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

Sharon Keller

February 10, 2000

Introduction

- Specifies key establishment schemes for the agreement of symmetric keys using Diffie-Hellman and MQV algorithms.
- Used to establish common shared secret information between parties, ex.:cryptographic keys

Major Steps of Key Agreement Process Using Diffie-Hellman or MQV Algorithms

1. Domain Parameter Generation
2. Exchange of Domain Parameters
3. Domain Parameter Validation
4. Generation of Private and Public Keys
5. Communication of Public Keys
6. Validation of Public Keys
7. Calculation of Shared Secret Number(s)
8. Key Derivation from Shared Secret Number(s)
9. Use of the Derived Key

Domain Parameter Generation

- **Input**: Integers L, m, and seed, where L=256n, for n>= 4, m>=160
- **Output**: (p,q,g) , pgenCounter, where
  - p is a prime
  - q is a prime factor of p-1
  - g is a q-order element in GF(p)
- **Actions**:
  1. Generate prime p such that $2^{(L-1)}<p<2^{L}$ and p-1 has a prime factor q, $2^{(m-1)}<q<2^{m}$ using the algorithm defined in Annex B.1.2.
  2. Select an element g with order q using the algorithm defined in Annex B.2.
  3. Output (p, q, g), pgenCounter

Domain Parameter Types

- **Static-key domain parameters**
  - domain parameters with which static private/public keys are generated.
  - Can also be used to generate ephemeral keys
- **Ephemeral-key domain parameters**
  - domain parameters with which ephemeral private/public keys are generated.
  - Not necessarily short-lived.
  - May be used to generate multiple ephemeral keys

Example of Domain Parameters

- Static Domain Parameters
  - L = 1024 (base 10)
  - m = 160 (base 10)
  - seed = d501 4e4b 60ef 2ba8 b621 1b40 62ba 3224 e042 7f4f
  - pgenCounter = 246 (base 10)
  - p_s = e001 e896 7db4 9353 e16f 8e89 220c cefc
    5c5f 12e3 dff8 f1d1 4990 12e6 ef53 e31f
    02ea cc5a ddf3 3789 35c9 5b21 ea3d 6f1c
    d7ce 6375 52ec 386c 0e34 f736 ad95 17ef
    fe5e 4da7 a86a f90e 2c22 8fe4 b9e6 d8f8
    f02d 20af 78ab b692 acbc 4b23 faf2 c5cc
    d49a 0c9a 8bcd 91ac 0c55 9201 e6c2 fd1f
    47c2 cb2a 88a8 3c21 0fc0 54db 292d bc45

Example of Domain Parameters (cont.)

- Static Domain Parameters (cont.)
  - q_s = 8647 17a3 9e6a ea7e 89c4 32ee 7743 1516 9677 c499
  - g_s = 1ce0 f669 2646 1197 ef45 c465 8b83 b8ab
    04a9 2242 6850 4d05 b819 8399 dd71 3718
    cc1f 245d 476c cf61 a2f9 3493 f41f 5552
    4865 57e6 d4ca a800 d6d0 db3c bf5a 954b
    208a 4eba f7e6 49fb 6124 d8a2 1ef2 f22b
    aaae 2921 1019 1051 4647 31b6 cc3c 93dc
    6e80 ba16 0b66 64a5 6cfa 96ea f1b2 8339
    8eb4 6164 e5e9 4384 ee02 24e7 1f03 7c23

Example of Domain Parameters

- Ephemeral Domain Parameters
  - L = 1024 (base 10)
  - m = 160 (base 10)
  - seed = d501 4e4b 60ef 2ba8 b621 1b40 62ba 3224 e042 7dd3
  - pgenCounter = 371 (base 10)
  - p_e = D757 262C 4584 C44C 211F 18BD 96E5 F061
    C4F0 A423 F7FE 6B6B 85B3 4CEF 72CE 14A0
    D3A5 222F E08C ECE6 5BE6 C265 8548 89DC
    1EDB D13E C8B2 74DA 9F75 BA26 CCB9 8772
    3602 787E 922B A844 21F2 2C3C 89CB 9B06
    FD60 FE01 941D DD77 FE6B 1289 3DA7 6EEB
    C1D1 28D9 7F06 78D7 722B 5341 C850 6F35
    8214 B16A 2FAC 4B36 8950 3878 11C7 DA33

Example of Domain Parameters (cont.)

- Ephemeral Domain Parameters (cont.)

q_e= C773 218C 737E C8EE 993B 4F2D ED30 F48E
DACE 915F

$h = g_e \text{ ^ } ((p_e-1)/q_e)) \bmod p_e$

g_e = 8226 9009 E14E C474 BAF2 932E 69D3 B1F1
8517 AD95 9418 4CCD FCEA E96E C4D5 EF93
133E 84B4 7093 C52B 20CD 35D0 2492 B395
9EC6 4996 25BC 4FA5 082E 22C5 B374 E16D
D001 32CE 71B0 2021 7091 AC71 7B61 2391
C76C 1FB2 E883 17C1 BD81 71D4 1ECB 83E2
10C0 3CC9 B32E 8105 61C2 1621 C73D 6DAA
C028 F4B1 585D A7F4 2519 718C C9B0 9EEF

Exchange of Domain Parameters

- Domain parameters may be:
  - generated by one party and transmitted to another party, or
  - generated by a third party
- Method for distribution of domain parameters not specified in this standard
- Result: Both parties have same set(s) of parameters

Domain Parameter Validation

- Party receiving the parameters should check or receive assurance that the parameters satisfy the arithmetic requirements.
- **Input**: (p, q, g), L, m, seed, pgenCounter
- **Output**: "Valid" or "Invalid"
- **Actions**:
  1. Verify whether p and q were generated using the Prime Generation Algorithm defined in Annex B1.2 with input L, m, and seed at the given value pgenCounter. If yes, go to step 2. Otherwise, output "Invalid".
  2. Verify whether g is a generator of order q by checking that $2 <= g <= (p-2)$ and that $g^q=1 \bmod p$. If yes, output "Valid". Otherwise, output "Invalid".

Private/Public Key Generation

- Using a bundle of given domain parameters and the following algorithm, a pair of private/public keys will be generated
- **Input**: (p, q, g)
- **Output**: x, y
- **Actions**:
  1. Select a statistically unique and unpredictable number x where $1 <= x <= (q-1)$.
  2. Calculate $y = g^x \bmod p$
  3. Output x, y

Private/Public Key Types

- **Static keys**
  - relatively long-lived
  - may be certified
  - y - public key
  - x - private key
- **Ephemeral keys**
  - relatively short-lived
  - may or may not be certified
  - t - public key
  - r - private key

Example of Private/Public Key Generation

- Static Private and Public Keys for U
  - x_U = 538d 3d64 274a 4005 9b9c 26e9 13e6 9153 237b 5583
  - y_U = bbe9 18dd 4b2b 941b 100e 8835 2868 fc62
    0438 a6db 32a6 9eee 6c6f 451c a3a6 d537
    7775 5bc1 370a cefe 2b8f 13a9 142c 5b44
    1578 8630 d695 b192 2063 a3cf 9def 6561
    274d 2401 e7a1 45f2 d8b9 3a45 17f4 19d0
    5ef8 cb35 5937 9d04 20a3 bf04 adfe a860
    b2c3 ee85 5890 f3b5 572b b4ef d78f 3768
    787c 7152 9d5e 0a61 4f09 8992 39f7 4b01

Example of Private/Public Key Generation

- Static Private and Public Keys for V
  - x_V = 7057 c2a0 9fcb 9716 ad48 88de c294 9677 a150 522f
  - y_V = a3f5 7dbe 9e2f 0ada a94e 4e6a f0e0 7147
    0e2e 412e de73 2a62 14c3 7c26 d4e9 9a54
    ba3d e749 8595 0ee9 14b2 9022 91dc ff61
    b2fc d1d0 1b11 14b6 0264 2b26 5d88 e08d
    bbe2 070b 48fb 0153 551e 5951 36f2 f9d1
    97fb 6612 845d edb8 9b2d 3e2b 8ceb 2a72
    409d 554c edeb 5502 ff8c b02e 0365 3f41
    b1ac a330 6bff 6df4 6de6 e10f 867c 4364

Example of Private/Public Key Generation

- Ephemeral Private/Public Keys for U
  - r_U = 5054 029c 7103 4ff1 3e3f 1960 6ac1 8359 dafc b5e1
  - $t_U = g_e \text{^} r_U \bmod p_e$ =
    29FA EBDE 6D75 216C C535 42B7 01FD 2319
    E3C2 029C 705F 32D7 5DFF 51FD FAA5 BB46
    2C19 9C92 F2FD 5343 1D57 5F39 8EAB 1DDD
    EB37 53BA F21B 3761 7DF8 C30D 9AC5 66D6
    B2D5 E693 18C6 F8CB 5F4D 3940 BEA1 4B1E
    F50D BD17 A58F 6CA3 7F35 0F68 DAC0 DF1B
    E689 137D 19ED 33B5 A5E6 CC54 54E2 DBA5
    6E8F 8644 3F63 136C 44C2 0F5F 10A4 ABB4

Example of Private/Public Key Generation

- Ephemeral Private/Public Keys for V
  - r_V = 6a4f 4e4b 7500 52a7 e9bc 14e1 a90a ffc0 c8a6 732e
  - $t_V = g_e \hat{} r_V \bmod p_e$ =

    41BF C74E B858 B850 88D8 5404 28A3 9A54

    A119 A725 FF8F F57C 3023 2697 85AA 93DE

    A450 69EF 1242 BADD 88BB 463E 13A2 535C

    CAAE 6CB1 C327 68CE 850F 1B24 6086 8810

    2BAC 1B10 7ABB D4E8 17A8 7D1F C9DE 6862

    9CB2 73D8 C8C1 55BA 7EDF A944 8458 520C

    0D90 65E3 F315 9FF4 C0FF 1E71 FBCC C1EC

    4d5e 0f04 74b3 dffa 72d1 450f 1d99 f27c

Communication of Static Public Keys

- Standard requires that communicating parties possess authentic copies of the static public keys of all other parties involved in key agreement process.
- Binding between entity and its static public key shall include checking that the entity possesses the corresponding private key (Proof of Possession(POP)).
- Means for achieving this is beyond scope of standard; recommended approach defined in ANSI X9.57 - Certificate Management

Communication of Ephemeral Public Keys

- May or may not be authenticated when transferred.

Public Key Validation

- The process of checking the arithmetic properties of a public key.
- May be part of certification process or may be done by receiving party.
- 5 methods of acceptable public key validation:
  - 3 are implicit methods
  - 2 are explicit methods
- Only one method MUST be carried out, but several methods CAN be applied to obtain greater assurance.

3 Implicit Public Key Validation Methods

- 1. Entity performs this by generating the public key using trusted routines. Can only be performed if the entity knows the private key associated with the public key being validated.
- 2. Entity receives assurance that another party (the assuring party) has implicitly validated the public key. Assuring party must know private key associated with the public key.

  Assuring party must be trusted for the lifetime of any key that is combined with the public key being validated using the key agreement schemes

3. Instead of validating the public key, the entity mitigates the risk of using an invalid public key.

2 Explicit Public Key Validation Methods

- 1. Entity performs explicit public key validation of the public key itself by using the following technique:
  - **Input**: (p, q, g) that have been validated, and y
  - **Output**: "Valid" or "Invalid"
  - **Actions**:
    1. Verify that 2 <= y <= p-2.
    2. Verify that $y^q = 1 \bmod p$.
    3. If 1 or 2 fail, output "invalid". Otherwise, output "valid".
- 2. Entity receives assurance that another party (the assuring party) has explicitly validated the public key by using the above technique.

  Assuring party must be trusted for the lifetime of any key that is combined with the public key being validated using the key agreement schemes

Calculation of a Shared Secret Number

- Each party calculates shared secret value using
  - domain parameters
  - other party's public key(s)
  - its own private and public key(s)

Calculation of a Shared Secret Number

- 2 basic algorithms used to calculate shared secret number:
  - Diffie-Hellman algorithm - named after W. Diffie and M. E. Hellman
  - MQV algorithm - named after Alfred Menezes, Minghua Qu and Scott Vanstone

Basic Diffie-Hellman Algorithm

- Party U executes the following algorithm:
- Input:
  - (p,q,g): a set of static (or ephemeral) key domain parameters.
  - y_v: V's public key
  - x_U: U's private key
- Actions:
  - Compute $Z=y_v \text{ ^ } x_U \bmod p$
  - Check whether Z=1. If yes, output "Failure", Otherwise, output Z.

Key Agreement Schemes using Diffie-Hellman

- dhStatic
- dhEphemeral
- dhOneFlow
- dhHybrid1
- dhHybrid2
- dhHybridOneFlow

Key Agreement Scheme - dhStatic

	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
Action 1	$Z_s = y_V \text{^} x_U \bmod p_s$	$Z_s = y_U \text{^} x_V \bmod p_s$
Output	$ZZ = oct(Z_s)$	$ZZ = oct(Z_s)$
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Agreement Scheme - dhStatic

- $Z_U = Z_V = y_V \text{^} x_U \bmod p_s = y_U \text{^} x_V \bmod p_s$
- $y_U = g_s \text{^} x_U \bmod p_s$
- $y_V = g_s \text{^} x_V \bmod p_s$
- $(g_s \text{^} x_V) \text{^} x_U \bmod p_s = (g_s \text{^} x_U) \text{^} x_V \bmod p_s$
- $g_s \text{^} (x_V * x_U) \bmod p_s = g_s \text{^} (x_U * x_V) \bmod p_s$

Example of dhStatic Key Agreement Scheme

- $y_U = g_s \text{^} x_U \bmod p_s$
- $y_V = g_s \text{^} x_V \bmod p_s$
- $Z_s = y_V \text{^} x_U \bmod p_s = y_U \text{^} x_V \bmod p_s =$

  4E6A CFFD 7D14 2765 EBF4 C712 414F E4B6
  AB95 7F4C B466 B466 0128 9BB8 2060 4282
  7284 2EE2 8F11 3CD1 1F39 431C BFFD 8232
  54CE 472E 2105 E49B 3D7F 113B 8250 76E6
  2645 8580 7BC4 6454 665F 27C5 E4E1 A4BD
  0347 0486 3229 81FD C894 CCA1 E293 0987
  C92C 15A3 8BC4 2EB3 8810 E867 C443 2F07
  259E C00C DBBB 0FB9 9E17 27C7 06DA 58DD

Key Agreement Scheme - dhEphemeral

	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
Action 1	$Z_e = t_V \text{^} r_U \bmod p_e$	$Z_e = t_U \text{^} r_V \bmod p_e$
Output	ZZ = oct(Z_e)	ZZ = oct(Z_e)
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Agreement Scheme - dhOneFlow

	Party U	Party V
Static Data	1. 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	1. N/A
Input	(p_s, q_s, g_s) r_U, y_V	(p_s, q_s, g_s), x_V, t_U
Action 1	$Z = y_V \text{^} r_U \bmod p_s$	$Z = t_U \text{^} x_V \bmod p_s$
Output	ZZ = oct(Z)	ZZ = oct(Z)
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Agreement Scheme - dhHybrid1

	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
Action 1	$Z_s = y_V \text{^} x_U \bmod p_s$	$Z_s = y_U \text{^} x_V \bmod p_s$
Action 2	$Z_e = t_V \text{^} r_U \bmod p_s$	$Z_e = t_U \text{^} r_V \bmod p_s$
Output	ZZ = oct(Z_e) || oct(Z_s)	ZZ = oct(Z_e) || oct(Z_s)
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Agreement Scheme - dhHybrid2

	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
Action 1	$Z_s = y_V \text{^} x_U \bmod p_s$	$Z_s = y_U \text{^} x_V \bmod p_s$
Action 2	$Z_e = t_V \text{^} r_U \bmod p_e$	$Z_e = t_U \text{^} r_V \bmod p_e$
Output	$ZZ = oct(Z_e) \mid\mid oct(Z_s)$	$ZZ = oct(Z_e) \mid\mid oct(Z_s)$
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Agreement Scheme - dhHybridOneFlow

	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. N/A
Input	(p_s, q_s, g_s), x_U, y_V, r_U	(p_s, q_s, g_s), x_V, y_U, t_U
Action 1	$Z_s = y_V \text{^} x_U \bmod p_s$	$Z_s = y_U \text{^} x_V \bmod p_s$
Action 2	$Z_e = y_V \text{^} r_U \bmod p_s$	$Z_e = t_U \text{^} x_V \bmod p_s$
Output	$ZZ = oct(Z_e) \mid\mid oct(Z_s)$	$ZZ = oct(Z_e) \mid\mid oct(Z_s)$
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

MQV Algorithm

- Variation of the Diffie-Hellman algorithm
- 2 Forms:
  - Interactive (MQV2) -symmetric
  - Store and Forward (MQV1) -asymmetric

Key Agreement Scheme - MQV2

	Party U	Party V
Static Data	1. Domain parameters (p, q, g) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p, q, g) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p, q, g) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	1. Domain parameters (p, q, g) 2. Ephemeral private key r_V 3. Ephemeral public key t_V
Input	(p, q, g), x_U, y_V, r_U, t_U, t_V	(p, q, g), x_V, y_U, r_V, t_V, t_U
	$w = \lceil ||q||/2 \rceil$	$w = \lceil ||q||/2 \rceil$
	$t_U' = (t_U \bmod 2^w) + 2^w$	$t_V' = (t_V \bmod 2^w) + 2^w$
	$S_U = (r_U + t_U' x_U) \bmod q$	$S_V = (r_V + t_V' x_V) \bmod q$
	$t_V' = (t_V \bmod 2^w) + 2^w$	$t_U' = (t_U \bmod 2^w) + 2^w$
	$Z_{MQV} = ((t_V(y_V \wedge t_V'))\wedge S_U) \bmod p$	$Z_{MQV} = ((t_U(y_U \wedge t_U'))\wedge S_V) \bmod p$
Output	$ZZ = oct(Z_{MQV})$	$ZZ = oct(Z_{MQV})$
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

MQV Algorithm - Store and forward form MQV1

- Both parties contribute different amounts of info
  - Initiating party contributes 2 pairs keys - static and ephemeral (Party U)
  - Receiving party contributes 1 pair of keys - static (Party V)
- Use different algorithms to obtain a shared secret value

Key Agreement Scheme - MQV1

	Party U - Initiator	Party V - Responder
Static Data	1. Domain parameters (p, q, g) 2. Static private key x_U 3. Static public key y_U	1. Domain parameters (p, q, g) 2. Static private key x_V 3. Static public key y_V
Ephemeral Data	1. Domain parameters (p, q, g) 2. Ephemeral private key r_U 3. Ephemeral public key t_U	N/A
Input	(p, q, g), x_U, y_V, r_U, t_U, w	(p, q, g), x_V, y_U, t_U, w
	$t_U' = (t_U \bmod 2^w) + 2^w$	$y_V' = (y_V \bmod 2^w) + 2^w$
	$S_U = (r_U + t_U' x_U) \bmod q$	$S_V = (x_V + y_V' x_V) \bmod q$
	$y_V' = (y_V \bmod 2^w) + 2^w$	$t_U' = (t_U \bmod 2^w) + 2^w$
	$Z_{MQV} = y_V (y_V \wedge y_V '))^{S_U} \bmod p$	$Z_{MQV} = ((t_U (y_U \wedge t_U'))^{S_V}) \bmod p$
Output	$ZZ = oct(Z_{MQV})$	$ZZ = oct(Z_{MQV})$
Derive Keying Data	Invoke Key Derivation function using ZZ, keylen, and OtherInfo	Invoke Key Derivation function using ZZ, keylen, and OtherInfo

Key Derivation from Shared Secret Number

- 2 key derivation functions defined:
  - Based on ASN.1 DER encoding
  - Based on concatenation of fixed-length fields

Key Derivation Function based on ASN.1

- Input:

  ZZ: Shared secret number

  keylen: length in bits of keying data to be generated

  OtherInfo: bit string in ASN.1 DER encoding consisting of:

  a. Key Specification Info

  1. AlgorithmID: a unique object identifier (OID) of the symmetric algorithm with which the derived key will be used, e.g. TDEA, HMAC, MAC, etc.
  2. Counter: 32-bit octet string

  b. Optional info including PartyUInfo, PartyVInfo, SuppPrivInfo, SuppPubInfo

Key Derivation Function based on ASN.1 (cont.)

Actions:

Let d = ⌈keylen/hashlen⌉

Initialize Counter to 1.

For I=1 to d,

Compute h_i = H(ZZ||OtherInfo) where h_i = hashvalue computed using appropriate hash function.

Increment Counter and I.

Compute Keying Data = leftmost keylen bits of $h_1||h_2||...||h_d$.

Output Keying Data - a bit string of length keylen bits.

Key Derivation based on Concatenation

- Input:

  ZZ: Shared secret number

  keylen: length in bits of keying data to be generated

  OtherInfo: bit string consisting of some data shared by the two entities intended to share the secret value ZZ.

Key Derivation Function based on Concatenation (cont.)

Actions:

Let d = $\lceil$keylen/hashlen$\rceil$

Initialize 32-bit octet string Counter as 1.

For I=1 to d,

Compute h_i = H(ZZ||Counter||OtherInfo) where h_i = hashvalue computed using appropriate hash function.

Increment Counter and I.

Compute Keying Data = leftmost keylen bits of $h_1||h_2||...||h_d$.

Output Keying Data - a bit string of length keylen bits.

Key Derivation

- This standard only defines how to extract a certain number of raw bits based on the shared secret number(s) and other inputs.
- The conditioning of the output data stream (e.g. parity adjustment) is beyond the scope of this standard.
- Similarly, decisions on how to parse and apply the output data stream into keys and initialization vectors, is also beyond the scope of this standard and is the responsibility of the key management protocol.

MAC Computation

- May optionally be computed using a key derived from a shared secret value and one of the key derivation methods defined.
- May be used for implementation validation
- Input: MacKey, MacData
- Actions:
  - Compute MacValue = MAC(MacKey, MacData)
  - Output MacValue

ANSI X9.42 Implementation Validation

- For purposes of validating implementations of this standard during an implementation validation test, value of MacData shall be the string
  - "ANSI X9.42 Testing Message"
  - followed by 16 bytes containing a 128-bit value used as a nonce.
- Default value of nonce - all zeros

Attributes Provided by Key Agreement Schemes

- Indicates the attributes provided by each key agreement scheme. Attributes include:
  - implicit key authentication
  - explicit key authentication
  - entity authentication
  - known-key security
  - forward secrecy
  - key-compromise impersonation resilience
  - unknown key-share resilience