowest 1 bit of s. The size and structure of s can be adjusted to compensate: Make s one bit larger, with a value for the low bit that may be revealed without compromising security.

In each of the preceding examples, merely validating that w_C is a member of the parent group (see NOTE and A.1.7) is sufficient to prevent any significant disclosure of s; one may omit the step of verifying w_C as a valid public key without compromising security. For other choices of domain parameters, further analysis may be required by the implementer in order to omit or replace this step without compromising security.

See A.1.1 for how to create DL $GF(p)$ domain parameters with $k = 2$ or $k/2$ prime, and for EC settings, IEEE Std 1363-2000, A.12.4 (prime case) and A.12.6 (binary case), FIPS PUB 186-2 [B14], and ANSI X9.62:1998 [B3].

NOTE—All schemes in this standard ensure that w_C is in the parent group to make subsequent primitive operations well-defined.

D.2.1.7 Explicit vs. implicit key authentication in PKAS

This document distinguishes between *implicit key authentication* and *explicit key authentication*, when discussing authenticated key agreement schemes and protocols. These terms are defined and discussed in Chapter 12 of Menezes, van Oorschot, Vanstone [B45], and further discussed (using the terms *direct* and *indirect authentication*) in Diffie, van Oorschot, Wiener [B11].

Executing a complementary pair of BPKAS or APKAS provides implicit key authentication of the agreed key for the parties. Implicit key authentication assures each party that its derived key is shared with another party only if that other party has used the corresponding password-derived data.

Schemes or their invoking applications may further provide explicit key authentication, which assures each party that the other has actually used the corresponding password-derived data. This is generally accomplished by having the parties exchange and verify each other's proof of knowledge of the agreed key. Explicit key authentication may be provided by the key confirmation steps in a PKAS (see D.2.1.2) or otherwise in an invoking application protocol.

D.2.1.8 Password commitment and proof in PKA protocols

An interactive *zero knowledge proof* (ZKP) has a commitment phase and a proof phase. It is a characteristic of all PKA methods, when used to provide a mutual ZKP of common password-derived data, that each party accepts a commitment from the other before sending the other a proof. Otherwise, if a party sent a proof of knowledge of a password to an uncommitted party, the uncommitted party would have the ability to perform unconstrained guessing by verifying the proof against a list of candidate passwords. The security of a PKAS relies on the property of a proof that can be verified only by one who has previously committed to the password value.

D.2.1.9 Unilateral and bilateral commitment

In any balanced or augmented PKAS, each party commits to knowledge of the password value π before the other party reveals information about π. The key agreement operations in any pair of PKAS provide either *unilateral commitment* or *bilateral commitment* to the password. In unilateral commitment, only one party sends a password-entangled public key to the other, thereby committing itself to a specific value for π. In bilateral commitment, both parties send password-entangled public keys to each other, committing each to a specific value for π. This difference raises a special concern for the secure use of unilateral schemes, discussed in D.2.1.9.1.

D.2.1.9.1 Secure use of unilateral commitment schemes

In a unilateral commitment scheme, only one of the two parties commits to the password by sending a password-entangled public key, which is constructed using a PEPKGP. The other party typically sends an unentangled public key, which is constructed using a PKGP. To preserve the zero-knowledge proof property, the committed party first verifies a proof that the uncommitted party knows the agreed secret key before revealing information about the derived key (see NOTE). Mandatory key confirmation operations provide this function.

Each unilateral commitment scheme defines mandatory key confirmation operations, which are performed in a specified order, as well as specific steps that may be used to implement these operations. Mandatory key confirmation may alternately be performed by other equivalent steps in an invoking application protocol, where the previously uncommitted party proves itself first using the specified order of key confirmation operations.

The specific unilateral commitment schemes that require this consideration are BPKAS-PAK, APKAS-AMP, APKAS-PAKZ, DLAPKAS-SRP3, DLAPKAS-SRP6, and ECAPKAS-SRP5. (See D.2.1.21 for discussion of how unilateral commitment relates to error handling for incomplete runs.)

In this discussion we have not yet addressed the question: Exactly *who is* that other party? The simplest answer is: The other party is one that has used the corresponding password-derived data. Whether any party associates a name with the other party, or performs any additional name-to-key binding, depends on the invoking application or protocol. (See D.2.1.10 for further discussion of identity attacks.)

NOTE—The threat to improper use of a unilateral commitment scheme is this: If Alice has not yet committed to the password, and Bob sends a confirmation message to Alice before verifying any commitment from her, an adversary masquerading as Alice may perform unconstrained off-line guessing.

D.2.1.10 Identity attacks

The schemes in this document provide authentication based solely on password-derived data, and derive keys based on this data and other general purpose input parameters; these schemes do not explicitly deal with party names or other aspects of party identity.

However, application protocols may use PKAS in combination with party identifiers and may associate passwords and related keys with identified parties in a variety of ways. Such protocols may have to deal with additional security issues related to binding password-data and password-derived keys to party identifiers. Two classes of identity attacks on a key agreement protocol that produce distinct forms of identity confusion are *impersonation attacks* and *unknown key share attacks*.

An *impersonation attack* on Alice is where an adversary, say, Marielle, causes Alice to believe she shares a key with Bob, when Alice actually shares the key with Marielle. PKAS generally make such attacks impossible, unless Marielle steals the appropriate password data.

In an *unknown key share attack*, as discussed in IEEE Std 1363-2000, D.5.1.3, an adversary, we will name Mallory, causes Alice to believe she shares a key with Mallory, when Alice actually shares the key with someone else, like Bob. Another related problem is where Mallory causes Alice to believe she shares a key with Charlie, when Alice actually shares the key with Bob. This might occur when using a PKAS where Alice uses the same password with Bob and Charlie, and uses a protocol that does not securely identify the intended target. Alice intends to start a session with Bob, but Mallory redirects the messages so that Alice actually talks to Charlie, with undesirable consequences.

These identity attacks are generally precluded when parties securely communicate their intentions to each other. When using a BPKAS, party identifiers may be included, with a pre-agreed meaning, in the common password-based input parameter π. There may also be relevant security analysis to consider, such as discussed in D.2.2.2.4, that presumes the incorporation of party identifiers. The result is that only parties that share this information will agree on the key. When using any APKAS or BPKAS, party identifiers may also be included in key derivation input parameters for the scheme, with similar effect. The parties can use the derived shared secret key in a key confirmation protocol (see D.2.1.2) to provide each other with a subsequent combined proof of knowledge of, or encrypted message describing, the intended participants. Such steps can prevent any confusion about the intended participants of the scheme.

D.2.1.11 Passphrase

A passphrase can be thought of as simply a long password. In some systems, especially where a password is intended to be used as a cryptographic key, a passphrase is suggested to help assure that sufficient entropy is present. In practice, it may be difficult to guarantee there is sufficient entropy in a human-selected passphrase, so that it may be prudent to assume it is vulnerable to brute-force attack.

D.2.1.12 Brute-force attack

A brute-force attack on a cryptographic authentication system systematically searches a space of keys, passwords, or other secret data to find a value that gives the attacker the ability to pose as the authentic party. For keys, the key size and quality is typically controlled so as to make brute-force attack infeasible. The difficulty of a brute-force attack on a key typically represents an upper bound on computational work required of an adversary. However, size and/or quality constraints on passwords (see D.2.1.14) often make them unsuitable to directly derive cryptographic keys, even when they still have significant value as authenticators in a suitable protocol.

When a targeted password is likely to be confined to a small subset of the space of all possible passwords, a brute-force password attack is much easier than an *exhaustive attack* of that space. The common term *dictionary attack* has been alternately used broadly as a synonym for brute-force attack, or an attack using a stored list of values, or, more specifically, using dictionary words.

D.2.1.13 Password space and entropy

A *password space* may be defined as the set of all passwords in a given system and operating environment, which generally depends on password size, construction, and other limitations of the system and its human participants. The size of the password space is simply the number of elements in the space. But this definition does not necessarily say much about how these elements are distributed in a given population of users. The *entropy* of a password is, theoretically, a measure of the uncertainty of the specific value that the password may be assigned, which is, roughly speaking, inversely related to the probability of guessing the value. But this concept may be difficult to use in practice, when it relates to psychological factors of how a password is selected, and further depends on the state of knowledge and capabilities of the guesser (see D.2.1.14). A subset of a user population may be at extreme risk even if the size of the password space is large, and even if the average password entropy is large; attackers may be able to focus their efforts on a vulnerable subset of the space.

For the purposes of this document, password entropy is presumed to be small, that is, within the computational ability of an adversary to perform a systematic search of the effective password space for a non-negligible subset of the users of the system.

Further discussion of how password strength is related to password selection is in D.2.1.14.

D.2.1.14 Password selection

Several competing factors should guide how a password is selected in a given system, as follows:

— Ease of remembering the password

— Ease of entering the password

— Resistance to guessing attack, within the constraints of the password system

The decisions to use randomly-generated or allow human chosen passwords, and of how to enforce specific size and construction of passwords, should be made in light of the overall design of the password system. These decisions include consideration of the system's ability to

— Detect failed access attempts,

— Disable or limit access for excessive failed access attempts, to constrain guessing attack, and

— Provide an alternate access strategy for forgotten or disabled passwords.

Statistics for the success of password guessing in a common login application may be found in Morris and Thompson [B46] and Klein [B33]. Further discussion of how one might attempt to quantitatively assess password strength for human chosen or randomly generated passwords may be found in Appendix A of NIST SP 800-63 [B47]. An example of an automated password generator is FIPS PUB 181 [B13].

NOTE—(*FIPS 181*) The password generator specified in FIPS 181 has the odd property of creating passwords with letter combinations in a similar frequency distribution to that found in English words. This practice reduces the entropy and makes it difficult to assess exactly what the entropy is in the generated password. No argument is given to suggest that, for example, using E more frequently than O, and S more frequently than Z, makes passwords any easier to remember. Given the human tendency to notice the unusual, one might even suspect that this practice has the reverse effect of making the average generated password less distinctive, and thus harder to remember.

D.2.1.15 Password verification data

Password verification data is typically stored in a *password database*, in association with user names and other user account information. If passwords were always high-grade secrets, one could responsibly use ordinary public-key techniques with a (suitably-formatted) password as the private key. But anyone with a list of all possible passwords, whether they are in possession of this list or have the ability to enumerate such a list, has the ability to verify whether the password-limited private key corresponds to the public key, and thus determine the password. Thus, it is a standard practice in secure systems to keep password verification data private.

To supplement the privacy mechanisms for password verification data, it is also common practice to use a password verification data derived from a one-way function of the password (e.g., a *hashed password*). It is also common to use a deliberately slow one-way function, such as an iterated hash (see PBKDF2 in PKCS #5 IETF RFC 2898 [B23]), or a repeatedly encrypted password. The general idea is to make it hard for an entity in possession of password verification data to determine the corresponding password by any means that is easier than by trial-and-error comparison. This practice is useful in both stand-alone and network systems for password verification. In a stand-alone verification system, a privileged process accepts a password (and salt, as discussed in D.2.1.15.2) as input, applies the one-way function, and compares the result to the stored password verification data. In a network verification system, a client accepts the password (and salt) as input, and performs a verification process in an interaction with a remote server that has access to the stored verification data. This standard is concerned only with the network setting, and includes a class of methods, APKAS, that define specialized one-way functions for computing the password verification data.

NOTE—When a deliberately slow hash function is used to pre-compute password verification data stored on a Server, only the Client needs to compute the hash during each run.

D.2.1.15.1 Storing password data

In general, all password-based systems should be designed to minimize the opportunity for access to sensitive password-derived data. The following general advice raises issues to consider when using the PKAS defined in this document:

— It is best to keep stored password-derived data unavailable to administrators of the system. This generally applies to both the password values π and password verification data derived from π.

— Another wise common practice is for systems to use and store password verification data that has been constructed using a one-way function of the password, in order to prevent administrators and abusers of the system from having access to password data in the forms that are easiest to memorize or use. Each APKAS defines a one-way function for computing password verification data. However, even when using a BPKAS, one may choose to define π as a hash of the password, to insure that the stored value is a one-way function of the password value that is entered by the user.

— Recognizing that passwords are low-grade secrets, it is also a wise practice to make the one-way function deliberately slow to somewhat increase the difficulty of brute-force attack on stolen verification data.

D.2.1.15.2 Incorporating salt in password verification data

Salt is a general term for (typically, non-secret) data that supplements a password in input to a one-way function that generates password verification data to make different instances of the function applied to the same input password produce different outputs. The practice of applying salt increases the cost of wide-scale brute-force attack on multiple accounts using a pre-computed set of values.

A salt value may be incorporated in a PKAS either as part of the input value π or as a key derivation parameter value P_i.

D.2.1.15.3 Incorporating party identifiers in password verification data

Incorporating identifiers for one or both parties in the one-way function is also a sensible practice to help prevent identity confusion attacks (see D.2.1.10).

In an APKAS it is recommended to incorporate the Client identifier in π, so as to prevent an adversary from obtaining a single set (or dictionary) of password verification data that is applicable to multiple Clients if the password file on the Server is ever compromised. (A suitably distinctive salt value may be considered as a form of party identifier.) It is also recommended to incorporate Client and Server identifiers in either π or the key derivation parameters, so as to prevent various spoofing attacks involving Clients that use the same password on multiple Servers, or Clients that use the same passwords as other Clients.

D.2.1.15.4 Incorporating keys in password verification data

Incorporating a secret key in the one-way function is a way to add an extra factor for authentication, to add extra strength to the system. This is commonly used in systems where password verification data may be exposed to public scrutiny, such as when an encrypted PIN is stored on a magnetic stripe of a banking teller machine access card, where the verifier is trusted to keep the encryption key secret.

D.2.1.16 Random private keys

Where PKAS select a random private key from the range of valid private keys, the same issues of private key generation apply as are generally noted for key agreement schemes. IEEE Std 1363-2000 D.4.1.2 and D.4.2.2 discuss private key generation for the DL and EC families, and both of these notes refer to IEEE Std 1363-2000, D.4.1.4 NOTE 7, which is repeated here:

Private keys: The private key should be generated at random from the range [1, r–1] since this maximizes the difficulty of recovering the private key by collision-search methods. A desired level of security can also be provided when the private key is restricted to a large enough subset of the range(e.g., is shorter than the subgroup order, has low weight, or has some other structure). Such choices require further security analysis by the implementer. (See also [IEEE Std 1363-2000] D.6 for more on random number generation.)

How to select a private key from a uniformly random distribution over the full range of private keys is further discussed in IEEE Std 1363a-2004, D.5.2.1 NOTE 3. One way to assure uniformity of a distribution across [1, r–1] to any desired level of satisfaction defined by a security parameter b, is to choose a random l-bit string, where $l = b + \lceil \log_2 r \rceil$, and then reduce the result modulo r. (See NOTE.)

However, performance may be improved in some DL settings by using a shorter private key and relying on the short exponent assumption (see D.2.1.4).

An implementer should analyze these performance and security issues when choosing to use a restricted range of private keys.

NOTE—The previous discussion presumes that r is the order of the desired group. For the SRP3 and SRP6 schemes, where the desired group order is q–1, the full range of private keys is [1, q–2].

D.2.1.17 Password-limited private keys

Many PKAS compute integers using a hash function of a password-based value to produce a private key, to be used in a group operation. These password-limited private keys may be subject to the usual concerns with random private key selection, as discussed in D.2.1.16. But, notably, the length of the output of a *HashPVD* or *HashRE* function that creates a password-limited private key, and the randomness of its password-based input value, impose upper bounds on the randomness of the resulting key.

Ideally, the output length of the hash function should be greater than or equal to $\lceil \log_8(r) \rceil$ (assuming that [1, r–1] defines the range of valid private keys). One might even choose to make the hash output length somewhat greater than $\lceil \log_8(r) \rceil$ to eliminate bias in high-order bits. However, such concerns over the size of the hash output may be mitigated because the randomness of a password-limited private key is ultimately limited by the randomness of its password-based input. Since each APKAS is designed to prevent unauthorized disclosure of its password-limited values, in some applications it may be reasonable to use a *HashPVD* or *HashRE* whose output length corresponds to the size of the password space.

D.2.1.18 Range of password-based field elements

Many PKAS compute integers using a hash function of a password-based value to produce a field element, as in the use of hash functions to derive password-based field element in an REDP. Notably, the length of the output of such a *HashRE* function imposes an upper bound on the randomness of the resulting field element.

Ideally, the output length of *HashRE* should be greater than or equal to $\lceil \log_8(q) \rceil$. And one might choose to make the hash output length somewhat greater than $\lceil \log_8(q) \rceil$ to eliminate bias in high-order bits. However, concern over the size of the hash function may be mitigated because the randomness of a password-based field element is also limited by the randomness of its password-based input, and because each APKAS is designed to prevent unauthorized disclosure of these password-based values.

D.2.1.19 Password registration

In using these password-based methods, one first establishes the password or related data on one or more machines, which may involve protocols and procedures that provide a different basis for security of the password. It is important to ensure that the level of security of the password establishment process is appropriate to the application.

One common practice to minimize password exposure when using relatively insecure registration processes (e.g., insecure email delivery) is to first enroll with a temporary password, which is later replaced with a longer-lived password selected by the user. The privileges enabled by the temporary password may be restricted and, perhaps, limited for the sole purpose of setting a new password during an act of enrollment.

It may be appropriate and reasonable to require the user to take additional steps during initial registration than would otherwise be required in everyday login. For example, registration may be protected using a certificate-based public-key system, where the user may be encouraged or required to check name and certificate bindings.

D.2.1.20 Security properties of random element derivation primitives

Each REDP ensures that the discrete log of any output value, with respect to any fixed element of the group of order r defined by the domain parameters, cannot be practically computed. This property should hold regardless of any known relationship between the selector input values. The ability to construct an output value with a known DL should require the ability to solve a hard problem, like inverting a hash function or solving discrete log.

The concept of using a hash function to ensure that there are no known exponential relationships between values was discussed in Vaudenay [B53]. Vaudenay suggested applying a hash function to a public seed value in order to derive a g parameter value for a signature scheme to thwart potential attacks on the scheme by an adversary who knows exponential relationships for g.

Each REDP meets a superset of more specific security requirements of different schemes in this document. See D.2.2.6.1 for the PAK family, D.2.2.2.2 for the SPEKE family, D.2.2.5.2 for APKAS-SRP5, and D.2.2.3.5 for PKRS-1.

D.2.1.21 Limiting password guesses

This subclause discusses some issues for applications that impose a limit for online password guessing.

Good vs. bad guesses. If a general limit is imposed, but bad guesses are not separately identified, then correct guesses would have to be included in the limit. This would require special action (such as requiring the password to be changed) when the general use limit is exceeded.

Mistakes vs. attacks. A system may try to discriminate between bad guesses that are (likely to be) honest mistakes and those that are more likely to be attacks, based on the source address, time, and/or frequency of access attempts. A discussion of one discriminating error handling system that uses cryptographic data to distinguish mistakes from attacks is in Jablon [B29].

Incomplete runs: When using a scheme where the Server provides the first proof of knowledge of an agreed key, as is required by the PAK and PAKZ schemes (see discussion of unilateral commitment in D.2.1.9), the failure of a Client to complete the protocol may indicate a bad guess by an adversary and may need to be counted as such by the Server.

Denial of service. Any limit should be considered in light of the trade-off that a lower limit makes denial of service attacks easier, and a higher limit requires more randomness in the password.

Group passwords. One of the features of PKRS-1 is "blinding," which makes it possible for the Client to remain an (anonymous) member of a group sharing the same password. A query limit for a group password would need to accommodate all users, which may motivate a significantly larger guessing limit.

Parallel guessing. When multiple Servers can simultaneously run a protocol using a common password, care may need to be taken to ensure that a malicious Client cannot take undue advantage of them to exceed bad guess limits. For example, consider a Client-initiated 3-pass password-authenticated key agreement protocol, where message 1 conveys the Client commitment, message 2 conveys both the Server commitment and the Server's proof for key confirmation, and message 3 conveys the Client's proof. A parallel attack by a Client may send message 1 to multiple Servers in parallel, and abort each run after receiving message 2. This concern generally applies to any protocol where the Server is the first to send a confirmation message conveying the proof of knowledge of the password. The system should (somehow) impose a limit on the number of Servers that can be abused in this manner.

In the 3-pass case, each unfinished (or aborted) session represents a potential password guess by a malicious Client. In contrast, consider a 4-pass protocol in which the Client sends the first proof, which requires the Server(s) to merely limit the number of aborted sessions, regardless of how many unfinished sessions are open.

It is also illuminating to consider the approach where the Servers report failures to a central server, and when the number of failures is too high, the central server broadcasts a revocation message to the local Servers. In a 3-pass mode, local Servers would have to report every open session to the central server, as well as the close of every session. This is more communication than is needed for the 4-pass case. The 3-pass central server might also need to maintain a count of unfinished and aborted sessions.

D.2.1.22 Comparison to IETF RFC 2945

APKAS-SRP3 is comparable to and partly compatible with IETF RFC 2945 [B24], with differences as noted in 9.8 NOTE 5.

D.2.1.23 Comparison to ISO/IEC 11770-4:2006

Several methods in this standard are comparable to methods described in ISO/IEC 11770-4:2006 [B26], "Key management: Mechanisms based on weak secrets." This ISO/IEC standard defines Key Agreement Mechanisms 1, 2, and 3, and Key Retrieval Mechanism 1, which are roughly comparable to BPKAS-SPEKE, APKAS-SRP6, APKAS-AMP, and PKRS-1, respectively. However, several differences in ISO/IEC 11770-4:2006 prevent consistent interoperability of any of these mechanisms with their IEEE 1363.2 counterparts.

Differences between these standards include the following:

— ISO/IEC 11770-4:2006 defines and uses the I2OS(x), FE2OS(x), and GE2OSx(x) primitives, in which the output string lengths depend on the value of x. In contrast, this standard uses fixed-length outputs with the comparable I2OSP, FE2OSP, and GE2SVFEP primitives. These methods are not interoperable whenever the high byte of the input value x is zero.

— ISO/IEC11770-4:2006 uses its GE2OSx(x) primitive in place of this standard's GE2OSP-X(x), which further prevents interoperability in EC settings, since GE2OSx does not output a prefix byte.

— ECREDP-1 will not interoperate with the comparable R_{1EC}, due to the omission of a hash step in R_{1EC} and other differences.

— BPKAS-SPEKE (see 9.4 NOTE 9) and PKRS-1 (see 10.2 NOTE 9) may have limited but unpredictable interoperability with their 11770-4 counterparts, depending on whether certain values have their high byte set to zero.

— APKAS-AMP (see 9.5 NOTE 6) and APKAS-SRP6 (see 9.8 NOTE 6) have other significant differences than their ISO/IEC 11770-4 counterparts that prevent any interoperability.

— ISO/IEC 11770-4:2006 restricts DL domain parameters to prime q and $k = 2p_1p_2...p_t$, where $t \geq 0$ and all factors $p_i > r$.

— ISO/IEC 11770-4:2006 restricts EC domain parameters to prime q or $q = 2^m$, and $k = \{1,2,$ or $4\}p_1p_2...p_t$, where $t \geq 0$ and all factors $p_i > r$.

— ISO/IEC 11770-4:2006 specifies a slightly different set of recommended hash functions.

D.2.2 Considerations for specific password schemes (PKAS and PKRS)

D.2.2.1 Considerations for APKAS-AMP

APKAS-AMP and its related primitives perform validity checks on several received, transmitted, and computed values to prevent a variety of potential attacks. Some of these potential attacks are described here. Others may be found in Kwon [B39], [B36], and Wan and Wang [B54].

AMP is a unilateral commitment scheme, as discussed in D.2.1.9.

D.2.2.1.1 Selection of domain parameters for AMP

For DL settings, it is strongly recommended that $q-1$ has no factors in the range $[3, r-1]$, and for EC settings, it is recommended that k be either 1, 2, or 4, in order to prevent the Pohlig-Hellman decomposition attack discussed in D.2.2.1.3. Such settings may also simplify the required checks for unacceptably small order elements as discussed in D.2.2.1.2 and D.2.2.1.4. These recommendations are in alignment with those for a similar scheme in ISO/IEC 11770-4:2006 [B26] (see also APKAS-AMP NOTE 6).

D.2.2.1.2 Client validation of Server's public key

The Client tests the Server's public key w_S and aborts if it is not in the parent group or is an element of unacceptably small order. This prevents the Client from computing a small order z, which could allow the Server to guess z without knowing the password verification data value v_π. See D.2.1.5 for the meaning of "unacceptably small" and further discussion of small subgroup confinement. To avoid computation in validating that w_S is a large order element of the parent group, it is recommended that the domain parameters be selected as described in D.2.2.1.1.

D.2.2.1.3 Server validation of Client's public key

The Server tests the Client's public key and aborts if it is not in the parent group. This prevents unexpected values from being passed to implementations of PEPKGP-AMP-SERVER or SVDP-AMP-SERVER, the latter of which might compute $z = 0$ in a DL setting.

Furthermore, when the order of $((w_C \wedge i_1) * v_\pi)$ is a composite with multiple factors significantly smaller than r, the method may be vulnerable to the Pohlig-Hellman decomposition attack discussed in D.2.1.6. To preclude such attack, the Server may check w_C and abort when it is not a valid public key. Alternately, to

avoid computational expense of validating w_C, it is recommended that the domain parameters be selected to preclude such attack, as described in D.2.2.1.1.

D.2.2.1.4 Server validation of shared secret key

SVDP-AMP-SERVER checks the order of its computed shared secret group element z_g and returns "invalid" if the order is unacceptably small. Without this step, an adversary posing as a Client could choose a small order element e, send $w_C = e/g$ to the APKAS-AMP Server, and confine z_g to a small group, and thus determine the APKAS-AMP Server's value for z without knowledge of π. See D.2.1.5 for the meaning of "unacceptably small" and further discussion of small subgroup confinement.

D.2.2.1.5 Need for *HashWC* in AMP

The *HashWC* function in APKAS-AMP (which was used in AMP$^+$ in Kwon [B39] and in AMP2 in Kwon [B38]), prevents a vulnerability noted in Kwon [B37] that applied to an earlier draft version of APKAS-AMP that was based on AMP of Kwon [B36]. The earlier version permitted an adversary who obtained the password verification data to masquerade as the Client without performing a dictionary attack, thus removing the augmented benefit.

D.2.2.1.6 Criteria for selecting *HashWC* in AMP

The *HashWC* function should be selected such that it outputs at least $\lceil \log_{256}(r) \rceil$ octets.

It is recommended in Kwon [B35] and [B34] that Client and Server identifiers be included in the o_{ID} input parameter for *HashWC*.

D.2.2.2 Considerations for SPEKE schemes

D.2.2.2.1 Key validation in SPEKE schemes

BPKAS-SPEKE, APKAS-BSPEKE2, and APKAS-WSPEKE include steps for verifying that the order of a received password entangled key w' is not unacceptably small, in order to defend against small subgroup confinement attack. The acceptable values for a received public key w' in the BPKAS-SPEKE, APKAS-BSPEKE2, and APKAS-WSPEKE methods are the elements of the parent group that generate any sufficiently large group. See D.2.1.5 for the meaning of "unacceptably small" and further discussion of small subgroup confinement.

D.2.2.2.2 Security properties of *Redp* for the SPEKE family

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.2.1.20. The SPEKE family of key agreement schemes (BPKAS-SPEKE, APKAS-BSPEKE2, and APKAS-WSPEKE) require that, for any two REDP output values x and y, it is impractical to compute the discrete log base x of y.

The REDP functions ensure that there is no known exponential relationship between two or more generators that correspond to known candidate passwords. Without this characteristic, attack against SPEKE (as it was described in Jablon [B31]) would otherwise be possible for an adversary who knows such an exponential relationship. Instances of such an attack are the "password-in-exponent" attack

described in Jablon [B28] and the attacks of Scott [B50] and Zhang [B58], which apply particularly to generators based on small unhashed passwords (see NOTE 1). The REDP function prevents such attack.

NOTE 1—The attacker would first establish a set of candidate passwords $\{\pi_0, \pi_1, \pi_2, ...\}$ and a corresponding set of generators $\{g_0, g_1, g_2, ...\}$ and exponent values $\{1, e_1, e_2, ...\}$ where each $g_i = f(\pi_i)$ and each $g_i = g_0 \hat{} e_i$. The attacker would then run the method just once using g_0 to derive candidate shared key z_0, and to obtain $Kcf(z')$, a key confirmation function of the other party's derived key. The remaining candidates $\{z_1, z_2, ...\}$ could be derived off-line using $z_i = z_0 \hat{} (e_i^{-1} \bmod r)$, which permits each $Kcf(z_i)$ to be verified against $Kcf(z')$.

NOTE 2—Discussion of the relative advantages of REDP-1 over REDP-2 and vice versa is in C.2.1.

D.2.2.2.3 Criteria for selecting *HashWS* in WSPEKE SVDP

The *HashWS* function should be selected such that it outputs at least $\lceil \lceil \log_{256}(r) \rceil / 2 \rceil$ octets.

D.2.2.2.4 Reduction argument for SPEKE

An argument that reduces the security of a form of SPEKE (see NOTE) to a *Decision Inverted-Additive Diffie-Hellman* (DIDH) assumption is presented in MacKenzie [B42]. Although there is no described relation between DIDH and the well-known Decision Diffie-Hellman (DDH) assumption, a lower bound for breaking DIDH is presented, similar to a lower-bound proof for DDH, and an analogous *Computational Inverted-Additive Diffie-Hellman* (CIDH) problem is shown to be equivalent to Computational Diffie-Hellman (CDH).

NOTE—The form of SPEKE analyzed in MacKenzie [B42] is essentially the same as BPKAS-SPEKE when using a DL $GF(p)$ setting with a safe prime p and when Client and Server identifiers are included as inputs to the KCFs (for example, by incorporating the identifiers in π).

D.2.2.3 Considerations for PKRS-1

PKRS-1 uses a blinding function and a permutation function to derive a permuted key for a Client that possesses a password with the assistance of a Server that possesses a static private key that is associated with the password. The Client uses the blinding function to create a blinded password-limited public key for the Server. The Server uses the static private key in combination with the blinded password-limited public key to derive a permuted blinded key. The Client uses an unblinding function, the inverse of the blinding function, to derive the permuted key from the permuted blinded key. In this process, the password is not revealed to the Server or to any other party, and the associated private key is not revealed to the Client or to any other party.

An application protocol that uses PKRS-1 has the following form: The Client transmits a message incorporating a blinded password-limited public key to the Server. The Server returns to the Client the permuted form of the blinded key. The Client unblinds the result to retrieve the permuted key and derives one or more keys from the result.

Note that a single PKRS-1 Server may succeed with a brute-force password attack if it has access to the keys retrieved by its PKRS-1 Client, by simply combining its private key with candidate guesses for the password. However, when the Client uses two or more distinct PKRS-1 Servers (with distinct associated private keys), and when the Client does not reveal the individual retrieved keys but instead uses multiple retrieved keys to derive a combined key, the combined key resists brute-force password attack unless all of the requisite Server's private keys are compromised.

D.2.2.3.1 Domain parameter selection for PKRS-1

For use of DL settings with a PKRS-1 Server that does not perform the optional validity check of the Client's public key, it is recommended that cofactor k not have any factors other than 1 or 2 that are smaller than r, in order to prevent a Pohlig-Hellman decomposition attack that might disclose the Server's private key, as discussed in D.2.1.6. The use of an appropriately constrained group can limit the disclosure of information about the Server's private key to the low 1 bit. One should further consider the issue of Static Diffie-Hellman Problem (SDHP) attack as discussed in Brown and Gallant [B9] and D.2.2.3.6, which may motivate another constraint for the domain parameters. See also D.2.2.3.2 and D.2.2.3.3 for related discussion of key validation.

NOTE—See A.1.1 for how to create DL $GF(p)$ domain parameters with a variety of recommended constraints.

D.2.2.3.2 Key validation in PKRS-1 Client

The PKRS-1 Client key establishment operation includes a step to ensure that received permuted blinded password w_S is a member of the parent group defined by the domain parameters. This guarantees that w_S is appropriate for use in the subsequent KRUP-1 operation (see NOTE). The step to validate that w_S is a valid public key is optional and specified only to accommodate implementations that might choose to do so. Full validity checking of w_S is not generally required for the scheme, since it is expected that any invoking protocol or application will confirm that the value of the permuted password z_g (or a key derived from z_g) is *correct* before revealing information derived from z_g. However, the method for confirming the value of z_g may create further requirements for validation of w_S, as discussed in .

NOTE—The protocols in Ford and Kaliski [B16] and Jablon [B29], which use a scheme compatible with PKRS-1, do not verify that the Client's received w_S or the Server's received w_C are members of the parent group. In these methods, the domain is restricted to DL $GF(p)$ settings where all values of w_C and w_S are assumed to be integers, and in which the result of the subsequent mathematical operations (equivalent to KRPP-1 and KRUP-1) force the resulting integer value to be either zero or an element of the parent group. In these cases, elements that are not in the parent group that may be submitted by an adversary have no detrimental effect on the security of the scheme. However, the issue of whether a step to check membership in the parent group can be omitted to produce a secure (functionally equivalent) scheme depends, in part, on how the implementations of KRPP-1 and KRUP-1 operate on values that are not in the parent group. This issue would need careful study by an implementer who desires to omit such verification steps.

D.2.2.3.3 Key validation in PKRS-1 Server

The PKRS-1 Server key establishment operation includes steps to verify that the received blinded password w_C is acceptable, to make subsequent KRPP-1 operation well-defined, and to defend against a specific attack.

The PKRS-1 Server validates that w_C is a member of the parent group, since the KRPP-1 operation is only defined for values that are in the parent group.

If the order of w_C contains too many small factors, an attacking Client may be able to perform a Pohlig-Hellman decomposition attack to determine the Server's private key u (see D.2.1.6 and IEEE Std 1363-2000, D.5.1.6). This attack is prevented in every DL or EC setting when the Server checks, in the (optional) step 3, whether w_C is a valid public key (see A.1.5), and aborts when w_C is invalid. This attack is also sufficiently mitigated by the selection of domain parameters as described in D.2.2.3.1.

D.2.2.3.4 Application considerations for PKRS-1

D.2.2.3.4.1 Use with multiple servers

Both Ford and Kaliski [B16] and Jablon [B29] describe use of a single PKRS-1-CLIENT with multiple independent PKRS-1-SERVERs, where the individual keys established with each Server are combined into a single Client *master key*. A variety of techniques might be used to derive the master key from multiple PKRS-1 retrieved keys and other components. Alternatively, the master key could be derived from a single PKRS-1 retrieved key and a component stored on another non-PKRS-1 server, where the Client demonstrates knowledge of the retrieved key to the non-PKRS-1 server in order to obtain the component (see NOTE). This variant protects the password against compromise of the PKRS-1 server. The master key could then be verified by the Client, perhaps using additional information stored on one or more of the Servers, and then used to derive static secret keys that unlock sensitive user data.

NOTE—The retrieved key itself should not be revealed to the non-PKRS-1 server, otherwise the non-PKRS-1 server has enough information itself (retrieved key + component) to determine the master key. Instead, a variant of the retrieved key could be used as an authentication key in some proof of knowledge protocol that does not reveal the retrieved key.

D.2.2.3.4.2 Need for key confirmation

In order to prevent disclosure of π to an untrusted party that knows u, or one that knows w_C and controls w_S, an application protocol that invokes PKRS-1-CLIENT needs to confirm that the permuted password z_g (or equivalently, a value derived from z_g, such as Z or K_i) is correct before the Client reveals information about z_g to untrusted parties.

However, unlike the key agreement schemes in this standard, PKRS-1 does not include a key confirmation operation. Methods for key confirmation may vary considerably, such as those described in Ford and Kaliski [B16] and Jablon [B29], which derive a key from a PKRS-1 established key in combination with other components. Depending on the method, further validation of w_S may be required, as shown in the following example.

Consider a PKRS-1 application where the Client attempts to confirm that its computed Z is correct by verifying that $Hash(Z) = o_{KCF}$, where o_{KCF} is a key confirmation value received from the Server. Let's call the correct key confirmation value o_{KCF}', which equals $Hash(Z')$, where Z' = GE2OSP-X(g_π^u). But, prior to key confirmation, this Client cannot assume that w_S is valid or that $o_{KCF} = o_{KCF}'$. Without further validation of w_S (other than it being in the parent group), an adversary could provide an arbitrary w_S value of small order, creating a significant chance for the adversary to further provide the Client with an o_{KCF} equal to $Hash(Z)$, and thus obtain the Client's predicted retrieved key value(s), even though $o_{KCF} \neq o_{KCF}$, $Z \neq Z'$, and the adversary has no knowledge of u.

Without further validation of w_S, it is evident that this $Hash(Z)$ method does *not* confirm that the Client's retrieved key is correct. An implementer may address this problem by making the Client abort when w_S is of small order, perhaps using Key Establishment step e2). Alternately, an implementer may use a key confirmation technique described in the aforementioned references, such as:

— Using $o_{KCF} = Hash(Z_1, Z_2, \ldots Z_n, \pi)$, where the Z_i values are derived using PKRS-1 with n distinct Servers, which prevents the Server from fooling the Client without first correctly guessing the password, or

— Using a separately secured channel to protect the integrity of o_{KCF}.

The extent to which the Client trusts the Server and the ways in which such trust is established or verified may vary depending on the application, and are beyond the scope of PKRS-1.

D.2.2.3.5 Security properties of *Redp* for PKRS-1

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.2.1.20. The PKRS-1 key retrieval scheme requires that, for any two REDP output values x and y, it is impractical to compute the DL base x of y.

D.2.2.3.6 SDHP attack on PKRS-1

This subclause addresses an attack on PKRS-1 based on the Static Diffie-Hellman Problem (SDHP) discussed in Brown and Gallant [B9]. PKRS-1 provides a DH oracle to an adversary, which is both a theoretical concern for security analysis and has been at least potentially exploited in the SDHP attack of Brown and Gallant [B9]. This attack improves the ability for an adversary to learn a static DH secret key x, given access to an oracle that provides g^x for submitted values of g. The significance of this new result depends on the domain parameters and on how the scheme is used.

In an extreme case of the SDHP attack that minimizes the adversary's off-line effort (described in Brown and Gallant [B9]), the effort is reduced from about $r^{1/2}$ operations to about $r^{1/3}$, where r is the order of g. However, this case would require about $r^{1/3}$ oracle queries, which may be unrealistically large for some applications.

Some distinct ways to address this attack are:

— Use a bigger group. For instance, use 240 bits instead of 160. Or if using an oversized group, such as a DL 1023 bit $r = (q-1)/2$, it is OK as is.

— Make sure $(r-1)$ has no factors in the critical range of $[C, r^{1/2}]$. At the low end, C is a small "cutoff" value (see Brown and Gallant [B9]). At the high end, $r^{1/2}$ is probably overkill, but easy to remember. When there is no factor in this range, the new attack provides no advantage over earlier methods. One suggestion is for $(r-1)/2$ to be prime. [Note that this is different from whether one uses DL $GF(p)$ with cofactor $k = (p-1)/r = 2$.]

— Make sure that the number of oracle queries that the PKRS-1 Server provides to an adversary is limited to less than that required for the new attack. This depends on the factorization of $(r-1)$, and on the cutoff value.

The last solution might be natural for some applications. Although PKRS-1 does not require the Server to confirm the Client's retrieved key, a PKRS-1 Server application may perform this function to detect and limit online guessing of the password. (Ford and Kaliski [B16] discuss applications of PKRS-1 and recommend that guessing limits be imposed; further discussion of guessing limitations is in D.2.1.21.) If a small limit is enforced by the application, which is less than that required by the attack, the problem is prevented. While the issues of determining a guessing limit may be complex, for applications where the designer has already chosen a limit, the limit may be sufficiently small to prevent this SDHP attack.

D.2.2.4 Considerations for APKAS-SRP3 and APKAS-SRP6

SRP3 and SRP6 are unilateral commitment schemes, as discussed in D.2.1.9.

D.2.2.4.1 Constructing domain parameter g_{q-1} for SRP3 and SRP6

Algorithm A.1.3 may be used to construct DL domain parameter g_{q-1} of order $q-1$ for any finite field that is intended for use in SRP3 and SRP6. While other algorithms may be used, A.1.3 insures that algorithm A.1.4 will output "true," rather than "unknown," for validating g_{q-1}. (See D.2.2.4.2 for how to validate g_{q-1}.)

Verifiably pseudo-random domain parameters for the prime case $GF(p)$ where $p = 2r+1$ may be generated by the algorithm in A.1.1.

D.2.2.4.2 Validating domain parameter g_{q-1} for SRP3 and SRP6

Algorithm A.1.4 may be used to verify whether g_{q-1} is of multiplicative order $q-1$. Note that this algorithm may, in some cases, output the result "unknown," for reasons of efficiency. (See D.2.2.4.1 for how to construct g_{q-1}.)

Domain parameters that were generated in a verifiably pseudo-random manner by A.1.1 may be verified by the algorithm in A.1.2.

D.2.2.4.3 Criteria for selecting *HashW2* in SRP6

The *HashW2* function should be selected such that it outputs at least $\lceil \lceil \log_{256}(r) \rceil / 2 \rceil$ octets.

D.2.2.4.4 Criteria for selecting *Mvcf* in SRP6

The purpose of the multiplier value creation function *Mvcf* in SRP6 is to prevent a two-for-one guessing attack, as discussed in Wu [B56]. For such an attack to succeed against SRP6, the attacker would have to manipulate the input value for *Mvcf* to make the output correspond to a predetermined result. The *Mvcf* needs to have pre-image resistance and a suitable output size to make such attack infeasible. (See MVCF-DP in 12.5.1.)

D.2.2.5 Considerations for APKAS-SRP5

SRP5 is a unilateral commitment scheme, as discussed in D.2.1.9.

D.2.2.5.1 Criteria for selecting *HashW2* in SRP5 SVDP

The *HashW2* function should be selected such that it outputs at least $\lceil \lceil \log_{256}(r) \rceil / 2 \rceil$ octets.

D.2.2.5.2 Security properties of *Redp* for SRP5

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.2.1.20. The SRP5 key agreement scheme requires that, for any REDP output value x, it is impractical to compute the discrete log base g of x.

D.2.2.6 Considerations for PAK schemes

PPK is a bilateral commitment scheme and PAK and PAKZ are unilateral commitment schemes, as discussed in D.2.1.9.

D.2.2.6.1 Security properties of *Redp* for the PAK family

Each REDP meets a superset of the security requirements of different schemes in this document, as described in D.2.1.20. The PAK family of key agreement schemes (PAK, PPK, and PAKZ) require that, for any two REDP output values x and y, it is impractical to compute the discrete log base g of the quotient of x and y.

D.2.2.6.2 Reduction arguments

Reduction arguments for the security of the PAK and PPK protocols were sketched in Boyko, MacKenzie, Patel [B8]. Slightly modified protocols and new reduction arguments (using the security definition from Bellare and Rogaway [B5]) were presented in MacKenzie [B43]. These arguments are based on the random oracle model of Bellare and Rogaway [B5] and the Computational Diffie Hellman assumption. BPKAS-PAK and BPKAS-PPK are reasonably close instantiations of the slightly modified PAK and PPK protocols, respectively, when the message parameter o_{ID} incorporates identifiers for the Client and Server.

A PAK–Z+ protocol and an argument for its security is presented in Gentry, MacKenzie, Ramzan [B18]. The argument is based on the random oracle model of Bellare and Rogaway [B5], the Computational Diffie Hellman assumption, and the assumption that signature scheme *Ss* is existentially unforgeable against adaptive chosen message attacks. APKAS-PAKZ is a reasonably close instantiation of PAK–Z+ when the shared password-based value π and the message parameter o_{ID} incorporate identifiers for the Client and Server.

Annex E

(informative)

Formats

Some of the subclauses in IEEE Std 1363-2000, Annex E (*), are applicable to the methods in this document. Relevant portions of this clause are listed here:

E.1 Overview *

E.2 Representing basic data types as octet strings

E.2.1 Integers (I2OSP and OS2IP)

E.2.2 Finite field elements (FE2OSP and OS2FEP)

E.2.3 Elliptic curve points (EC2OSP and OS2ECP) *

E.2.3.1 Compressed elliptic curve points

E.2.4 Polynomials over $GF(p)$, $p \geq 2$ (PN2OSP and OS2PNP) *

E.3 Representing outputs of schemes as octet strings

E.3.1 Output data format for DL/EC signature schemes *

NOTE—The (*) asterisk denotes areas amended in IEEE Std 1363a-2004.

Annex F

(informative)

Bibliography

[B1] ANSI X9.30.1:1997, Public Key Cryptography Using Irreversible Algorithms—Part 1: The Digital Signature Algorithm (DSA).[7]

[B2] ANSI X9.42:2003, Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography.

[B3] ANSI X9.62:1998, Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA).

[B4] Bellare, M., and Goldreich, O., "On Defining Proofs of Knowledge," In Advances in Cryptology—CRYPTO '92 Proceedings, pp. 390–420, Lecture Notes in Computer Science, vol. 740. Springer-Verlag, Berlin, 1992.

[B5] Bellare, M., and Rogaway, P., "Random oracles are practical: a paradigm for designing efficient protocols," Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, pp. 62–73, ACM Press, November 1993.

[B6] Bellovin, S. M., and Merritt, M., "Augmented Encrypted Key Exchange: a Password-Based Protocol Secure Against Dictionary Attacks and Password File Compromise," AT&T Bell Laboratories (c. 1994).

[B7] Bellovin, S. M., and Merritt, M., "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks," *Proceedings of the I.E.E.E. Symposium on Research in Security and Privacy,* Oakland, May 1992.

[B8] Boyko, V., MacKenzie, P., and Patel, S., "Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman," Advances in Cryptology—EUROCRYPT 2000, Preneel, B. (ed.), May 14–18, 2000.

[B9] Brown, D., and Gallant, R., "The Static Diffie-Hellman Problem," November 15, 2004, preliminary version submitted to Eurocrypt 2005. Available at http://eprint.iacr.org/2004/306/.

[B10] Diffie, W., and Hellman, M., "New directions in cryptography," *IEEE Transactions on Information Theory,* vol. IT-22, no. 6, pp. 644–654, 1976.

[B11] Diffie, W., van Oorschot, P.C., and Wiener, M., "Authentication and Authenticated Key Exchanges," *Designs Codes and Cryptography,* vol. 2, 107–125, 1992.

[B12] FIPS PUB 180-1 (1995), Secure Hash Standard, Federal Information Processing Standards Publication 180-1, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, Virginia (supersedes FIPS PUB 180).[8]

[B13] FIPS PUB 181, Announcing the Standard for Automated Password Generator (APG), Federal Information Processing Standards Publication 181, U.S. Department of Commerce/National Institute of Standards and Technology, October 5, 1993.

[B14] FIPS PUB 186-2, Digital Signature Standard, Federal Information Processing Standards Publication 186-2, U.S. Department of Commerce/National Institute of Standards and Technology, National Technical Information Service, Springfield, Virginia, January 27, 2000 (supersedes FIPS PUB 186-1). Available at http://csrc.nist.gov/fips/.

[7] ANSI publications are available from the Sales Department, American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).
[8] FIPS publications are available from the National Technical Information Service (NTIS), U. S. Dept. of Commerce, 5285 Port Royal Rd., Springfield, VA 22161 (http://www.ntis.org/).

[B15] FIPS Draft 186-3, Digital Signature Standard (DSS), Draft Federal Information Processing Standards Publication 186-3, Information Technology Laboratory, National Institute of Standards and Technology, Gaithersburg, MD 20899-8900, March 13, 2006. (Planned to supersede FIPS PUB 186-2).

[B16] Ford, W., and Kaliski, B., "Server-Assisted Generation of a Strong Secret from a Password," *Proceedings of the IEEE 9th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises,* NIST, Gaithersburg MD, June 14–16, 2000.

[B17] Gennaro, R., "An Improved Pseudo-random Generator Based on Discrete Log," Lecture Notes in Computer Science, Volume 1880 / 2000, pp. 469–481, August 2000, Springer-Verlag Heidelberg, ISSN: 0302-9743, Proceedings of CRYPTO 2000, M. Bellare (Ed.).

[B18] Gentry, C., MacKenzie, P., and Ramzan, Z., "PAKZ+," Contribution to the IEEE P1363 Working Group, August 15, 2005. Available at http://grouper.ieee.org/groups/1363/passwdPK/contributions.html#GMR05b.

[B19] Goldreich, O., and Krawczyk, H., "On the composition of zero-knowledge proof systems," *SIAM Journal on Computing,* 25(1):169-192, February 1996.

[B20] Gong, L., et al., "Protecting Poorly Chosen Secrets from Guessing Attacks," *IEEE Journal on Selected Areas in Communications,* Vol. 11, No. 5, June 1993, pp. 648–656.

[B21] IEEE 100, *The Authoritative Dictionary of IEEE Standards Terms.* New York: Institute of Electrical and Electronics Engineers, Inc.[9]

[B22] IETF RFC 2631, Diffie-Hellman Key Agreement Method, Rescorla, E., June 1999. [10]

[B23] IETF RFC 2898, PKCS #5: Password-Based Cryptography Specification, Version 2.0, Kaliski, B., September 2000.

[B24] IETF RFC 2945, The SRP Authentication and Key Exchange System, Wu, T., September 2000.

[B25] ISO/IEC 10118-3:1998, Information technology—Security techniques—Hash functions—Part 3: Dedicated hash-functions. [11]

[B26] ISO/IEC 11770-4:2006, Information technology—Security techniques—Key management—Part 4: Mechanisms based on weak secrets. Available at http://www.iso.org/iso/en/CatalogueDetailPage.CatalogueDetail?CSNUMBER=39723.

[B27] Jablon, D., "B-SPEKE," Integrity Sciences white paper, September 1, 1999, available at www.integritysciences.com until early 2001.

[B28] Jablon, D., "Extended Password Key Exchange Protocols Immune to Dictionary Attacks," Proceedings of the Sixth Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET-ICE '97) IEEE Computer Society, June 18–20, 1997, Cambridge, MA, pp. 248–255.

[B29] Jablon, D., "Password Authentication Using Multiple Servers," LNCS 2020: Topics in Cryptology—CT-RSA 2001, April 8–12, 2001 Proceedings, pp. 344–360, 2001, Springer-Verlag.

[B30] Jablon, D., "SRP-4," a P1363.2 submission to the IEEE P1363 Working Group, May 9, 2002.

[B31] Jablon, D., "Strong Password-Only Authenticated Key Exchange," Computer Communication Review, ACM SIGCOMM, vol. 26, no. 5, pp. 5–26, October 1996.

[B32] Jablon, D., "The SPEKE Password-based Key Agreement Methods," draft-jablon-speke-02.txt, IETF Internet Draft, October 23, 2003.

[9] IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).
[10] IETF publications are available at www.ietf.org/rfc.
[11] ISO/IEC publications are available from the ISO Central Secretariat, Case Postale 56, 1 rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iso.ch/). ISO/IEC publications are also available in the United States from Global Engineering Documents, 15 Inverness Way East, Englewood, Colorado 80112, USA (http://global.ihs.com/). Electronic copies are available in the United States from the American National Standards Institute, 25 West 43rd Street, 4th Floor, New York, NY 10036, USA (http://www.ansi.org/).

[B33] Klein, D., "Foiling the cracker: A survey of, and improvements to, password security," *Proceedings of the USENIX UNIX Security Workshop,* pp. 5–14, Portland, OR, USA, August 1990.

[B34] Kwon, T., "Addendum to Summary of AMP," Submission to the IEEE P1363 Working Group, received November 20, 2003. Available at http://grouper.ieee.org/groups/1363/passwdPK/contributions.

[B35] Kwon, T., "Authentication and Key Agreement via Memorable Password," *NDSS 2001 Symposium Conference Proceedings,* February 7–9, 2001.

[B36] Kwon, T., "Authentication via Memorable Passwords—Revised Submission to IEEE P1363.2," Submission to the IEEE P1363 Working Group, received October 31, 2002. Available at http://grouper.ieee.org/groups/1363.

[B37] Kwon, T., "Revision of AMP in IEEE P1363.2 and ISO/IEC 11770-4," Contribution to IEEE P1363 Working Group, received June 8, 2005. Available at http://grouper.ieee.org/groups/1363.

[B38] Kwon, T., "Summary of AMP (Authentication and key agreement via Memorable Passwords)," Contribution to the IEEE P1363 Working Group, received August 22, 2003. Available at http://grouper.ieee.org/groups/1363.

[B39] Kwon, T., "Ultimate Solution to Authentication via Memorable Password," May 30, 2000. Submission to the IEEE P1363 Working Group, received June 22, 2000. Available at http://grouper.ieee.org/groups/1363.

[B40] Lenstra, A., and Haber, S., letter to NIST during the comment period on the then proposed DSA standard, November 26, 1991.

[B41] MacKenzie, P., "More Efficient Password-Authenticated Key Exchange," *LNCS 2020: Topics in Cryptology—CT-RSA 2001,* April 8–12, 2001 Proceedings, pp. 361–377, 2001, Springer-Verlag.

[B42] MacKenzie, P., "On the Security of the SPEKE Password-Authenticated Key Exchange Protocol," Cryptology ePrint Archive: Report 2001/057, received July 19, 2001. Available at http://eprint.iacr.org/2001/057/.

[B43] MacKenzie, P., "The PAK Suite: Protocols for Password-Authenticated Key Exchange," a P1363.2 submission to the IEEE P1363 Working Group, April 24, 2002.

[B44] MacKenzie, P., "The PAK Suite: Protocols for Password-Authenticated Key Exchange," DIMACS Technical Report 2002-46, October 2002.

[B45] Menezes, A., van Oorschot, P., and Vanstone, S., *Handbook of Applied Cryptography.* CRC Press, ISBN: 0-8493-8523-7, October 1996.

[B46] Morris, R., and Thompson, K., "UNIX password security," *Communications of the ACM,* v. 22, no. 11, p. 594–597, November 1979.

[B47] NIST SP 800-63, "Electronic Authentication Guideline," Burr, W., Dodson, D., and Polk, W., September 27, 2004. Available at http://csrc.nist.gov/publications/nistpubs/.[12]

[B48] Perlman, R., and Kaufman, C., "Secure Password-Based Protocol for Downloading a Private Key," *Proceedings of the 1999 Network and Distributed System Security,* February 3–5, 1999.

[B49] Scott, M., "MIKE—Mike's Integrated Key Exchange," Dublin City University, School of Computing, Working Paper CA-1300, November, 2000. Available at http://www.computing.dcu.ie/research/papers/2000.html.

[B50] Scott, M., "P1363: AMP protocol," IEEE P1363 Working Group mailing list post, July 19, 2001.

[B51] Scott, M., "SRP attack?" Cryptography mailing list post, May 4, 2001. Available at http://www.mail-archive.com/cryptography-digest@senator-bedfellow.mit.edu/msg05485.html.

[12] NIST publications are available from the National Institute of Standards and Technology, NIST Public Inquiries, NIST, 100 Bureau Drive, Stop 3460, Gaithersburg, MD, 20899-3460, USA (www.nist.gov).

[B52] van Oorschot, P. C., and Wiener, M., "On Diffie-Hellman Key Agreement with Short Exponents," *Proceedings of Eurocrypt '96,* LNCS 1070, pages 332–343, Springer-Verlag, May 1996.

[B53] Vaudenay, Serge, "Hidden Collisions on DSS," Proceedings of Crypto '96, Springer-Verlag, LNCS 1109, pp. 83–88, 1996.

[B54] Wan, Z., and Wang, S., "Cryptanalysis of Two Password-Authenticated Key Exchange Protocols," H. Wang et al. (eds.): ACISP 2004, LNCS 3108, pp. 164–175, Springer-Verlag, Berlin, Heidelberg, 2004.

[B55] Wang, Y., "IEEE P1363.2 Submission / D2001-06-21," [P1363.2-ecsrp-06-21.doc] A contribution by Yongge Wang for P1363.2 giving an elliptic curve version of the SRP protocol, June 21, 2001.

[B56] Wu, T., "SRP-6: Improvements and Refinements to the Secure Remote Password Protocol," Submission to IEEE P1363 Working Group, October 29, 2002.

[B57] Wu, T., "The Secure Remote Password Protocol," *Proceedings of the 1998 Internet Society Network and Distributed System Security Symposium,* San Diego, March 1998, pp. 97–111.

[B58] Zhang, M., "Analysis of the SPEKE password-authenticated key exchange protocol," *IEEE Communications Letters,* vol. 8, no. 1, pp. 63–65, January 2004, ISSN: 1089-7798.