SUMMARY OF ANSI X9.42
Agreement of Symmetric Keys Using Discrete Logarithm Cryptography

1. Scope

This standard specifies schemes for the agreement of symmetric keys using the Diffie-Hellman and MQV algorithms. These methods may be used by different parties to establish common shared secret information such as cryptographic keys. The shared secret information may be used with symmetrically-keyed algorithms to provide confidentiality, authentication, and data integrity services, or used as a key-encrypting key with other key management protocols. The key agreement schemes specified in ANSI X9.42 may be used as subroutines to build key establishment protocols.

2. Contents

The asymmetric key establishment schemes in ANSI X9.42 are used by an entity *U* (the initiator) who wishes to establish a symmetric key with another entity *V* (the responder). Each entity has at least one key pair. If *U* and *V* simultaneously execute a scheme with corresponding keying material as input, then at the end of the execution of the scheme, *U* and *V* will share keying data. The keying data can then be used to supply cryptographic material (e.g., keys and IVs) for symmetric algorithms.

ANSI X9.42 specifies eight asymmetric key establishment schemes. Six schemes using Diffie-Hellman and two schemes using MQV are provided in the document. A variety of schemes are specified in order to provide a wide variety of services for various environments.

The standard also contains specifications for:

- Domain parameter generation and validation
- Key pair generation and public key validation
- A Diffie-Hellman algorithm
- Two MQV algorithms
- Two forms of a Key Derivation function
- A MAC calculation and an ANSI X9.42 validation

In addition, for each key agreement scheme specified in the standard, a security assessment is provided.

3. Basic Algorithms and Functions

3.1 Keying Material

The keying material needed to perform key agreement as specified by ANSI X9.42 includes a set of domain parameters and one or more key pairs.

The domain parameters include:

- p A prime defining the Galois Field *GF(p)*, which is used as a modulus in the operations of *GF(p)*, where $2^{(L-1)} < p < 2^L$, for $L \geq 1024$, and L is a multiple of 256.
- q A prime factor of p-1 such that $p = jq+1$ and $q > 2^{m-1}$. *GF(p)** has a cyclic subgroup of order q.
- g A generator of the q-order cyclic subgroup of *GF(p)**, that is, an element of order q in the multiplicative group of *GF(p)*.

The domain parameters may be distributed, for example, in a public key certificate. Two classes of domain parameters may exist: static domain parameters or ephemeral domain parameters. Static domain parameters may be used with either static or ephemeral keys. Static domain parameters are designated with a subscript of *s* (i.e., p_s, q_s, g_s). Ephemeral domain parameters are used with ephemeral keys and are designated with a subscript of *e* (i.e., p_e, q_e, g_e).

A key pair consists of a private key and a public key. Two classes of key pairs may be used in a scheme:

- static key pairs (x_s, y_s), where x_s is the private key and y_s is the public key, and
- ephemeral key pairs (r_e, t_e), where r_e is the private key and t_e is the public key.

Static key pairs are longer-lived; the public key from a static key pair (y_s) may be included in a public key certificate. Ephemeral keys are shorter lived (e.g., for the duration of a message or a communication session).

3.2 Domain parameter Generation and Validation

ANSI X9.42 provides primitives for both domain parameter generation and validation. Domain parameter validation may be used to verify that the domain parameters satisfy certain basic criteria (e.g., whether g is a generator of order $2 \leq g \leq (p-2)$ and that $g^q = 1 \mod p$). Domain parameter validation could be performed by a Certificate Authority (CA) or some other trusted party.

3.3 Key Pair Generation and Public Key Validation

ANSI X9.42 provides primitives for key pair generation and public key validation. Both static keys and ephemeral keys may be generated using the same primitives.

Methods for implicit and explicit public key validation are provided. Implicit public key validation requires that the validator know the private key and can, therefore, compute the public key. Alternatively, implicit key authentication is provided if p and q are related in a defined way. For explicit public key validation , an entity verifies that a public key appears to satisfy certain basic criteria (e.g., by verifying that $2 \leq y \leq p\text{-}2$ and that $y^q = 1 \bmod p$). A static public key could be validated, for example, by a CA, a trusted party or a responder in a key establishment protocol. An ephemeral public key might be validated by the responder in a key establishment protocol.

3.4 Diffie-Hellman Algorithm

The Diffie-Hellman algorithm is used to compute a shared secret value for six of the schemes in the standard. In Section 5.1, the schemes are defined for an initiator (U) and a responder (V). However, both parties perform similar computations. The description in this section is expressed for parties A and B, where party A is the local party, and party B is the other party (i.e., when the initiator (U) is performing the calculations, A=U and B=V; when the responder (V) is performing the calculations, A=V and B=U).

Input:

- a set of domain parameters (p, q, g)
- a private key x_A — Note: this could be r_A if an ephemeral private key is used in the calculation
- a public key y_B — Note: this could be t_B if an ephemeral private key is used in the calculation

Compute:

$$Z = y_B^{X_A} \bmod p$$

Output: Z, the shared value

3.5 MQV Algorithms

The MQV algorithm is used to derive a shared secret value. There are two forms of the algorithm: an interactive form, and a store-and-forward form. The two schemes are again defined for an initiator (U) and a responder (V).

3.5.1 Interactive Form

Interactive communications are required when using this form of the MQV algorithm. The same calculations are performed by both parties in a communication. Therefore, the concept of party A as the local party and party B as the other party is used here (i.e., when the initiator (U) is performing the calculations, A=U and B=V; when the responder (V) is performing the calculations, A=V and B=U).

Input:

- a set of domain parameters (p, q, g)
- the local party's static private key – x_A
- the local party's ephemeral key pair – (r_A, t_A)
- the other party's static and ephemeral public keys – y_B and t_B

Compute:

$w = \lceil ||q||/2 \rceil$

$t_A' = t_A (\text{mod } 2^w) + 2^w$

$S_A = (r_A + t_A' x_A) \text{ mod } q$

$t_B' = t_B (\text{mod } 2^w) + 2^w$

$Z = ((t_B (y_B^{t_B'}))^{S_A}) \text{ mod } p$

Output: Z, the shared value.

3.5.2 Store-and-Forward Form

This form of the MQV algorithm does not require interactive communications. The initiator (U) and responder (V) in a key agreement process perform slightly different calculations.

3.5.2.1 The Initiator's Process

Input:

- a set of domain parameters (p, q, g)
- the initiator's static private key – x_U
- the initiator's ephemeral key pair – (r_U, t_U)
- the responder's static public key – y_V

Compute:

$w = \lceil ||q||/2 \rceil$

$t_U' = t_U (\text{mod } 2^w) + 2^w$

$S_U = (r_U + t_U' x_U) \text{ mod } q$

$y_V' = y_V (\text{mod } 2^w) + 2^w$

$Z = (y_V (y_V^{y_V'}))^{S_U}) \text{ mod } p$

Output: *Z*, the shared value

3.5.2.2 The Responder's Process

Input:

- a set of domain parameters (*p, q, g*)
- the responder's static key pair – (x_V, y_V)
- the initiator's static and ephemeral public keys – y_U and t_U

Compute:

$w = \lceil ||q||/2 \rceil$

$y_V' = y_V \pmod{2^w} + 2^w$

$S_V = (x_V + y_V' x_V) \bmod q$

$t_U' = t_U \pmod{2^w} + 2^w$

$Z = (t_U (y_U^{t_U'}))^{S_V} \bmod p$

Output: *Z*, the shared value

3.6 Key Derivation Functions

In the ANSI X9.42 key agreement schemes, keying data is derived from a shared secret value. Two key derivation functions are defined: one based on ASN.1 DER encoding, and the other based on the concatenation of fixed length fields. Both of the key derivation functions are constructed from an ANSI-approved cryptographic hash function (e.g., SHA-1). Multiple keys may be produced from either 1) a single invocation of the function, or 2) multiple invocations of the function using the same shared value for each invocation, but different supplementary information.

3.6.1 Key Derivation Function Based on ASN.1

Input:

- A shared secret value *ZZ*
- The length of the keying data to be determined – *keylen*
- Other information – *OtherInfo*, containing an algorithm ID, counter, optional information for each party, and optional supplementary public and private information

Compute:

$d = \lceil keylen/hashlen \rceil$

Note: *hashlen* is the length of the output of the chosen hash function

counter = 1

For i = 1 to d
$Hash_i = H(ZZ||OtherInfo)$
$counter = counter + 1$
Increment i
$KeyingData$ = leftmost $keylen$ bits of $Hash_1||Hash_2||\ldots||Hash_d$

Output: *KeyingData*

3.6.2 Key Derivation Function Based on Concatenation

Input:

- A shared secret value *ZZ*
- The length of the keying data to be determined – *keylen*
- Other information – *OtherInfo*, data shared by the two parties

Compute:

$d = \lceil keylen/hashlen \rceil$

Note: *hashlen* is the length of the output of the chosen hash function

$counter$ = 1
For i = 1 to d
$Hash_i = H(ZZ||counter||OtherInfo)$
$counter = counter + 1$
Increment i
$KeyingData$ = leftmost $keylen$ bits of $Hash_1||Hash_2||\ldots||Hash_d$

Output: *KeyingData*

3.7 MAC Computation and ANSI X9.42 Validation

A MAC may optionally be computed using a key derived from a shared secret value and one of the key derivation methods defined in Section 3.6 using an ANSI-approved MAC function (e.g., HMAC).

Input:

- A MAC key – *MacKey*
- The data to be MACed – *MacData*

Compute:

$MacValue = MAC(MacKey, MacData)$

Output: *MacValue*

The MAC may be used for implementation validation with *MacData* specified as the string “ANSI X9.42 Testing Message” plus a nonce, where *MacData* is encoded in ASN.1.

4. Security Attributes

The key agreement schemes defined in ANSI X9.42 offer one or more of the follow security attributes:

- Implicit Key Authentication (IKA) – establishes the identity of the other party; provided by a static key that is bound to the other party’s identity.
- Explicit Key Authentication (EKA) – knowledge that the other party did indeed calculate the shared key; may be provided by doing key confirmation in a scheme that provides implicit key authentication.
- Forward Secrecy (FS) – the assurance provided to an entity that the session key established with another entity will not be compromised by the compromise of either entity's static private key in the future; provided by the use of an ephemeral public key.
- Entity Authentication (EA) – consists of identification and “liveness”, whereas EKA is concerned with identification and key possession. If explicit confirmation is deemed adequate to demonstrate liveness, then it is possible for schemes that achieve EKA to also support EA. Whether or not the two are synonymous depends upon the interpretations of “entity” and “liveness”, and on the timespan for the protocol that realizes the key agreement scheme. Therefore, the EA attribute does not follow automatically from EKA. ANSI X9.42 does not address entity authentication more specifically because EKA is neither always necessary nor always sufficient to achieve EA.
- Known-Key Security (K-KS) – assurance that a particular session key will not be compromised as a result of the compromise of other keys; provided by using an ephemeral key to compute the key(s) for only one session.
- Unknown Key-share Resilience (U-KS) – assurance provided to one party (A) that if party A and party B share a session key, then party B does not mistakenly believe the session key is shared with an entity other than party A.
- Key-Compromise Impersonation Resilience (K-CI) – assurance provided to a party (A) that, even if an adversary somehow obtains party A’s static private key, the adversary cannot successfully impersonate another party to party A.

5. Key Agreement Schemes

Multiple key agreement schemes using Diffie-Hellman and MQV techniques are specified in ANSI X9.42. The key agreement schemes and their security attributes discussed in this section. Let party U be the initiator and party V be the responder in each key agreement scheme.

5.1 Key Agreement using Diffie-Hellman

5.1.1 dhStatic

Both parties have only static keys. Using the same set of static-key domain parameters (p_s, q_s, g_s), Party U and Party V generate individual static private/public key pairs, denoted (x_U, y_U) and (x_V, y_V), respectively.

This scheme corresponds to the Static Unified Model scheme in ANSI X9.63. The scheme provides Implicit Key Authentication to both parties, and Unknown-Keyshare Resilience to both parties if knowledge of the private key (also known as proof of possession) was checked during the certification of the static public keys. Any type of communication may be used.

	Party U	**Party V**
Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	N/A	N/A
Input	(p_s, q_s, g_s), x_U, y_V	(p_s, q_s, g_s), x_V, y_U
	$Z_s = y_V^{x_U} \bmod p_s$	$Z_s = y_U^{x_V} \bmod p_s$
Shared secret value	$ZZ = Z_s$	$ZZ = Z_s$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*

5.1.2 dhEphem

Both parties have only ephemeral keys. Using the same set of ephemeral-key domain parameters (p_e, q_e, g_e), Party U and Party V generate individual ephemeral private/public key pairs, denoted (r_U, t_U) and (r_V, t_V) respectively.

This scheme corresponds to the Ephemeral Unified Model scheme in ANSI X9.63. The scheme provides Known-Key Security and Forward Secrecy to both parties if explicit authentication is supplied for all session keys. Interactive communications are required, i.e., both parties must actively participate in the key agreement process.

	Party U	**Party V**

Static Data	N/A	N/A
Ephemeral Data	1. Domain parameters (p_e, q_e, g_e) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	1. Domain parameters (p_e, q_e, g_e) 2. Ephemeral private key r_V 3. Ephemeral public key t_V
Input	(p_e, q_e, g_e), r_U, t_V	(p_e, q_e, g_e), r_V, t_U
	$Z_e = t_V^{r_U} \bmod p_e$	$Z_e = t_U^{r_V} \bmod p_e$
Shared secret value	$ZZ = Z_e$	$ZZ = Z_e$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo*.	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo.*

5.1.3 dhOneFlow

Party U has only ephemeral keys, and Party V has only static keys. Party U's ephemeral private/public key pair is denoted (r_U, t_U). Party V's static private/public key pair is denoted (x_V, y_V). Both of these key pairs have been generated using the same static domain parameters (p_s, q_s, g_s).

This scheme corresponds to the One Pass Diffie-Hellman scheme of ANSI X9.63. The scheme provides Implicit Key Authentication to the initiator (U). Although there is no static key for the initiator to lose, some measure of Key Compromise Impersonation resilience is afforded to the initiator since this scheme enables the initiator to identify the recipient; the compromise of any other initiator long-term secrets does not leave the initiator vulnerable to an impersonating recipient. Interactive or store-and-forward communications (only 1 party actively participates) may be used.

	Party U	**Party V**
Static Data	N/A	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	N/A
Input	(p_s, q_s, g_s), r_U, y_V	(p_s, q_s, g_s), x_V, t_U
	$Z = y_V^{r_U} \bmod p_s$	$Z = t_U^{x_V} \bmod p_s$
Shared secret value	$ZZ = Z$	$ZZ = Z$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo.*	Invoke Key the Derivation function using *ZZ*, *keylen*, and *OtherInfo.*

5.1.4 dhHybrid1

Both Parties U and V have two pairs of private/public keys that are generated using the same set of static-key domain parameters (p_s, q_s, g_s). One key pair is static, and the other key pair is ephemeral. Party U and Party V generate individual static private/public key pairs, denoted (x_U,y_U) and (x_V,y_V), respectively, and individual ephemeral private/public key pairs, denoted (r_U,t_U) and (r_V,t_V), respectively.

There is no corresponding scheme in ANSI X9.63. This scheme provides Implicit Key Authentication to both parties, Known-Key Security and Forward Secrecy to both parties if explicit authentication is supplied to all session keys, and Unknown-Keyshare Resilience to both parties when knowledge of the private key (also known as proof of possession) is checked during the certification of the static public key. Interactive communications are required to use this scheme.

	Party U	**Party V**
Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_V 3. Ephemeral public key t_V
Input	(p_s, q_s, g_s), x_U, y_V, r_U, t_V	(p_s, q_s, g_s), x_V, y_U, r_V, t_U
	$Z_s = y_V^{x_U} \bmod p_s$	$Z_s = y_U^{x_V} \bmod p_s$
	$Z_e = t_V^{r_U} \bmod p_s$	$Z_e = t_U^{r_V} \bmod p_s$
Shared secret value	$ZZ = Z_e \| Z_s$	$ZZ = Z_e \| Z_s$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo*.	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo*.

5.1.5 dhHybrid2

Both Parties U and V have two pairs of private/public keys, each consisting of a static and an ephemeral key pair. The static key pair is generated using a set of static-key domain parameters (p_s, q_s, g_s). The ephemeral key pair is generated using a set of ephemeral-key domain parameters (p_e, q_e, g_e).

This scheme corresponds to the Full Unified Model scheme of ANSI X9.63. The scheme provides Implicit Key Agreement to both parties, Known-Key Security and Forward Secrecy to both parties if explicit authentication of all session keys is performed, and Unknown Keyshare Resilience to both parties if knowledge of the private key was checked during the certification of the static public keys. Interactive communications are required for this scheme.

	Party U	**Party V**

Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_e, q_e, g_e) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	1. Domain parameters (p_e, q_e, g_e) 2. Ephemeral private key r_V 3. Ephemeral public key t_V
Input	(p_s, q_s, g_s), x_U, y_V (p_e, q_e, g_e), r_U, t_V	(p_s, q_s, g_s), x_V, y_U (p_e, q_e, g_e), r_V, t_U
	$Z_s = y_V^{x_U} \bmod p_s$	$Z_s = y_U^{x_V} \bmod p_s$
	$Z_e = t_V^{r_U} \bmod p_e$	$Z_e = t_U^{r_V} \bmod p_e$
Shared secret value	$ZZ = Z_e \| \| Z_s$	$ZZ = Z_e \| \| Z_s$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*

5.1.6 dhHybridOneFlow

Party U has both a static and an ephemeral key pair, and Party V has one static key pair. Party U's static private/public key pair is denoted (x_U, y_U), and Party U's ephemeral private/public key pair is denoted (r_U, t_U). Party V's static key pair is denoted (x_V, y_V). All of these key pairs have been generated using the same domain parameters (p_s, q_s, g_s).

This scheme corresponds to the One Pass Unified Model scheme of ANSI X9.63. The scheme provides Implicit Key Authentication to both parties, Known-Key Security and Key-Compromise Impersonation Resilience to the initiator, Forward Secrecy to the initiator, and Unknown Key-Share Resilience to both parties when knowledge of the private key is checked during the certification of the static public keys. Interactive or store-and-forward communications may be used.

	Party U	**Party V**
Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	N/A
Input	(p_s, q_s, g_s), x_U, r_U, y_V	(p_s, q_s, g_s), x_V, y_U, t_U
	$Z_s = y_V^{x_U} \bmod p_s$	$Z_s = y_U^{x_V} \bmod p_s$
	$Z_e = y_V^{r_U} \bmod p_s$	$Z_e = t_U^{x_V} \bmod p_s$
Shared secret value	$ZZ = Z_e \| \| Z_s$	$ZZ = Z_e \| \| Z_s$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ, keylen,* and	Invoke the Key Derivation function

	OtherInfo.	using *ZZ, keylen,* and *OtherInfo.*

5.2 Key Agreement using MQV

In the key agreement schemes using the MQV algorithm, a single set of domain parameters is used for both static key pairs and ephemeral key pairs. Therefore, the subscripts will be omitted for the domain parameters.

5.2.1 MQV2

MQV2 uses the interactive form of the MQV algorithm. Both Parties U and V have two pairs of private/public keys using the same set of domain parameters (p_s, q_s, g_s). One key pair is static and the other key pair is ephemeral. Using the same set of domain parameters (p_s, q_s, g_s), Party U and Party V generate individual static private/public key pairs, denoted (x_U, y_U) and (x_V, y_V), respectively, and individual ephemeral private/public key pairs, denoted (r_U, t_U) and (r_V, t_V), respectively.

This scheme corresponds to the Full MQV scheme in ANSI X9.63. The scheme provides Implicit Key Agreement, Known-Key Security and Key Compromise Impersonation resilience to both parties, and Forward Secrecy to both parties if explicit authentication is supplied for all session keys. Interactive communications are required.

	Party U	**Party V**
Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_V 3. Ephemeral public key t_V
Input	(p_s, q_s, g_s), x_U, y_V, r_U, t_U, t_V.	(p_s, q_s, g_s), x_V, y_U, r_V, t_V, t_U.
	$w = \lceil \|\|q\|\|/2 \rceil$	$w = \lceil \|\|q\|\|/2 \rceil$
	$t_U' = (t_U \bmod 2^w) + 2^w$	$t_V' = (t_V \bmod 2^w) + 2^w$
	$S_U = (r_U + t_U' x_U) \bmod q_s$	$S_V = (r_V + t_V' x_V) \bmod q_s$
	$t_V' = (t_V \bmod 2^w) + 2^w$	$t_U' = (t_U \bmod 2^w) + 2^w$
	$Z_{MQV} = ((t_V (y_V^{t_V'}))^{S_U}) \bmod p_s$	$Z_{MQV} = ((t_U (y_U^{t_U'}))^{S_V}) \bmod p_s$
Shared secret value	$ZZ = Z_{MQV}$	$ZZ = Z_{MQV}$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*	Invoke the Key Derivation function using *ZZ, keylen,* and *OtherInfo.*

5.2.2 MQV1

MQV1 uses the store-and-forward form of the MQV algorithm. The MQV1 scheme is used in situations where two parties contribute different amounts of information and use different algorithms to obtain the common shared secret value. More precisely, Party U, the initiator, has two private/public key pairs. One key pair is static, denoted (x_U, y_U), and the other key pair is ephemeral, denoted (r_U, t_U). Party V, the receiving party, has one private/public key pair that is static. It is denoted (x_V, y_V).

This scheme corresponds to the One Pass MQV scheme in ANSI X9.63. The scheme provides Implicit Key Authentication to both parties, and Key- Compromise Impersonation Resilience to the initiator. Interactive or store-and-forward communications may be used.

	Party U	**Party V**
Static Data	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p_s, q_s, g_s) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p_s, q_s, g_s) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	N/A
Input	(p_s, q_s, g_s), x_U, y_V, r_U, t_U.	(p_s, q_s, g_s), x_V, y_U, t_U.
	$w = \lceil \|\|q\|\|/2 \rceil$	$w = \lceil \|\|q\|\|/2 \rceil$
	$t_U' = (t_U \bmod 2^w) + 2^w$	$y_V' = y_V \pmod{2^w} + 2^w$
	$S_U = (r_U + t_U' x_U) \bmod q_s$	$S_V = (x_V + y_V' x_V) \bmod q_s$
	$y_V' = y_V \pmod{2^w} + 2^w$	$t_U' = (t_U \bmod 2^w) + 2^w$
	$Z_{MQV} = (y_V (y_V^{y_V'}))^{S_U} \bmod p_s$	$Z_{MQV} = (t_U (y_U^{t_U'}))^{S_V} \bmod p_s$
Shared secret value	$ZZ = Z_{MQV}$	$ZZ = Z_{MQV}$
Derive *KeyingData*	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo.*	Invoke the Key Derivation function using *ZZ*, *keylen*, and *OtherInfo.*