INTERNATIONAL STANDARD

ISO/IEC 11770-7

First edition
2021-07

Information security — Key management —

Part 7: Cross-domain password-based authenticated key exchange

Sécurité de l'information — Gestion des clés —

Partie 7: Échange de clés authentifié entre mots de passe entre domaines

Reference number
ISO/IEC 11770-7:2021(E)

ISO copyright office
CP 401 • Ch. de Blandonnet 8
CH-1214 Vernier, Geneva
Phone: +41 22 749 01 11
Email: copyright@iso.org
Website: www.iso.org

Published in Switzerland

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work.

The procedures used to develop this document and those intended for its further maintenance are described in the ISO/IEC Directives, Part 1. In particular, the different approval criteria needed for the different types of document should be noted. This document was drafted in accordance with the editorial rules of the ISO/IEC Directives, Part 2 (see www.iso.org/directives).

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights. Details of any patent rights identified during the development of the document will be in the Introduction and/or on the ISO list of patent declarations received (see www.iso.org/patents) or the IEC list of patent declarations received (see http://patents.iec.ch).

Any trade name used in this document is information given for the convenience of users and does not constitute an endorsement.

For an explanation of the voluntary nature of standards, the meaning of ISO specific terms and expressions related to conformity assessment, as well as information about ISO's adherence to the World Trade Organization (WTO) principles in the Technical Barriers to Trade (TBT), see www.iso.org/iso/foreword.html.

This document was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *Information security, cybersecurity and privacy protection*.

A list of all parts in the ISO/IEC 11770 series can be found on the ISO website.

Any feedback or questions on this document should be directed to the user's national standards body. A complete listing of these bodies can be found at www.iso.org/members.html.

Introduction

In a security domain, two entities can authenticate each other and establish a shared session key to protect their communication. This authentication is typically based on pre-established information, such as a shared password or symmetric key or possession of each other's public key certificates. In a cross-domain communication, two entities assigned to two distinct security domains may not have suitable pre-established authentication information. However, they can still establish a shared session key by using the authentication information that each entity shares with its own domain server and relying on the domain servers themselves to authenticate each other.

Practical cross-domain communication scenarios include email communication, mobile phone communication, and instant messaging. In these cases, communications need to be protected against both passive and active attackers. In these scenarios, each entity is typically registered with a domain-specific server, such as an email exchange server (for email communications) or a home location register (for mobile phone communications). Moreover, the two communicating entities from different domains typically neither share a password or a symmetric key nor possess each other's public key certificate.

An authenticated key exchange (AKE) mechanism enables two entities to establish a shared session key based on their pre-established authentication information. A password-based AKE mechanism is based on two entities pre-sharing a password. Similarly, a symmetric key or an asymmetric key based AKE mechanism is based on two entities pre-sharing a secret key or possessing each other's public key certificate (and a trusted means to verify a certificate). In this document, these three types of mechanisms are referred to as two-party password-based authenticated key exchange (2PAKE) protocols, two-party symmetric key based authenticated key exchange (2SAKE) protocols and two-party asymmetric key based authenticated key exchange (2AAKE) protocols, respectively. 2PAKE protocols are specified in ISO/IEC 11770-4, 2SAKE protocols are specified in ISO/IEC 11770-2 and 2AAKE protocols are specified in ISO/IEC 11770-3. All the mechanisms specified in ISO/IEC 11770-1[6], ISO/IEC 11770-2 and ISO/IEC 11770-3 are appropriate for use in a single security domain. For example, the mechanisms specified in ISO/IEC 11770-4 are used in authenticated key exchange applications, where two players, usually referred to as a server and a client, are in the same security domain.

This document (i.e. ISO/IEC 11770-7) specifies cross-domain password-based authenticated key exchange mechanisms. Such mechanisms enable a user from one domain to establish a session key shared with another user from a different domain through their respective domain servers, and the only pre-established authentication information that each user has is a password shared with their domain server.

More specifically, each mechanism specified in this document involves four parties in two security domains, in which each user and server pair are in the same domain. This type of mechanism is referred to as a four-party password-based authenticated key exchange (4PAKE) protocol. This document contains a framework for designing such 4PAKE protocols using a compositional approach. That is, a 4PAKE protocol can be implemented based on two building blocks:

a) a 2PAKE protocol;

b) a 2SAKE protocol or a 2AAKE protocol.

This document also specifies several mechanisms for such 4PAKE protocols. The 2PAKE, 2SAKE and 2AAKE protocols used to implement such 4PAKE protocols are chosen from ISO/IEC 11770-4, ISO/IEC 11770-2 and ISO/IEC 11770-3 respectively.

The hash functions and key derivation functions used in the mechanisms specified in this document are specified in ISO/IEC 10118-3 and ISO/IEC 11770-6, respectively.

The conversion functions in Annex B used in the mechanisms specified in this document are specified in ISO/IEC JTC 1/SC 27 WG 2 SD 7 [16].

Information security — Key management —

Part 7: Cross-domain password-based authenticated key exchange

1 Scope

This document specifies mechanisms for cross-domain password-based authenticated key exchange, all of which are four-party password-based authenticated key exchange (4PAKE) protocols. Such protocols let two communicating entities establish a shared session key using just the login passwords that they share with their respective domain authentication servers. The authentication servers, assumed to be part of a standard public key infrastructure (PKI), act as ephemeral certification authorities (CAs) that certify key materials that the users can subsequently use to exchange and agree on as a session key.

This document does not specify the means to be used to establish a shared password between an entity and its corresponding domain server. This document also does not define the implementation of a PKI and the means for two distinct domain servers to exchange or verify their respective public key certificates.

2 Normative references

The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 11770-2, *IT Security techniques — Key management — Part 2: Mechanisms using symmetric techniques*

ISO/IEC 11770-3, *Information technology — Security techniques — Key management — Part 3: Mechanisms using asymmetric techniques*

ISO/IEC 11770-4, *Information technology — Security techniques — Key management — Part 4: Mechanisms based on weak secrets*

3 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 11770-4 and the following apply.

ISO and IEC maintain terminological databases for use in standardization at the following addresses:

— ISO Online browsing platform: available at https://www.iso.org/obp

— IEC Electropedia: available at https://www.electropedia.org/

3.1
asymmetric key pair
pair of related keys where the private key defines the private transformation and the public key defines the public transformation

[SOURCE: ISO/IEC 11770-3:2015, 3.3]

3.2
asymmetric-key authenticated key exchange
process of establishing one or more shared secret keys between two entities using asymmetric-key techniques and neither of them can predetermine the values of the shared secret keys

3.3
certification authority
entity trusted to create and assign *public key certificates* (3.9)

[SOURCE: ISO/IEC 11770-1:2010, 2.3]

3.4
cross-domain password-based authenticated key exchange
process of establishing one or more shared secret keys between two entities associated with two distinct *security domains* (3.10) using the entity's prior domain-specific password-based information such that neither of the entities can predetermine the values of the shared secret keys

3.5
distinguishing identifier
information which unambiguously distinguishes an entity

[SOURCE: ISO/IEC 11770-1:2010, 2.9]

3.6
key derivation function
function which takes as input a number of parameters, at least one of which is secret, and which gives as output keys appropriate for the intended algorithm(s) and applications

[SOURCE: ISO/IEC 11770-2:2018, 3.6, modified — Note 1 to entry has been removed.]

3.7
key establishment
process of making available a shared key to one or more entities, where the process includes key agreement or key transport

[SOURCE: ISO/IEC 11770-3:2015, 3.23]

3.8
non-interactive key exchange
process of establishing one or more shared secret keys between two entities in a non-interactive manner with mutual implicit key authentication

3.9
public key certificate
public key information of an entity signed by the *certification authority* (3.3)

[SOURCE: ISO/IEC 11770-1:2010, 2.37]

3.10
security domain
set of elements, security policy, security authority and set of security-relevant activities in which the set of elements are subject to the security policy for the specified activities, and the security policy is administered by the security authority for the security domain

[SOURCE: ISO/IEC 11770-1:2010, 2.43]

3.11
signature
data unit appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to verify the origin and integrity of the data unit and protect the sender and the recipient of the data unit against forgery by third parties, and the sender against forgery by the recipient

[SOURCE: ISO/IEC 11770-3:2015, 3.7, modified — the word "digital" has been removed from the term.]

3.12
symmetric-key
key used with symmetric cryptographic techniques and usable only by a set of specified entities

3.13
symmetric-key authenticated key exchange
process of establishing one or more shared secret keys between two entities using *symmetric-key* (3.12) techniques such that neither of the entities can predetermine the values of the shared secret keys

4 Symbols and abbreviated terms

4.1 Abbreviated terms

2AAKE	two-party asymmetric-key authenticated key exchange
2NIKE	two-party non-interactive key exchange protocol
2PAKE	two-party password-based authenticated key exchange
2SAKE	two-party symmetric-key authenticated key exchange
4PAKE	four-party password-based authenticated key exchange
BS2I	function that converts a bit string into an integer
BS2OS	function that converts a bit string to an octet string
CA	certification authority
FE2I	function that converts a field element to an integer
FE2OS	function that converts a field element to an octet string
$GE2OS_x$	function that converts a group element with x-coordinate to an octet string
I2BS	function that converts an integer to a bit string
I2OS	function that converts an integer to an octet string
KD	key derivation function
MAC	message authentication code
MAX	maximum value function
MIN	minimum value function
PKI	public key infrastructure
$Param_A$	2AAKE prior shared system parameters
$Param_N$	2NIKE prior shared system parameters

$Param_S$	2SAKE prior shared system parameters
$Param_P$	2PAKE prior shared system parameters
SIF	session identity generation function

4.2 Symbols

A, *B*	distinguishing clients' identities including their respective domain names represented as octet strings
$AK_{X, Y}$	symmetric authentication key shared between *X* and *Y*
aKG	authentication public/private key pair generation function
aKV	authentication public key validation function
apk_X	entity *X*'s authentication public key
ask_X	entity *X*'s authentication private key corresponding to apk_X
c_X	ciphertext generated from a symmetric encryption function by entity *X*
DEC(*K*, *c*)	symmetric decryption function taking a secret key *K* and a ciphertext *c* as input and giving as output a message *m* or a decryption failure symbol "⊥"
DL	discrete logarithm setting
EC	elliptic curve setting
ENC(*K*, *m*)	symmetric encryption function taking a secret key *K* and a variable-length message *m* as input and giving a ciphertext *c* as output, e.g. by using one of the symmetric encryption systems specified in ISO/IEC 18033-3[9] and ISO/IEC 18033-4[10]
$EK_{X, Y}$	symmetric encryption key shared between *X* and *Y*
eKG	ephemeral public/private key pair generation function
eKV	ephemeral public key validation function
epk_X	entity *X*'s ephemeral public key
esk_X	entity *X*'s ephemeral private key corresponding to epk_X
GE	group element under either discrete logarithm or elliptic curve setting
H	hash-function taking an octet string as input and giving a bit string as output, e.g. one of the dedicated hash-functions specified in ISO/IEC 10118-3[4]
h_X	entity *X*'s private key agreement key corresponding to p_X
$K_{X, Y}$	symmetric key shared between two entities *X* and *Y*
MAC(*K*, *m*)	MAC function taking a symmetric key *K* and a variable-length message *m* as input and giving a fixed-length cryptographic checksum μ_X as output, e.g., by using one of the MAC algorithms specified in ISO/IEC 9797-2[3]
MAX(*x*, *y*)	maximum value function taking two integers *x* and *y* as input and giving the maximum value between *x* and *y* as output

MIN(x, y)	minimum value function taking two integers x and y as input and giving the minimum value between x and y as output
MiX	step i performed by entity X in a mechanism
p_X	entity X's public key agreement key
$pwd_{S,C}$	password shared between a domain server S and a domain client C
S_A, S_B	distinguishing identities of domain servers of clients A and B, respectively, represented as octet strings
SI	group setting index
sid	session identity which is uniquely indicating to the session
SK_X	entity X's private signature key corresponding to VK_X
sn_A	session id contribution from entity A
TK	2SAKE symmetric authentication key
ts	timestamp specifying a start time and an end time
VK_X	entity X's public verification key
μ_X	cryptographic checksum generated from a MAC function by entity X
Σ	digital signature system
σ_X	digital signature generated from Σ.SIG by entity X
Σ.SIG(SK_X, m)	private signature transformation function taking entity X's private signature key SK_X and a variable-length message m as input and giving a digital signature σ_X as output, e.g. by using one of the digital signature systems specified in ISO/IEC 9796 (all parts)[2] and ISO/IEC 14888 (all parts)[8]
Σ.VER(VK_X, m, σ_X)	public signature verification function taking entity X's public verification key VK_X, a variable-length message m and a digital signature σ_X as input, and giving a single bit output: 0 (invalid) or 1 (valid)
||	X||Y denotes the result of the concatenation of octet strings X and Y in the order specified. In cases where the result of concatenating two or more octet strings is input to a cryptographic function as part of one of the mechanisms specified in this document, this result shall be composed so that it can be uniquely resolved into its constituent octet strings, i.e. so that there is no possibility of ambiguity in interpretation. This latter property can be achieved in a variety of different ways, depending on the application. For example, it can be guaranteed by a) fixing the length of each of the octet strings throughout the domain of use of the mechanism, or b) encoding the sequence of concatenated octet strings using a method that guarantees unique decoding, e.g. using the distinguished encoding rules defined in ISO/IEC 8825-1[1].
|y|	bit length of a binary string y

$\lceil x \rceil$	ceiling function taking a real number x as input and giving the least integer greater than or equal to x as output
$\lfloor x \rfloor$	floor function taking a real number x as input and giving the greatest integer less than or equal to x as output
0_E	point at infinity on an elliptic curve E

5 Requirements

Each security domain shall have a trusted domain server acting as an authentication server governing a group of entities. Each entity within the domain shall share a password with the server. The entity shall possess a copy of its server's credential, such as a public key certificate and the means to verify it, in order to verify messages generated by the server. However, it is not necessarily for the server credential to be distributed in advance. In practice, the servers can distribute their credentials to the entities within their respective domains during the execution of a cross-domain key exchange protocol as specified in this document.

Further, a domain server shall make its public key available to other domain servers in the form of a public key certificate. That is, each domain server shall have access to the authenticated credentials of other domain servers. However, there is no interaction between domain servers during a protocol run.

The 4PAKE protocols specified in this document shall make use of currently available secure two-party key exchange protocols, including:

— 2PAKE protocols as specified in ISO/IEC 11770-4;

— 2AAKE protocols as specified in ISO/IEC 11770-3;

— 2SAKE protocols as specified in ISO/IEC 11770-3;

— 2NIKE protocols as specified in ISO/IEC 11770-3.

NOTE Further information about these protocols can be found in References [12], [15] (for 2PAKE) and [13] (for 2AAKE and 2SAKE).

In addition, Annex A lists the object identifiers which shall be used to identify the mechanisms specified in this document.

6 Mechanisms

6.1 General

This clause specifies three cross-domain password-based authenticated key exchange mechanisms. The key exchange mechanisms specified in 6.3 to 6.5 require that:

a) each domain client involved shares a password-based weak secret with its domain server;

b) each domain server involved shall possess an asymmetric key pair.

For the key exchange mechanisms specified in 6.3 and 6.4, each domain server involved possesses a private signature key and the corresponding public verification key of a digital signature system.

For the key exchange mechanism specified in 6.5, each domain server involved possesses a pair of private and public key agreement keys for a two-party non-interactive key exchange protocol.

All three cross-domain password-based authenticated key exchange mechanisms have the following initialization and key establishment processes.

Initialization process

a) The two involved domain servers acquire each other's public key and agree to provide authentication tokens to assist their respective domain clients in establishing a shared secret key.

b) Each involved domain client and its respective domain server agree to use a shared password, which is known only to them, and a two-party password-based authenticated key exchange (2PAKE) protocol to perform intra-domain client-server authentication and key exchange. Each domain server also makes its public key available to all its domain clients.

c) The two involved clients and their respective domain servers agree to use a two-party authenticated key exchange protocol, which is either symmetric or asymmetric key based, to perform authenticated key exchange between the two clients.

A domain server can deliver its pubic key to other entities using one of the public key transport mechanisms specified in ISO/IEC 11770-3. To prevent potential attacks, it is recommended to use public key transport mechanism 3 specified in ISO/IEC 11770-3:2015, which uses digital certificates issued by trusted certification authorities (i.e. a public key infrastructure) to bind together the identity and public key of a domain server.

Key establishment process

a) Initial information exchange: both the involved domain clients generate an ephemeral public key according to the agreed two-party symmetric-key or asymmetric-key authenticated key exchange (2SAKE or 2AAKE) protocol. The two clients then exchange their user identities and ephemeral public keys, and validate the received public key from the peer client according to the agreed 2SAKE or 2AAKE protocol. If any validation fails, output "invalid" and stop.

b) Intra-domain client-server authentication and key exchange: both the involved clients perform a two-party password-based authenticated key exchange (2PAKE) protocol with their respective domain server to establish a local secret session key between the client and its domain server. If the 2PAKE protocol fails, output "fail" and stop.

c) Authentication token acquirement: each involved client sends the information exchanged in step a) to its domain server in an authenticated form using the local secret session key derived from the 2PAKE protocol in step b), and acquires from its domain server an authentication token derived using the private key of the domain server, the client identities and ephemeral public keys exchanged by the two clients in step a).

d) Cross-domain two-party authenticated key exchange: the two domain clients involved execute the 2AAKE or 2SAKE protocol, using the ephemeral public keys and the corresponding ephemeral private keys generated in step a) and the authentication tokens obtained in step c), to establish a shared secret key. If the 2AAKE or 2SAKE protocol fails, output "fail" and stop.

NOTE The mechanisms presented in this document are based on the three four-party password-based authenticated key exchange (4PAKE) protocols given in Reference [14].

6.2 Sub-protocols and functions

6.2.1 General

The cross-domain password-based key exchange mechanisms specified in 6.3 to 6.5 incorporate several sub-protocols, including two-party password-based authenticated key exchange (2PAKE), two-party asymmetric-key authenticated key exchange (2AAKE), two-party symmetric-key authenticated key exchange (2SAKE), and two-party non-interactive key exchange (2NIKE).

6.2.2 Two-party password-based authenticated key exchange

All three mechanisms given in this document use a two-party password-based authenticated key exchange (2PAKE) protocol as a sub-protocol. The 2PAKE protocol is used by a client to perform mutual

authentication and key exchange with its domain server using a shared password. The 2PAKE protocol shall be secure against dictionary attacks and can be based on one of the password-authenticated key agreement mechanisms specified in ISO/IEC 11770-4.

All three mechanisms treat the agreed 2PAKE protocol as a generic function with the following input and output.

— Input:

  — $Param_P$: prior shared parameters for the 2PAKE protocol, which include the identities of the two entities involved;

  — a secret password shared between the two entities involved.

— Output: a secret session key.

NOTE In all the mechanisms presented in this document, since the 2PAKE protocol is used only within a local domain by a domain client and its respective domain server. The two domains involved can choose to employ different 2PAKE protocols.

6.2.3 Two-party asymmetric-key authenticated key exchange

The mechanisms in 6.3 and 6.4 use a two-party asymmetric-key authenticated key exchange (2AAKE) protocol as a sub-protocol. The 2AAKE protocol is used by the two clients to perform authenticated key exchange with the help from their respective domain servers, who are responsible for helping their respective domain clients generate the authentication tokens used in the 2AAKE protocol.

A 2AAKE protocol has the following input and output.

— Input:

  — $Param_A$: prior shared parameters for the 2AAKE protocol, which include the identities of the two entities involved;

  — authentication public/private key pairs of the two entities involved;

  — ephemeral public/private key pairs of the two entities involved.

— Output: a secret session key.

The 2AAKE protocol shall contain the following functions.

a) Authentication key generation function, aKG:

  — Input: $Param_A$: 2AAKE prior shared parameters.

  — Output:

    — *ask*: an authentication private key randomly chosen from the authentication private key domain;

    — *apk*: an authentication public key corresponding to *ask*.

b) Authentication key validation function, aKV:

  — Input:

    — $Param_A$: 2AAKE prior shared parameters;

    — *apk*: an authentication public key.

  — Output:

    — 0 (indicating *apk* is invalid);

— 1 (indicating *apk* is valid).

c) Ephemeral key generation function, eKG:

— Input: $Param_A$: 2AAKE prior shared parameters.

— Output:

— *esk*: an ephemeral private key randomly chosen from the ephemeral private key domain;

— *epk*: an ephemeral public key corresponding to *esk*.

d) Ephemeral public key validation function, eKV:

— Input:

— $Param_A$: 2AAKE prior shared parameters;

— *epk*: an ephemeral public key.

— Output:

— 0 (indicating *epk* is invalid);

— 1 (indicating *epk* is valid).

The implementation of the above functions shall follow the specification of the selected 2AAKE protocol.

NOTE For the 2AAKE protocol presented in 6.3, the authentication public key of a client is the public key of its domain server. Hence, the clients only need to generate their ephemeral key pairs. In the 2AAKE protocol presented in 6.4, the clients need to generate both an authentication public/private key pair and an ephemeral key pair.

6.2.4 Two-party symmetric-key authenticated key exchange

The mechanism in 6.5 uses a two-party symmetric-key authenticated key exchange (2SAKE) protocol as a sub-protocol. The 2SAKE protocol is used by the two clients to perform authenticated key exchange with the help from the two domain servers who are responsible for assisting their respective domain clients to generate a common symmetric authentication key used by the clients in the 2SAKE protocol.

A 2SAKE protocol has the following input and output.

— Input:

— $Param_S$: prior shared parameters for the 2SAKE protocol, which include the identities of the two entities involved;

— a common secret key shared between the two entities involved;

— ephemeral public/private key pairs of the two entities involved.

— Output: a secret session key.

A 2SAKE protocol contains the following functions.

a) Ephemeral key generation function, eKG:

— Input: $Param_S$: 2SAKE prior shared parameters, including the ephemeral key domain.

— Output: an ephemeral key, containing an ephemeral private key *esk* and/or an ephemeral public key *epk* (see NOTE below).

b) Ephemeral key validation function, eKV:

- Input:
  - $Param_S$: 2SAKE prior shared parameters;
  - an ephemeral key containing *esk* and/or *epk*.
- Output:
  - 0 (indicating ephemeral key is invalid);
  - 1 (indicating ephemeral key is valid).

The implementation of the above functions shall follow the specification of the selected 2SAKE protocol.

NOTE Different 2SAKE protocols can have different formats for the ephemeral key. It is possible that some 2SAKE protocols do not have the ephemeral private key (i.e. *esk = NULL*) or the ephemeral public key (i.e. *epk = NULL*).

6.2.5 Two-party non-interactive key exchange

The mechanism in 6.5 uses a two-party non-interactive key exchange (2NIKE) protocol as a sub-protocol. The 2NIKE protocol is used by two domain servers S_A and S_B to derive a shared symmetric key based on their public and private key pairs, which subsequently use the derived symmetric key to generate a symmetric authentication key to be used by two clients A and B in a 2SAKE protocol.

A two-party non-interactive key exchange (2NIKE) protocol between an entity X and a peer entity Y has the following input and output.

- Input:
  - $Param_N$: prior shared parameters for the 2NIKE protocol, which include the identities of the two entities involved;
  - entity X's private key agreement key h_X;
  - entity Y's public key agreement key p_Y.
- Output: a secret session key $K_{X,Y}$.

NOTE An example of a 2NIKE protocol suitable for the purpose is the key agreement mechanism 1 specified in ISO/IEC 11770-3.

6.2.6 Session identity function

The schemes specified in 6.3 to 6.5 make use of the following function SIF, also defined in ISO/IEC 11770-4, to compute the session identity *sid*.

The session identity generation function, SIF, takes as input two distinguishing identifiers, represented as octet strings A and B, two group elements X and Y, and optionally two texts represented as octet strings *text*1 and *text*2, and produces an integer written *sid* = SIF(A, B, X, Y, *text*1, *text*2) as output. The function SIF is defined as follows:

- sn_A = BS2I($H(A||\text{GE2OS}_x(X)||text1)$);
- sn_B = BS2I($H(B||\text{GE2OS}_x(Y)||text2)$);
- SIF(A, B, X, Y, *text*1, *text*2) = MAX(sn_A, sn_B)||MIN(sn_A, sn_B).

The conversion functions BS2I and GE2OS_x specified in Annex B shall be used in the session identity function.

6.3 Mechanism 1

6.3.1 General

This cross-domain password-based key exchange mechanism allows two clients A and B associated with two different domain servers S_A and S_B, respectively, to perform mutual authentication and key exchange. This mechanism is built on two sub-protocols:

— a two-party password-based authenticated key exchange (2PAKE) protocol.

— a signature-based two-party asymmetric-key authenticated key exchange (2AAKE) protocol.

It requires both domain servers to possess a public and private key pair for a digital signature system.

In this mechanism, a domain server's public and private key pair serves as the authentication key pair of its client in the signature-based 2AAKE protocol.

6.3.2 Prior shared parameters

The key exchange between clients A and B with domain servers S_A and S_B, respectively, takes place in an environment where the involved entities share the following parameters:

— a password-based octet string $pwd_{SA,A}$ shared between A and S_A.

— a password-based octet string $pwd_{SB,B}$ shared between B and S_B.

— the specification of a two-party password-based authenticated key exchange protocol, 2PAKE.

— the specification of a signature-based two-party asymmetric-key authenticated key exchange protocol, 2AAKE.

— the specification of an asymmetric digital signature system, Σ.

— the specification of a message authentication code function, MAC.

— the specification of a session identifier generation function, SIF.

— the domain server S_A's public verification key VK_{SA} for Σ.

— the domain server S_B's public verification key VK_{SB} for Σ.

6.3.3 Key exchange operation

6.3.3.1 General

This mechanism involves clients A and B in two different domains with respective domain servers S_A and S_B. The mechanism has five steps, numbered $M1A$ – $M5A$ and $M1B$ – $M5B$ (for steps performed by A and B, respectively), and is illustrated in Figure 1.

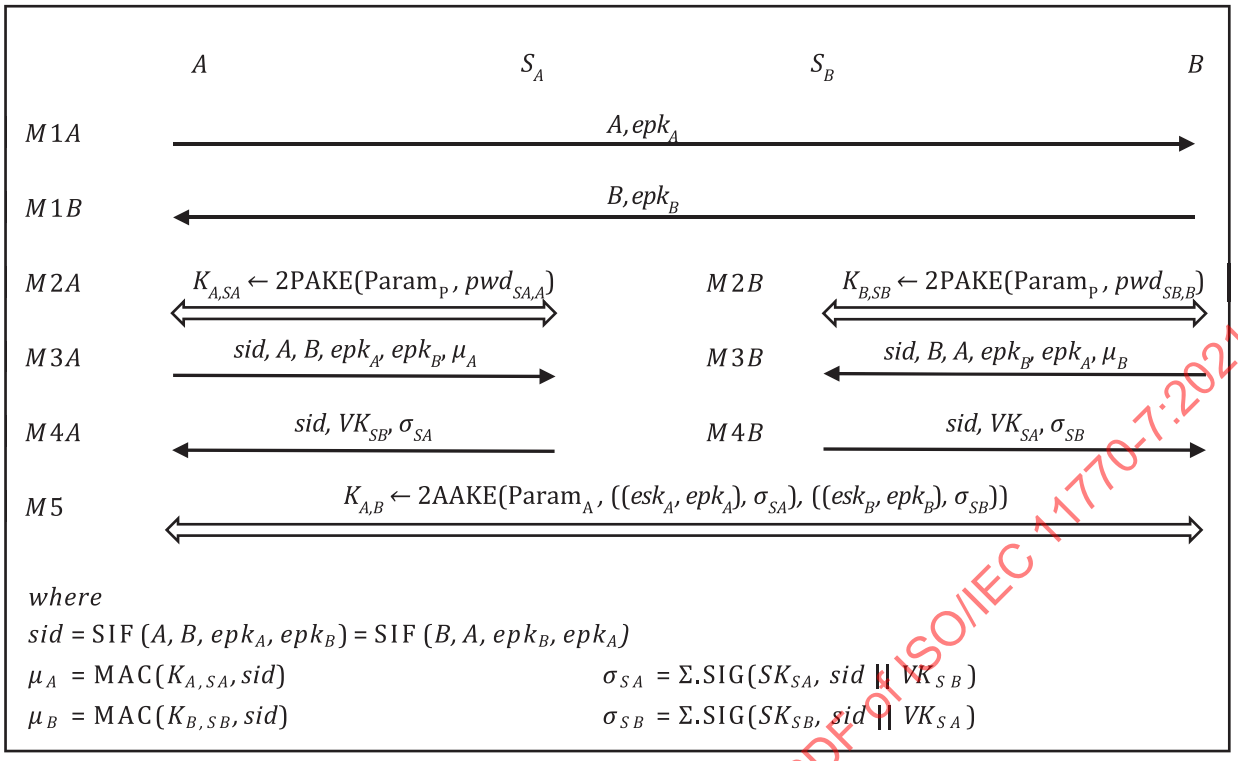

Figure 1 — Mechanism 1

NOTE 1 In Mechanism 1, a client uses the authentication token produced by its domain server to replace the authentication public and private key pair in the input to the 2AAKE protocol, i.e. the client implicitly uses its domain server's public and private key pair as its authentication key pair.

NOTE 2 It is assumed that the server S_A (or S_B) trusts that its client A (or B) is honest and knows the private counterpart of the ephemeral public key epk_A (or epk_B).

Below are the detailed operations of each step.

6.3.3.2 Ephemeral public key construction (*M1A*)

— A invokes eKG of the signature-based 2AAKE protocol to generate an ephemeral public key epk_A and its associated ephemeral private key esk_A.

— A makes epk_A available to B.

6.3.3.3 Ephemeral public key construction (*M1B*)

— B invokes eKG of the signature-based 2AAKE protocol to generate an ephemeral public key epk_B and its associated ephemeral private key esk_B.

— B makes epk_B available to A.

6.3.3.4 Local password key exchange (*M2A*)

— A receives epk_B.

— A invokes eKV of the 2AAKE protocol to validate the epk_B; if eKV outputs 0, output “invalid” and stop. Otherwise, carry on.

— A runs the 2PAKE protocol with S_A to establish a local secret key $K_{A,SA}$; if 2PAKE fails, stop.

6.3.3.5 Local password key exchange (*M2B*)

— *B* receives epk_A.

— *B* invokes eKV of the 2AAKE protocol to validate the epk_A; if eKV outputs 0, output "invalid" and stop. Otherwise, carry on.

— *B* runs the 2PAKE protocol with S_B to establish a local secret key $K_{B, SB}$; if 2PAKE fails, stop.

6.3.3.6 Local client authentication (*M3A*)

— *A* computes the session id *sid* = SIF(*A, B,* epk_A, epk_B).

— *A* computes μ_A = MAC($K_{A, SA}$, *sid*).

— *A* makes (*sid, A, B,* epk_A, epk_B, μ_A) available to S_A.

6.3.3.7 Local client authentication (*M3B*)

— *B* computes the session id *sid* = SIF(*B, A,* epk_B, epk_A).

— *B* computes μ_B = MAC($K_{B, SB}$, *sid*).

— *B* makes (*sid, B, A,* epk_B, epk_A, μ_B) available to S_B.

6.3.3.8 Authentication token acquirement (*M4A*)

— S_A receives (*sid, A, B,* epk_A, epk_B, μ_A).

— S_A computes the session id *sid'* = SIF(*A, B,* epk_A, epk_B); if *sid'* ≠ *sid*, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A computes μ_{SA} = MAC($K_{A, SA}$, *sid*).

— S_A compares μ_{SA} against μ_A; if $\mu_{SA} \neq \mu_A$, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A computes the authentication token σ_{SA} = Σ.SIG(SK_{SA}, *sid*||VK_{SB}).

— S_A makes (*sid,* VK_{SB}, σ_{SA}) available to *A*.

6.3.3.9 Authentication token acquirement (*M4B*)

— S_B receives (*sid, B, A,* epk_B, epk_A, μ_B).

— S_B computes the session id *sid'* = SIF(*B, A,* epk_B, epk_A); if *sid'* ≠ *sid*, S_B returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B computes μ_{SB} = MAC($K_{B, SB}$, *sid*).

— S_B compares μ_{SB} against μ_B; if $\mu_{SB} \neq \mu_B$, S_B returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B computes the authentication token σ_{SB} = Σ.SIG(SK_{SB}, *sid*||VK_{SA}).

— S_B makes (*sid,* VK_{SA}, σ_{SB}) available to *B*.

6.3.3.10 Cross-domain key exchange (*M5A*)

— *A* receives (*sid,* VK_{SB}, σ_{SA}).

— *A* runs Σ.VER(VK_{SA}, *sid*||VK_{SB}, σ_{SA}) to validate σ_{SA}; if the output is 0, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *A* executes the signature-based 2AAKE with *B* based on the ephemeral key pair (epk_A, esk_A) and authentication token σ_{SA}; if the 2AAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2AAKE as the shared secret key $K_{A,B}$.

6.3.3.11 Cross-domain key exchange (*M5B*)

— *B* receives (*sid*, VK_{SA}, σ_{SB}).

— *B* runs Σ.VER(VK_{SB}, *sid*||VK_{SA}, σ_{SB}) to validate σ_{SB}; if the output is 0, B returns "invalid" and stops the protocol. Otherwise, carry on.

— *B* executes the signature-based 2AAKE with *A* based on the ephemeral key pair (epk_B, esk_B) and authentication token σ_{SB}; if the 2AAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2AAKE as the shared secret key $K_{A,B}$.

NOTE Examples of signature-based 2AAKE protocols suitable for the purpose are the SIG-DH protocol in Reference [13] and key agreement mechanism 7 in ISO/IEC 11770-3. Use of key agreement mechanism 7 in ISO/IEC 11770-3 has practical advantages, since the mechanism description and the terms and definitions specified in ISO/IEC 11770-3 are compatible with those used in this document.

6.4 Mechanism 2

6.4.1 General

This mechanism allows two clients to use a single pair of authentication tokens received from their respective domain servers to establish multiple session keys within a certain time period. This mechanism is built upon two sub-protocols:

— a two-party password-based authenticated key exchange (2PAKE) protocol;

— a two-party asymmetric-key authenticated key exchange (2AAKE) protocol.

In this mechanism, a domain server serves as the CA for its clients, and the signature-based authentication token provided by a domain server to its client serves as a digital certificate for the 2AAKE authentication public key generated by the client.

6.4.2 Prior shared parameters

The key exchange between clients *A* and *B* with domain servers S_A and S_B, respectively, takes place in an environment where the involved entities share the following parameters:

— a password-based octet string $pwd_{SA,A}$ shared between *A* and S_A;

— a password-based octet string $pwd_{SB,B}$ shared between *B* and S_B;

— the specification of a two-party password-based authenticated key exchange protocol, 2PAKE;

— the specification of a two-party asymmetric-key authenticated key exchange protocol, 2AAKE;

— the specification of an asymmetric digital signature system, *Σ*;

— the specification of a message authentication code function, MAC;

— the specification of a session identifier generation function, SIF;

— the domain server S_A's public verification key VK_{SA} for *Σ*;

— the domain server S_B's public verification key VK_{SB} for *Σ*.

6.4.3 Key exchange operation

6.4.3.1 General

This mechanism involves clients A and B in two different domains with respective domain servers S_A and S_B. The mechanism has six steps, numbered $M1A$ – $M6A$ and $M1B$ – $M6B$ (for steps performed by A and B, respectively), and is illustrated in Figure 2.

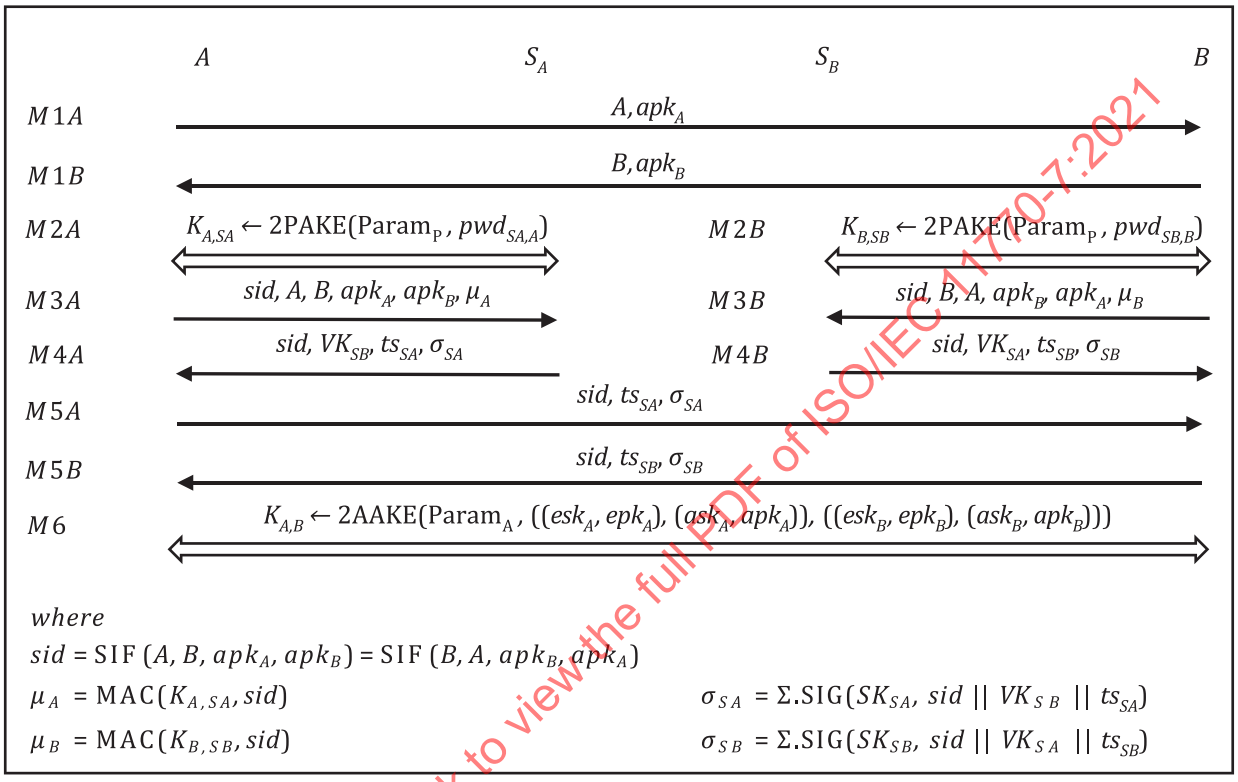

Figure 2 — Mechanism 2

NOTE It is assumed that the server S_A (or S_B) trusts that its client A (or B) is honest and knows the private counterpart of the authentication key apk_A (or apk_B). Otherwise, the client needs to provide a proof of possession of their key.

6.4.3.2 Client authentication key generation (*M1A*)

— A invokes aKG of the 2AAKE protocol to generate an authentication key pair (apk_A, ask_A).

— A makes apk_A available to B.

6.4.3.3 Client authentication key generation (*M1B*)

— B invokes aKG of the 2AAKE protocol to generate an authentication key pair (apk_B, ask_B).

— B makes apk_B available to A.

6.4.3.4 Local password-based key exchange (*M2A*)

— A receives apk_B.

— A invokes aKV of the 2AAKE protocol to validate the apk_B; if aKV outputs 0, output “invalid” and stop. Otherwise, carry on.

— A runs the 2PAKE protocol with S_A to establish a local secret key $K_{A,SA}$; if 2PAKE fails, stop.

6.4.3.5 Local password-based key exchange (*M2B*)

— B receives apk_A.

— B invokes aKV of the 2AAKE protocol to validate the apk_A; if aKV outputs 0, output "invalid" and stop. Otherwise, carry on.

— B runs the 2PAKE protocol with S_B to establish a local secret key $K_{B, SB}$; if 2PAKE fails, stop.

6.4.3.6 Local client authentication (*M3A*)

— A computes the session id sid = SIF(A, B, apk_A, apk_B).

— A computes μ_A = MAC($K_{A, SA}$, sid).

— A makes (sid, A, B, apk_A, apk_B, μ_A) available to S_A.

6.4.3.7 Local client authentication (*M3B*)

— B computes the session id sid = SIF(B, A, apk_B, apk_A).

— B computes μ_B = MAC($K_{B, SB}$, sid).

— B makes (sid, B, A, apk_B, apk_A, μ_B) available to S_B.

6.4.3.8 Client certificate acquirement (*M4A*)

— S_A receives (sid, A, B, apk_A, apk_B, μ_A).

— S_A computes the session id sid' = SIF(A, B, apk_A, apk_B); if $sid' \neq sid$, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A computes μ_{SA} = MAC($K_{A, SA}$, sid).

— S_A compares μ_{SA} against μ_A; if $\mu_{SA} \neq \mu_A$, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A generates a timestamp ts_{SA}.

— S_A computes the authentication token σ_{SA} = Σ.SIG(SK_{SA}, $sid||VK_{SB}||ts_{SA}$).

— S_A makes (sid, VK_{SB}, ts_{SA}, σ_{SA}) available to A.

6.4.3.9 Client certificate acquirement (*M4B*)

— S_B receives (sid, B, A, apk_B, apk_A, μ_B).

— S_B computes the session id sid' = SIF(B, A, apk_B, apk_A); if $sid' \neq sid$, S_B returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B computes μ_{SB} = MAC($K_{B, SB}$, sid).

— S_B compares μ_{SB} against μ_B; if $\mu_{SB} \neq \mu_B$, S_B returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B generates a timestamp ts_{SB}.

— S_B computes the authentication token σ_{SB} = Σ.SIG(SK_{SB}, $sid||VK_{SA}||ts_{SB}$).

— S_B makes (sid, VK_{SA}, ts_{SB}, σ_{SB}) available to B.

6.4.3.10 Client certificate validation (*M5A*)

— *A* receives (*sid*, VK_{SB}, ts_{SA}, σ_{SA}).

— *A* runs Σ.VER(VK_{SA}, *sid*||VK_{SB}||ts_{SA}, σ_{SA}) to89 validate σ_{SA}; if the output is 0, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *A* makes (*sid*, ts_{SA}, σ_{SA}) available to *B*.

6.4.3.11 Client certificate validation (*M5B*)

— *B* receives (*sid*, VK_{SA}, ts_{SB}, σ_{SB}).

— *B* runs Σ.VER(VK_{SB}, *sid*||VK_{SA}||ts_{SB}, σ_{SB}) to validate σ_{SB}; if the output is 0, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *B* makes (*sid*, ts_{SB}, σ_{SB}) available to *A*.

6.4.3.12 Cross-domain key exchange (*M6A*)

— *A* receives (*sid*, ts_{SB}, σ_{SB}).

— *A* verifies the current time is within the time period specified by ts_{SB}; if the verification fails, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *A* runs Σ.VER(VK_{SB}, *sid*||VK_{SA}||ts_{SB}, σ_{SB}) to validate σ_{SB}; if the output is 0, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *A* executes 2AAKE with *B* based on a freshly generated ephemeral key pair (epk_A, esk_A) and its authentication key pair (apk_A, ask_A); if the 2AAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2AAKE as the shared secret key $K_{A,B}$.

6.4.3.13 Cross-domain key exchange (*M6B*)

— *B* receives (*sid*, ts_{SA}, σ_{SA}).

— *B* verifies the current time is within the time period specified by ts_{SA}; if the verification fails, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *B* runs Σ.VER(VK_{SA}, *sid*||VK_{SB}||ts_{SA}, σ_{SA}) to validate σ_{SA}; if the output is 0, *A* returns "invalid" and stops the protocol. Otherwise, carry on.

— *B* executes 2AAKE with *A* based on a freshly generated ephemeral key pair (epk_B, esk_B) and its authentication key pair (apk_B, ask_B); if the 2AAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2AAKE as the shared secret key $K_{A,B}$.

NOTE The 2AAKE protocol used in this mechanism can be based on various asymmetric authentication mechanisms. A variety of 2AAKE protocols are specified in ISO/IEC 11770-3.

6.5 Mechanism 3

6.5.1 General

This mechanism is built on three sub-protocols:

— a two-party non-interactive key exchange protocol (2NIKE);

— a two-party password-based authenticated key exchange (2PAKE) protocol;

— a two-party symmetric-key authenticated key exchange (2SAKE) protocol.

It requires each domain server involved to possess a public and private key pair for a two-party non-interactive key exchange protocol.

In this mechanism, the two domain servers involved first establish a shared symmetric key using the 2NIKE protocol, and then use the established symmetric key to generate a symmetric authentication key to be used by the two clients in the 2SAKE protocol.

6.5.2 Prior shared parameters

The key exchange between clients A and B with domain servers S_A and S_B, respectively, takes place in an environment where the involved entities share the following parameters:

— a password-based octet string $pwd_{SA,A}$ shared between A and S_A;

— a password-based octet string $pwd_{SB,B}$ shared between B and S_B;

— the specification of a two-party password-based authenticated key exchange protocol, 2PAKE;

— the specification of a two-party symmetric-key authenticated key exchange protocol, 2SAKE;

— the specification of a two-party non-interactive key exchange protocol, 2NIKE;

— the specification of a message authentication code function, MAC;

— the specification of a key derivation function, KD;

— the specification of a session identifier generation function, SIF;

— the domain server S_A's public key p_{SA} for the 2NIKE;

— the domain server S_B's public key p_{SB} for the 2NIKE;

— a secret key shared by S_A and S_B:

$K_{SA,\,SB}$ = 2NIKE(Param$_N$, h_{SB}, p_{SA}) = 2NIKE(Param$_N$, h_{SA}, p_{SB}).

6.5.3 Key exchange operation

6.5.3.1 General

This mechanism involves clients A and B in two different domains with respective domain servers S_A and S_B. This mechanism has five steps, numbered $M1A$ – $M5A$ and $M1B$ – $M5B$ (for steps performed by A and B, respectively), and is illustrated in Figure 3.

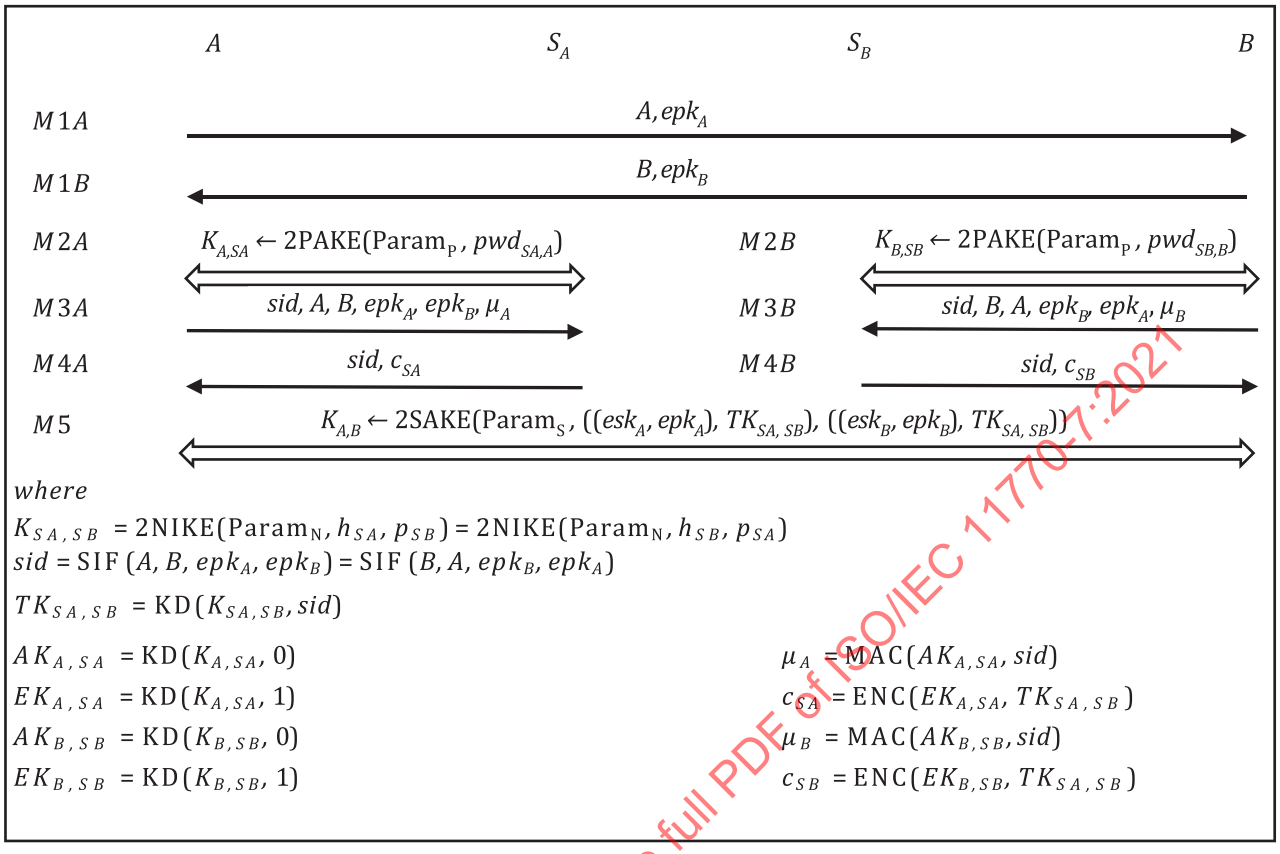

Figure 3 — Mechanism 3

NOTE It is assumed that the server S_A (or S_B) trusts that its client A (or B) is honest and knows their private counterpart of the ephemeral key epk_A (or epk_B). Otherwise, the client needs to provide a proof of possession of their key.

Below are the detailed operations of each step.

6.5.3.2 Ephemeral public key construction (*M1A*)

— A invokes eKG of the 2SAKE protocol to generate an ephemeral public key epk_A and its associated ephemeral private key esk_A.

— A makes epk_A available to B.

6.5.3.3 Ephemeral public key construction (*M1B*)

— B invokes eKG of the 2SAKE protocol to generate an ephemeral public key epk_B and its associated ephemeral private key esk_B.

— B makes epk_B available to A.

6.5.3.4 Local password key exchange (*M2A*)

— A receives epk_B.

— A invokes eKV of the 2SAKE protocol to validate the epk_B; if eKV outputs 0, output "invalid" and stop. Otherwise, carry on.

— A runs the 2PAKE protocol with S_A to establish a local secret key $K_{A,SA}$; if 2PAKE fails, stop.

— A invokes the key derivation function KD with secret value $K_{A, SA}$ and key derivation parameter "0" to derive a local symmetric authentication key $AK_{A, SA}$ = KD($K_{A, SA}$, 0).

— A invokes the key derivation function KD with secret value $K_{A, SA}$ and key derivation parameter "1" to derive a local symmetric encryption key $EK_{A, SA}$ = KD($K_{A, SA}$, 1).

6.5.3.5 Local password key exchange (*M2B*)

— B receives epk_A.

— B invokes eKV of the 2SAKE protocol to validate the epk_A; if eKV outputs 0, output "invalid" and stop. Otherwise, carry on.

— B runs the 2PAKE protocol with S_B to establish a local secret key $K_{B, SB}$; if 2PAKE fails, stop.

— B invokes the key derivation function KD with secret value $K_{B, SB}$ and key derivation parameter "0" to derive a local symmetric authentication key $AK_{B, SB}$ = KD($K_{B, SB}$, 0).

— B invokes the key derivation function KD with secret value $K_{B, SB}$ and key derivation parameter "1" to derive a local symmetric encryption key $EK_{B, SB}$ = KD($K_{B, SB}$, 1).

6.5.3.6 Local client authentication (*M3A*)

— A computes the session id sid = SIF(A, B, epk_A, epk_B).

— A computes μ_A = MAC($AK_{A, SA}$, sid).

— A makes (sid, A, B, epk_A, epk_B, μ_A) available to S_A.

6.5.3.7 Local client authentication (*M3B*)

— B computes the session id sid = SIF(B, A, epk_B, epk_A).

— B computes μ_B = MAC($AK_{B, SB}$, sid).

— B makes (sid, B, A, epk_B, epk_A, μ_B) available to S_B.

6.5.3.8 Client key acquirement (*M4A*)

— S_A receives (sid, A, B, epk_A, epk_B, μ_A) from A.

— S_A computes the session id sid' = SIF(A, B, epk_A, epk_B); if $sid' \neq sid$, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A invokes the key derivation function KD with secret value $K_{A, SA}$ and key derivation parameter "0" to derive a local symmetric authentication key $AK_{A, SA}$ = KD($K_{A, SA}$, 0).

— S_A invokes the key derivation function KD with secret value $K_{A, SA}$ and key derivation parameter "1" to derive a local symmetric encryption key $EK_{A, SA}$ = KD($K_{A, SA}$, 1).

— S_A computes μ_{SA} = MAC($AK_{A, SA}$, sid).

— S_A compares μ_{SA} against μ_A; if $\mu_{SA} \neq \mu_A$, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_A invokes the key derivation function KD with secret value $K_{SA, SB}$ and key derivation parameter sid to derive a 2SAKE authentication key $TK_{SA, SB}$ = KD($K_{SA, SB}$, sid).

— S_A computes c_{SA} = ENC($EK_{A, SA}$, $TK_{SA, SB}$) and makes (sid, c_{SA}) available to A.

6.5.3.9 Client key acquirement (*M4B*)

— S_B receives (*sid, B, A,* epk_B, epk_A, μ_B) from *B*.

— S_B computes the session id *sid'* = SIF(*B, A,* epk_B, epk_A); if *sid'* ≠ *sid*, S_A returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B invokes the key derivation function KD with secret value $K_{B, SB}$ and key derivation parameter "0" to derive a local symmetric authentication key $AK_{B, SB}$ = KD($K_{B, SB}$, 0).

— S_B invokes the key derivation function KD with secret value $K_{B, SB}$ and key derivation parameter "1" to derive a local symmetric encryption key $EK_{B, SB}$ = KD($K_{B, SB}$, 1).

— S_B computes μ_{SB} = MAC($AK_{B, SB}$, *sid*).

— S_B compares μ_{SB} against μ_B; if $\mu_{SB} \neq \mu_B$, S_B returns "invalid" and stops the protocol. Otherwise, carry on.

— S_B invokes the key derivation function KD with secret value $K_{SA, SB}$ and key derivation parameter *sid* to derive a 2SAKE authentication key $TK_{SA, SB}$ = KD($K_{SA, SB}$, *sid*).

— S_B computes c_{SB} = ENC($EK_{B, SB}$, $TK_{SA, SB}$) and makes (*sid,* c_{SB}) available to *B*.

6.5.3.10 Cross-domain key exchange (*M5A*)

— *A* receives (*sid,* c_{SA}) from S_A.

— *A* runs DEC($EK_{A, SA}$, c_{SA}); if the output is "⊥", *A* returns "invalid" and stops the protocol. Otherwise, *A* obtains $TK_{SA, SB}$ = DEC($EK_{A, SA}$, c_{SA}) and executes 2SAKE with *B* based on the ephemeral key pair (epk_A, esk_A) and authentication key $TK_{SA, SB}$; if the 2SAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2SAKE as the session key $K_{A, B}$.

6.5.3.11 Cross-domain key exchange (*M5B*)

— *B* receives (*sid,* c_{SB}) from S_B.

— *B* runs DEC($EK_{B, SB}$, c_{SB}); if the output is "⊥", *B* returns "invalid" and stops the protocol. Otherwise, *B* obtains $TK_{SA, SB}$ = DEC($EK_{B, SB}$, c_{SB}) and executes 2SAKE with *A* based on the ephemeral key pair (epk_B, esk_B) and authentication key $TK_{SA, SB}$; if the 2SAKE protocol fails, output "fail" and stop. Otherwise, return the output from 2SAKE as the session key $K_{A, B}$.

NOTE A variety of 2SAKE protocols are specified in ISO/IEC 11770-2.