Association Models Supplement to the Certified Wireless Universal Serial Bus Specification

March 2, 2006

Revision 1.0

REVISION HISTORY

The 1.0 revision of the specification is intended for product design. Every attempt has been made to ensure a consistent and implementable specification. Implementations should ensure compliance with this revision.

Revision	Issue Date	Comments
1.0	March 2, 2006	Final specification.

Please send comments via electronic mail to techsup@usb.org
For industry information, refer to the USB Implementers Forum web page at http://www.usb.org

ACKNOWLEDGEMENT OF TECHNICAL CONTRIBUTION

The authors of this specification would like to recognize the following people and companies who participated in the Wireless USB Association Models technical working group. The list is sorted alphabetically, first by company name, then by individual's last name.

Alereon	Nirmalendu Patra
Hewlett-Packard	Ed Beeman
Hewlett-Packard	Dave Patton
Intel	Brad Hosler
Intel	John Howard
Intel	Preston Hunt (Chair and Editor)
Intel	Rahman Ismail
Intel	John Keys
Intel	Vic Lortz
Intel	Rich Minter
Microsoft	Randy Aull
Microsoft	Fred Bhesania
Microsoft	Mark Maszak
Microsoft	Glen Slick
NEC Electronics	Masahiro Noda
NEC Electronics	Hiro Sakamoto
Nokia	N. Asokan
Nokia	Janne Marin
Nokia	Kaisa Nyberg
Nokia	Richard Petrie
Philips	Ernst Haselsteiner
Philips	Dave Holmes
Philips	Young Kim
Philips	Stefan Posch
Staccato Communications	Roberto Aiello
Staccato Communications	Billy Brackenridge
Staccato Communications	Shyam Narayanan
Staccato Communications	Philip Zimmermann
Texas Instruments	Jin-Meng Ho
Texas Instruments	Yoram Solomon

CONTENTS

TABLES

1 Introduction

This document defines an industry standard method for establishing a first-time connection between Certified Wireless USB hosts and devices. In Certified Wireless USB, the process of establishing a first-time connection between hosts and devices is called “association”. Association is a pre-requisite process that must be completed by hosts and devices prior to implementing the security requirements outlined in chapter 7 of the Wireless USB Specification 1.0.

This document is primarily targeted at peripheral developers, but provides valuable information for platform operating system developers. This specification can be used for developing new products and associated software.

2 Terms, References, and Conventions

2.1 Terms

CBAF	Cable-Based Association Framework
CC	Connection Context
CDID	Connection Device ID
CHID	Connection Host ID
CK	Connection Key
Conditioning	The process whereby a user grants permission to a host or device to perform new associations
HMAC	Keyed-Hash Message Authentication Code
HMAC-SHA-256	An HMAC that uses SHA-256 as the hash function
SHA-256	Secure Hash Algorithm

2.2 References

[FIPS180-2]	FIPS 180-2: Secure Hash Standard
[FIPS198A]	FIPS 198: The Keyed-Hash Message Authentication Code
[RFC2104]	RFC 2104: HMAC: Keyed-Hashing for Message Authentication
[RFC2631]	RFC 2631: Diffie-Hellman Key Agreement Method
[RFC2945]	RFC 2945: SRP Authentication and Key Exchange System, section 2
[RFC3526]	RFC 3526: More Modular Exponential Diffie-Hellman groups for Internet Key Exchange
[RFC4086]	RFC 4086: Randomness Recommendations for Security
[UNICODE]	Unicode 4.1.0 Specification
[USBOTG]	On-The-Go Supplement to the USB 2.0 Specification
[WUSB1]	Wireless Universal Serial Bus Specification 1.0

2.3 Diffie-Hellman Key Exchange

Diffie-Hellman allows two parties to agree upon a shared secret over an insecure medium without any prior secrets. By itself Diffie-Hellman is vulnerable to man-in-the-middle attacks, so Certified Wireless USB association methods that use Diffie-Hellman require user intervention to validate the connection. Diffie-Hellman is the only public key technology allowed for Certified Wireless USB association.

A man-in-the-middle attack is an attack in which an attacker is able to read, insert, and modify at will messages between the device and the host without being detected. The threat to Certified Wireless USB is that an attacker could trick the host or device into connecting to the attacker instead of the intended target.

Below is a brief summary of the generic Diffie-Hellman key exchange. Refer to [RFC2631] for a more detailed explanation.

1. Device and host agree to use a prime number p and base g.
2. Device chooses a secret integer A and sends $g^A \bmod p$ to the host.
3. Host chooses a secret integer B and sends $g^B \bmod p$ to the device.
4. Device computes $S = (g^B \bmod p)^A \bmod p = g^{AB} \bmod p$.

5. Host computes $S = (g^A \bmod p)^B \bmod p = g^{AB} \bmod p$.
6. Device and host use S as the basis for establishing a secure channel.

Please note that the actual implementation in this specification makes a protocol improvement to the generic Diffie-Hellman key exchange.

2.4 Secure Hash Algorithm

A cryptographic hash function takes a message of any length and produces a fixed-length string as an output. Whenever a one-way hashing function is required, Certified Wireless USB association will always use SHA-256 as specified in [FIPS180-2].

The nomenclature used throughout this document to denote the use of a SHA-256 hash will be:

Hash_Text = SHA-256(Message_Text)

The SHA-256 algorithm produces 256-bit long hashes.

2.5 Keyed-Hash Message Authentication Code (HMAC)

A keyed-hash message authentication code, or HMAC, is used to simultaneously verify both the data integrity and the authenticity of a message. See [RFC2104] or [FIPS198A] for more information.

Certified Wireless USB association uses HMAC as specified in [RFC2104]. The hashing function is SHA-256. The nomenclature used throughout this document to indicate the use of HMAC is:

$Digest = HMAC\text{-}SHA\text{-}256_{key}(Message\_Text)$

The secret HMAC key must be exactly 256 bits long (including any possible leading zero bits).

2.6 Byte Order

Throughout this document, the byte order conventions from [RFC2945] are used: Text representing integers is presented most significant byte first, with each byte consisting of two hexadecimal characters.

An n-byte string S can be converted to an integer as follows:

i = S[n-1] + 256 * S[n-2] + 256^2 * S[n-3] + ... + 256^(n-1) * S[0]

Where i is the integer and S[x] is the value of the x'th byte of S.

In human terms, the string of bytes is the integer expressed in base 256, with the most significant byte first.

For example, consider the text “4400 51D6 F0B5”. The integer represented by this string is:

i = B5H + (F0H * 256) + (D6H * 256^2) + (51H * 256^3) + (00H * 256^4) + (44H * 256^5)

Although numbers are expressed as most significant byte first in this document, note that all numbers are sent least significant byte first when transmitted across USB.

3 Association Overview and Requirements

The security chapter of [WUSB1] details the algorithms and protocols used by Certified Wireless USB to ensure secure operation. In particular, the Connection Context (CC) is introduced as the tuple (CHID, CDID, CK) comprising the connection host identification, connection device identification, and connection key. The CC is used to set up a secure communications channel in Certified Wireless USB between a host and a device.

The purpose of Certified Wireless USB association is to take a CC from the host and establish it on the device such that attackers cannot eavesdrop on the CK information in the CC or insert themselves as imposters in the exchange. Once the device has the CC, it follows the procedures outlined in [WUSB1] to communicate securely with the host.

Under normal operating conditions, association is a one-time process that is only necessary the very first time that a device is used with a host. On subsequent uses, the device and host will automatically connect using stored copies of the CC from a previous association.

The security design goal of Certified Wireless USB association is the same as the security design goal of Certified Wireless USB security, which is to provide security equivalent to that of wired USB.

3.1 Requirements For Hosts

Certified Wireless USB hosts (not including embedded hosts and dual-role devices, which are covered in section 3.3) must implement both the Cable Association Model (section 4) and the Numeric Association Model (section 5). Hosts must be capable of displaying at least 4 digits to the user. Hosts are required to implement both of these models to ensure that a Certified Wireless USB device will always be able to connect to a Certified Wireless USB host, regardless of which association method the device supports.

3.2 Requirements For Devices

Certified Wireless USB devices that provide an upstream facing wired USB interface (B-receptacle or equivalent) must implement the Cable Association Model. Devices with a display must implement the Numeric Association Model and be capable of displaying at least 2 digits to the user. Devices must implement either the Cable Association Model or the Numeric Association Model. Devices with both a display and an upstream facing wired USB interface must implement both the Cable and Numeric Association Models.

3.3 Requirements For Embedded Hosts and Dual-role Devices

This section applies to Certified Wireless USB embedded hosts, limited hosts, and dual-role devices, all of which are referred to in this section as the umbrella term “embedded hosts” in the interests of brevity and clarity.

Embedded hosts that provide a wired USB interface (A- or B-receptacle or equivalent) must implement the Cable Association Model. Embedded hosts with a display must implement the Numeric Association Model. Embedded hosts must implement either the Cable Association Model or the Numeric Association Model. Embedded hosts with both a display and a wired USB interface must implement both the Cable and Numeric Association Models. If any device on the embedded host’s targeted peripheral list (TPL) supports only the Numeric Association Model, then the embedded host must support the Numeric Association Model. (Refer to section 3.3 of [USBOTG] for more information about the TPL.) If any device on the embedded host’s TPL supports only the Cable Association Model, then the embedded host must support the Cable Association Model.

3.4 User Conditioning

Conditioning is the process where a user implicitly or explicitly grants permission to both the host and the device to allow an association. Certified Wireless USB requires mandatory user conditioning before every association.

Implicit permission is granted when the user performs some action such as plugging in a cable.

Explicit permission is granted when the user manually pushes a button, navigates a software menu, or enters something on a keypad.

Mandatory conditioning is required to prevent accidental association when there are multiple hosts and devices trying to connect to each other in the same physical area. Mandatory conditioning also ensures the security of both hosts and devices by helping to prevent active attacks. Without mandatory conditioning, it would be possible to associate a device to a host without the host owner's permission. Similarly, without mandatory conditioning, it would be possible for a host to gain access to a device without the device owner's permission.

3.5 User-perceived Response Time

The maximum time that a user must wait for any specific part of the association process must not exceed two seconds. This time is measured from the moment that the user makes some sort of input or conditioning step until the moment when the user is again able to take some sort of action.

3.6 Device-side Management of Connection Contexts

Devices that wish to connect to a host via Certified Wireless USB without having to repeat a new association each time must store the CC in non-volatile memory. Because each host requires a separate, unique CC, devices will need to provide memory for as many host associations as they wish to support. The number of CCs to support is left up to device manufacturers based on the usage scenarios for their devices. However, devices must provide non-volatile storage for at least one CC.

If a device runs out of room for CCs, it must replace an existing CC with the new one. It is up to the device to decide which CC to overwrite in order to make room for the new CC. If a device discovers an existing CC for a host that is trying to re-associate, the device must replace the existing CC for that host with the newer one being provided by the host.

3.7 Random Numbers

The association models described in this document use random numbers. Whenever a random number is needed, all of the following criteria must be satisfied:

- It must be derived from a physical entropy source, such as RF noise, thermal noise, or other unpredictable physical sources of entropy. [RFC4086] gives a detailed explanation of cryptographic grade random numbers and provides guidance for collecting suitable randomness.
- It must be freshly generated, meaning that it must not have been used in a previous calculation. For example, it is acceptable to freshly generate a number at power-up, and then use that number much later in time.
- It must be greater than or equal to two.
- It must be less than or equal to $2^L - 1$, where L is the number of random bits required.
- It must be chosen with equal probability from the entire available number space, e.g., $[2, 2^L - 1]$.

3.8 Data Structures for Association Information

The data structures used in this document to transfer association information between hosts and devices will use the attribute system defined in this section. A flexible attribute system is defined in which data

passed between hosts and devices is tagged with identification numbers that uniquely identify the data. Each data attribute transferred is always preceded by an identifier number and a length field.

Table 3-1 lists the attributes used in Certified Wireless USB:

Table 3-1: Association Attributes

Id	Name	Length	Description
0000H	AssociationTypeId	2	ID that uniquely identifies the underlying technology being associated (e.g., Certified Wireless USB = 0001H)
0001H	AssociationSubTypeId	2	ID of a specific association request/response pair. This ID uniquely identifies a specific transaction in the CBAF.
0002H	Length	4	Total length of the association request/response data packet.
0004H	AssociationStatus	4	Indicates the status of the association request to the device.
0008H	LangID	2	UNICODE language ID code
000BH	DeviceFriendlyName	0-64	Friendly name provided by the device to the host.
000CH	HostFriendlyName	0-64	Friendly name provided by the host to the device.
1000H	CHID	16	Connection Host ID
1001H	CDID	16	Connection Device ID
1002H	ConnectionContext	48	The CHID, CDID, and CK concatenated together.
1004H	BandGroups	2	This bitmap reports the UWB band groups that are supported by the host or device. See [WUSB1] section 7.4.1.

4 Cable Association Model

4.1 General Overview

The Cable Association Model extends the existing USB infrastructure (both in terms of hardware support as well as in user experience) by utilizing a USB cable to perform the first-time association between a host and a device.

In the Cable Association Model, the device is connected to the host with a USB cable for the first association. Once the association is complete, the cable is no longer required and subsequent use of the device can be done wirelessly without the cable. However, after association is complete, the device may choose to remain in wired mode if it is still connected via the cable and switch to wireless mode only when the cable is removed.

As discussed in section 3.4, the act of plugging in a cable is a type of implicit user conditioning. For extra security, a host may implement additional explicit user conditioning after a cable has been plugged in. This is not required and implementation is left up to the manufacturer.

The Cable Association Model uses the USB Cable-Based Association Framework (CBAF). A Certified Wireless USB device must report support for the CBAF only when connected via cable with its wired USB port. Thus, requests detailed in section 4.3 are available only when connected via cable.

4.2 Protocol Overview

This section outlines the steps in the Cable Association Model. This section is provided for informative purposes and security analysis only. The normative requirements are described in section 4.3.

1. The user connects the device to the host using a USB cable.
2. The host detects that the device supports the CBAF.
3. The host detects that the device is capable of configuring Certified Wireless USB using the CBAF.
4. The host sends its CHID to the device, along with other information such as the supported band groups.
5. If the device has a valid CC with a matching CHID, the device will respond to the host with the CDID from the CC.
6. If the CDID returned from the device matches the CDID in the host's copy of the CC, then the host must generate and send a new CC to the device. Because the device has proven that it had been associated previously, hosts that would otherwise require explicit user conditioning of the association may choose to skip the conditioning step.
7. Otherwise, if the CDID returned from the device did not match the host's copy, the host may optionally require explicit user conditioning to confirm the association. The host will then generate and send a new CC to the device. Explicit user conditioning is not required and implementation, if any, is left up to the manufacturer.
8. Upon receiving the CC, the device must store the CC in non-volatile memory, replacing any existing CC with a matching CHID if it exists.
9. First time association is complete: Host has securely transferred the CC to the device.

4.3 Normative Requirements

This section specifies what is required of Certified Wireless USB devices that implement the Cable Association Model. This section uses a general purpose Cable-Based Association Framework. This framework is extensible and can be used to associate any protocol, including Certified Wireless USB. The focus of this section is on how the general framework is used for the association of Certified Wireless USB devices over a wired USB connection. As such, some optional components of the CBAF, such as the interrupt endpoint and segmented data transfer support, are not discussed here.

The steps in using the CBAF to perform Certified Wireless USB association are:

1. Reporting/Discovery: The host detects that a device supports the CBAF when the device enumerates the CBAF Interface in the configuration descriptor set provided to the host when the device is attached via a USB 2.0 cable.
2. Getting Available Association Requests: The host queries the device for a list of the protocols that the device wants to associate using the CBAF.
3. Getting Existing Certified Wireless USB Associations: The host queries the device for existing Certified Wireless USB associations (if any) on the device.
4. Setting a New Certified Wireless USB Association: The host sets a new CC on the device.

The CBAF defines three USB control requests as shown in Table 4-1.

Table 4-1: CBAF Control Requests

bRequest	Name
01H	GET_ASSOCIATION_INFORMATION
02H	GET_ASSOCIATION_REQUEST
03H	SET_ASSOCIATION_RESPONSE

The default device behavior for the CBAF control requests is as follows:

Default state: Device behavior is not specified when requests are received while the device is in the default state.

Address state: Device must respond with Request Error.

Configured state: Requests are valid when the device is in the configured state.

4.3.1 Reporting/Discovery

A Certified Wireless USB device will report its support of the CBAF via the inclusion of the CBAF Interface in the default descriptor set. When a device that supports the CBAF is connected to the wired USB port of a host that also supports the CBAF, the host will identify the CBAF Interface and load the appropriate driver.

Table 4-2 represents a typical interface descriptor for a device that supports the CBAF.

Table 4-2: CBAF Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	Interface descriptor type (04H)
2	bInterfaceNumber	1	Number	Zero-based value identifying the interface
3	bAlternateSetting	1	Constant	Alternate settings are not used (00H)
4	bNumEndpoints	1	Number	Number of endpoints besides the control endpoint. 00H implies no optional interrupt endpoint. 01H implies an interrupt endpoint is available. All other values are RESERVED.
5	bInterfaceClass	1	Constant	EFH
6	bInterfaceSubClass	1	Constant	03H

7	bInterfaceProtocol	1	Constant	01H
8	iInterface	1	Number	Index of string descriptor describing this interface. This field must not be 00H.

4.3.2 Getting Available Association Types

After the CBAF Interface has been identified, the host will issue a GET_ASSOCIATION_INFORMATION control request. This request will return a list of all of the technologies that the device wants to configure using the CBAF.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ASSOCIATION_INFORMATION (01H)	0	Interface	Data Length	ASSOCIATION_INFORMATION

The device will return the data structure shown in Table 4-3 during the data phase of the request.

Table 4-3: ASSOCIATION_INFORMATION Format

Offset	Field	Size	Value	Description
0	Length	2	Number	The length of this entire data structure.
2	NumAssociationRequests	1	Number	Number of association requests in this response.
3	Flags	2	Bitmap	For the implementation described in this document, this value should be set to 0000H.
5	AssociationRequestsArray	N*A	Record	An array of ASSOCIATION_REQUEST records (see Table 4-4). N = NumAssociationRequests A = sizeof(ASSOCIATION_REQUEST)

The ASSOCIATION_INFORMATION data structure contains information about what technologies the device wants to associate using the CBAF. The ASSOCIATION_INFORMATION data structure contains one or more ASSOCIATION_REQUEST records (the actual number included is specified by NumAssociationRequests). Each ASSOCIATION_REQUEST record represents a request from the device for association information of some kind. Note that an association request should not be confused with a USB control request.

Table 4-4: ASSOCIATION_REQUEST Format

Offset	Field	Size	Value	Description
0	AssociationDataIndex	1	Number	Ordinal value used by GET_ASSOCIATION_REQUEST to retrieve the specific association information.
1	Reserved	1	Constant	This value is reserved and must be set to 00H.
2	AssociationTypeId	2	Number	This value globally and uniquely identifies the Association Type. For Certified Wireless USB, this value is 0001H.
4	AssociationSubTypeId	2	Number	This value uniquely identifies this request within the Certified Wireless USB Association Type attribute ID space.
6	AssociationTypeInfoSize	4	Number	This is the size in bytes of the specific information related to this request that the host will retrieve from the device when the host issues a GET_ASSOCIATION_REQUEST request. A value

				of zero indicates that there is no information to be transferred to the host for the specified AssociationSubTypeId.

The AssociationDataIndex field is used to manipulate the Association Type-specific data in the control transfers described in the following sections.

Each ASSOCIATION_REQUEST record contains AssociationTypeId, which uniquely identifies the technology being associated (in this case, for Certified Wireless USB, AssociationTypeId = 0001H). In addition, in the CBAF, each round-trip data exchange between the host and the device gets its own ASSOCIATION_REQUEST entry with a unique AssociationSubTypeId. Certified Wireless USB association requires two round-trip exchanges, and thus the minimal implementation of a device that supports only Certified Wireless USB will have a NumAssociationRequests value of 2.

Table 4-5 defines the possible values for AssociationSubTypeId in Certified Wireless USB.

Table 4-5: AssociationSubTypeId Values for Certified Wireless USB

SubTypeId	Name	Description
0000H	RetrieveHostInfo	This sub type indicates that the host must send its CHID value to the device. This is a mandatory sub type that must occur before the AssociateWUSB sub type.
0001H	AssociateWUSB	When an association request is issued with this sub type, the host will generate a response that contains the CC and return it to the device. This is a mandatory sub type that must occur after the RetrieveHostInfo sub type.

An example ASSOCIATION_INFORMATION response for a device that only supports Certified Wireless USB is shown in Table 4-6. This example can be used exactly as specified below by device manufacturers. The example shown is generic and will satisfy the minimum requirements for Certified Wireless USB devices; no further customization is required.

Table 4-6: Example ASSOCIATION_INFORMATION Implementation for Certified Wireless USB

Offset	Field	Size	Value	Data
0	Length	2	Number	0019H
2	NumAssociationRequests	1	Number	02H
3	Flags	2	Bitmap	0000H
5	AssociationDataIndex	1	Number	01H
6	Reserved	1	Constant	00H
7	AssociationTypeId	2	Number	0001H
9	AssociationSubTypeId	2	Number	0000H
11	AssociationTypeInfoSize	4	Number	00000000H
15	AssociationDataIndex	1	Number	02H
16	Reserved	1	Constant	00H
17	AssociationTypeId	2	Number	0001H
19	AssociationSubTypeId	2	Number	0001H
21	AssociationTypeInfoSize	4	Number	00000033H

Note that the data structure returned in response to the AssociateWUSB subtype is variable in length (see offsets 17-21 in Table 4-6). The AssociationTypeInfoSize will be between 2CH and 6CH and is dependent

on the value provided by the device for its DeviceFriendlyName (see Table 4-8). The example provided in Table 4-6 assumes that the device returns "SAMPLE" as its DeviceFriendlyName.

4.3.3 Getting Existing Certified Wireless USB Associations

After retrieving the ASSOCIATION_INFORMATION data structure, the host will start processing the ASSOCIATION_REQUEST records. The first association request for Certified Wireless USB must be RetrieveHostInfo (AssociationSubTypeId = 0000H), which means that the host must send its CHID value to the device. The host may also send its friendly name and will send the UWB band groups that it supports.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ASSOCIATION_RESPONSE (03H)	0101H	Interface	Data Length	HOST_INFO

The wValue field is comprised of the AssociationDataIndex (from the ASSOCIATION_REQUEST) in the upper byte and TransferFlags in the lower byte.

The CBAF features a segmented data transfer model in order to support response data sizes up to 4GB (normally USB control transfers are limited to 64KB). This data segmentation model is not described or used here because Certified Wireless USB association does not need it. Thus, TransferFlags will always be 01H in this document, indicating that this is the last block to be transferred.

The HOST_INFO data structure sent to the device during the data phase of the request is shown in Table 4-7. See section 3.8 for information about the attributes used in the HOST_INFO data structure.

Table 4-7: HOST_INFO Format

Offset	Field	Attribute Type Id	Attribute Length	Data
0	AssociationTypeId	0000H	0002H	0001H
6	AssociationSubTypeId	0001H	0002H	0000H
12	CHID	1000H	0010H	CHID_DATA: CHID value from the host
32	LangID	0008H	0002H	UNICODE language ID code for the HostFriendlyName field
38	HostFriendlyName	000CH	Variable	Null-terminated UNICODE string (shortest-form UTF-8) used to hold the host-assigned host friendly name. Attribute length must be between 0000H and 0040H.

The host may optionally provide to the device a short, UNICODED string representing a friendly name. If the device supports multiple CCs and a display, the device may use this friendly string, perhaps in combination with the CHID, to uniquely identify the host to its user. The default character encoding for Certified Wireless USB association is shortest-form UTF-8 as specified in [UNICODE].

The host will then ask the device to check if it has a CC that matches the CHID value that was just sent to it.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10100001B	GET_ASSOCIATION_REQUEST (02H)	0200H	Interface	Data Length	DEVICE_INFO

The wValue field is comprised of the AssociationDataIndex (from the ASSOCIATION_REQUEST) in the upper byte and BlockNumber in the lower byte.

The CBAF features a segmented data transfer model in order to support request data sizes up to 1MB (normally USB control transfers are limited to 64KB). This data segmentation model is not described or used here because Certified Wireless USB association does not need it. Thus, BlockNumber will always be 00H in this document, indicating that this is the first (and only) block to be transferred.

Each device has zero or more stored CCs. Each CC is the tuple (CHID, CDID, CK). Upon receiving the GET_ASSOCIATION_REQUEST request, the device will search its list of stored CCs and search for one that has a CHID that matches the CHID_DATA provided by the host. If the device finds a match, it will set CDID_RESPONSE to the corresponding CDID from the matching CC in the DEVICE_INFO data structure (Table 4-8), which is returned during the data phase of the GET_ASSOCIATION_REQUEST request. If the device does not have a matching CC, then it must return a value of zero (00H) for CDID_RESPONSE (although the other fields in the DEVICE_INFO data structure should still be returned with valid values).

See section 3.8 for information about the attributes used in the DEVICE_INFO data structure.

Table 4-8: DEVICE_INFO Format

Offset	Field	Attribute Type Id	Attribute Length	Data
0	Length	0002H	0004H	Length of this data structure
8	CDID	1001H	0010H	CDID_RESPONSE: CDID value from the device
28	BandGroups	1004H	0002H	This bitmap reports the UWB band groups that are supported by the host. See [WUSB1] section 7.4.1.
34	LangID	0008H	0002H	UNICODE language ID code for the DeviceFriendlyName.
40	DeviceFriendlyName	000BH	Variable	Null-terminated UNICODE string (shortest-form UTF-8) used to hold the device-assigned device friendly name. Attribute length must be between 0000H and 0040H.

If the device returns a zero CDID_RESPONSE, then the host must treat the device as an untrusted device. Hosts can implicitly accept the device without further user involvement, or hosts may optionally implement explicit user conditioning before continuing with the association. Implementation of any optional user conditioning is not defined by this specification and is left up to the host manufacturers.

If the device returns a non-zero CDID_RESPONSE, this indicates that the device has previously received a valid CC from the host. The host must check to see if it has an existing CC with a CDID that matches the CDID_RESPONSE from the device.

If there is not a match, the host must treat the device as if the device had returned a zero CDID_RESPONSE (see above).

If there is a match, the host must respond by sending a new CC to the device as described in section 4.3.4. Because the device has been authorized previously, the host should choose to skip any additional explicit user conditioning for the transmission of the new CC. The device will then replace its existing CC for that host with the updated one. The new CC must include a freshly generated CK; the CHID and CDID may remain the same.

4.3.4 Setting a New Certified Wireless USB Association

After receiving the DEVICE_INFO response from the device and determining whether a CC should be transferred to the device, the host will issue a SET_ASSOCIATION_RESPONSE control request:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00100001B	SET_ASSOCIATION_RESPONSE (03H)	0201H	Interface	Data Length	CC_DATA

The format of the CC_DATA data structure for a successful association is shown in Table 4-9. All fields are required. See section 3.8 for information about the attributes used in the CC_DATA data structure.

Table 4-9: CC_DATA Format (Success Case)

Offset	Field	Attribute Type Id	Attribute Length	Data
0	AssociationTypeId	0000H	0002H	0001H
6	AssociationSubTypeId	0001H	0002H	0001H
12	Length	0002H	0004H	Length of this data structure in bytes
20	Connection Context	1002H	0030H	Concatenation of CHID, CDID, CK
72	BandGroups	1004H	0002H	This bitmap reports the UWB band groups that are supported by the host. See [WUSB1] section 7.4.1.

The format of the CC_DATA data structure for an unsuccessful association is shown in Table 4-10. All fields are required. If a failure occurs, products must display a failure notification to the user. Devices can discard the CC_DATA data structure after the failure notification has been displayed.

Table 4-10: CC_DATA Format (Failure Case)

Offset	Field	Attribute Type Id	Attribute Length	Data
0	AssociationTypeId	0000H	0002H	0001H
6	AssociationSubTypeId	0001H	0002H	0001H
12	Length	0002H	0004H	Length of this data structure in bytes
20	AssociationStatus	0004H	0004H	Indicates the status of the association request: 0001H = Association unsuccessful 0002H = Malformed association request 0003H = Association type not supported All other values are reserved.

4.3.5 Cable Association Is Complete

Once the procedures in sections 4.3.1 through 4.3.4 have been completed, the device has all of the information that it needs to communicate securely with the host. When the device is ready to communicate over Certified Wireless USB with the host, it connects to the host using the CC information established in section 4.3.4. The host and device then complete the four-way handshake and begin normal operation as specified in [WUSB1].

5 Numeric Association Model

5.1 General Overview

In the Numeric Association Model, the first time association between a device and a host is performed over the ultra-wideband radio using standard Certified Wireless USB control requests. This process is sometimes referred to as "in-band" association. The Diffie-Hellman protocol is used to establish a temporary secure channel. To guard against a man-in-the-middle attack, the host and device each display a value that is derived from the Diffie-Hellman keys, and the user is asked to verify that the two values match.

The following steps explain the numeric model from the user's perspective:

1. The user directs the device to begin searching for a host that is accepting connections from "new" devices. The exact method for how a device implements this user control is beyond the scope of this specification. However, it could be a specific button, a selection provided on a display, etc.
2. The user directs the host to begin accepting connections from new devices. The host will begin advertising over the Certified Wireless USB channel that it is accepting "new" device connections.
3. The host and device each display a short number (two to four digits) on their respective displays.
4. If the values match, the user confirms the association on the host and on the device by pressing a button on each side (or taking some equivalent action).
5. If the values do not match, association has failed (please note that user validation is the only indication of success or failure in the numeric association model). The device must return to the state that the device was in before the association process began. Products must display a failure notification to the user in the event of a failed association.

5.2 Protocol Overview

This section outlines the steps in the Numeric Association Model. This section is provided for informative purposes and security analysis only. The normative requirements are described in section 5.3.

1. User conditions the host to allow new associations and the device to start a new association.
2. Device listens for a host accepting new device connections and initiates a "new" connection when one is found.
3. Device generates a fresh random secret A and computes $PK_D = g^A \bmod p$. The A and PK_D values are prohibited from being hard coded in the device at manufacture time.
4. Device computes the hash SHA-256(PK_D || N_D) and sends the hash to the host, where N_D is the number of digits that the device is capable of displaying. This hash commits the device to the PK_D and N_D values, without revealing the values until later, after the host's public key and is revealed.
5. Host generates a fresh random secret B and computes $PK_H = g^B \bmod p$. The B and PK_H values are prohibited from being hard coded into the host at manufacture time.
6. Host sends PK_H to the device. Device aborts the association if PK_H equals 1 or p-1 (see Table 5-1 for p's value).
7. Device sends PK_D and N_D to the host. Host aborts the association if PK_D equals 1 or p-1.
8. Host computes SHA-256(PK_D || N_D) and verifies the result with the hash commitment received from the device previously. Host aborts the association if the values don't match.
9. Host computes the shared secret DHKey = SHA-256($PK_D^B \bmod p$).
10. Device computes the shared secret DHKey = SHA-256($PK_H^A \bmod p$).
11. In order to protect against man-in-the-middle attacks, both sides compute a common value V = SHA-256(PK_D || PK_H || "displayed digest") and show 2-4 digits of this number to the user on their displays. (Steps 11 and 12 can be done before or in parallel with steps 9 and 10.)
12. User verifies that the numbers shown on the host and device match and presses "ok" (or takes some equivalent action) on both the host and the device. If the user presses "don't match" or user confirmation is not received on both the host and the device within the timeout period, then the association is aborted and a failure indication is displayed to the user. The timeout period must be at least 20 seconds. There is no maximum timeout period.

13. If the user approves the association, the host and device both compute the connection key CK = the first 128 bits of HMAC-SHA-256$_{DHKey}$("connection key").
14. The host also sends to the device any remaining non-private information that it needs to complete the association (e.g., host id, device id, friendly name, etc.)
15. First time association is complete: Host has securely transferred all of the necessary information for the device to construct the CC.
16. If any other applications need additional keys for whatever purpose, then a key derivation key KDK is computed as KDK = HMAC-SHA-256-$_{DHKey}$("key derivation key"). The KDK value can then be used immediately or stored away for later use as the keying material for any other purposes.
17. All variables used in this section are erased except for the CC and the KDK (if any).

Note that all public keys used in this specification are "ephemeral public keys", meaning that they are only used for one key exchange and then discarded.

5.3 Normative Requirements

This section specifies what is required of Certified Wireless USB devices that implement the Numeric Association Model. The constants g and p used throughout this section are defined in Table 5-1.

Table 5-1: Constants Used in the Numeric Association Model

Constant	Description	Value
g	Diffie-Hellman generator	2
p	Diffie-Hellman modular exponential group (a prime number) This value is defined in [RFC3526] as the 3072-bit MODP group (group id 15).	FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AAAC42D AD33170D 04507A33 A85521AB DF1CBA64 ECFB8504 58DBEF0A 8AEA7157 5D060C7D B3970F85 A6E1E4C7 ABF5AE8C DB0933D7 1E8C94E0 4A25619D CEE3D226 1AD2EE6B F12FFA06 D98A0864 D8760273 3EC86A64 521F2B18 177B200C BBE11757 7A615D6C 770988C0 BAD946E2 08E24FA0 74E5AB31 43DB5BFC E0FD108E 4B82D120 A93AD2CA FFFFFFFF FFFFFFFF

5.3.1 Starting the Association

In order to implement the Numeric Association Model, a Certified Wireless USB device must:

1. Be conditioned by the user to start a new association.
2. Listen for a host allowing new connect notifications. Hosts indicate that they are enabled to accept new connects by setting the "ALL" value in the "Connection Availability" field of bmAttributes in the Host Information Element of the MMCs transmitted on the Certified Wireless USB channel (see section 7.5.2 of [WUSB1]). A device must search for a new host for at least 30 seconds but not more than 2 minutes (measured from when the user first conditions the device). A host must hold its new connection window open for at least 60 seconds but not more than 5 minutes (measured from when the user first conditions the host). These timeout conditions only apply to section 5.3.1 and are disregarded once a host and device start section 5.3.2.
3. Issue a DN_Connect to the host with the "New Connection" bit set (see section 7.6.1 of [WUSB1]) and with the CDID field set as follows: If the device has an existing CC with a CHID that matches the host's CHID, the device must set the CDID field of the DN_Connect to the existing CDID value from the CC. In all other cases, the device must generate a random number

for the CDID as required by section 6.2.10.3 of [WUSB1]. See section 3.7 of this document for random number requirements.

4. Receive a ConnectAck response from the host which issues the device an unauthenticated Certified Wireless USB device address.
5. Perform the procedures outlined in sections 5.3.2 through 5.3.12 with the host using SET_SECURITY_DATA and GET_SECURITY_DATA requests (refer to section 7.3.2.6 of [WUSB1] for detailed information about these requests).

5.3.2 Host Retrieves Hash Commitment From Device

Device generates its public key $PK_D = g^A \bmod p$. The resulting PK_D value is a 3072-bit (384-byte) number.

A is a 256-bit random numbers as described in section 3.7. Devices are prohibited from hard coding A and PK_D at manufacture time. Devices are required to freshly generate A and PK_D every single time they perform an association procedure as specified in this section. Note that the security of the Numeric Association Model depends on the secrecy of A's value. It must never be exposed or used for any purpose other than as described in this specification.

Device prepares the data structure M_3 as shown in Table 5-4 and computes SHA-256(M_3). When computing the hash, all 385 bytes of M_3 must be hashed, including any leading zero bytes.

By sending the hash rather than the public key itself, the device is forced to commit to the PK_D and N_D values, and the host is forced to respond with its own public key without knowledge of the device's public key. This enables the user to thwart man-in-the-middle attacks by comparing or entering a short verification number in section 5.3.7. If the user confirms that these values match on both sides, then the device and host both know that the association has succeeded.

Note that if the Diffie-Hellman public keys are not freshly-generated, an eavesdropper of a prior exchange could discover their values in advance and compute key pairs that would yield the same verification prefixes and thereby enable a man-in-the-middle attack.

After the device has been assigned an unauthenticated device address, the host will retrieve M_1 from the device by issuing the following control request:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_SECURITY_DATA	1	0	Size of M_1	M_1

Device returns M_1 during the data stage of the GET_SECURITY_DATA request. The format of M_1 is shown in Table 5-2.

Table 5-2: M_1 Format

Offset	Field	Size	Value	Description
0	Version	1	Constant	01H
1	SHA-256(M_3)	32	Number	SHA-256 hash of the M_3 data structure.

5.3.3 Host Sends Its Public Key to Device

Host generates its public key $PK_H = g^B \bmod p$. The resulting PK_H value is a 3072-bit (384-byte) number.

B is a 256-bit random number as described in section 3.7. Hosts are prohibited from hard coding B and PK_H at manufacture time. Hosts are required to freshly generate B and PK_H every single time they perform an association procedure as specified in this section. Note that the security of the Numeric Association Model depends on the secrecy of B's value. It must never be exposed or used for any purpose other than as described in this specification.

Host prepares M_2 as shown in Table 5-3.

Table 5-3: M_2 Format

Offset	Field	Size	Value	Description
0	Version	1	Constant	01H
1	PK_H	384	Number	Host's Diffie-Hellman public key

After retrieving M_1, the host issues a SET_SECURITY_DATA control request to transmit M_2 to the device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_SECURITY_DATA	2	0	Size of M_2	M_2

Upon receiving PK_H, the device must immediately check for insecure public keys. The device must abort the association if PK_H equals 1 or p-1.

5.3.4 Host Retrieves Device's Public Key and Display Size

After sending M_2, the host issues the following control request to retrieve M_3 from the device:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
10000000B	GET_SECURITY_DATA	3	0	Size of M_3	M_3

Device returns M_3 during the data stage of the GET_SECURITY_DATA request. The format of M_3 is shown in Table 5-4.

Table 5-4: M_3 Format

Offset	Field	Size	Value	Description
0	PK_D	384	Number	Device's Diffie-Hellman public key
384	N_D	1	Number	The number of digits that the device can display. Must be between 02H and 04H.

Upon receiving PK_D, the host must immediately check for insecure public keys. The host must abort the association if PK_D equals 1 or p-1.

Next, the host must compute SHA-256(M_3) and compare this value to the value that was received in message M_1 from section 5.3.2. If the values do not match, then the host must erase all temporary values and abort the association process. The values that must be erased include B, PK_H, PK_D, N_D, M_1, M_2, M_3, and SHA-256(M_3). Failure to follow this requirement may result in a compromise of the security of the system.

Note that the device has no way of knowing if the host aborts the association in this section. If the device supports a "cancel association" feature and if the host directs the user to do so, then the user may manually abort the association on the device side. Otherwise, the device will eventually timeout, at which point it will abort the association.

5.3.5 Device Derives Shared Secret and Verification Number

Device computes DHKey = SHA-256(PK_H^A mod p). The resulting DHKey value is a 256-bit (32-byte) number.

Device computes the verification number V_D = first 32 bits of SHA-256(M_V). Table 5-5 shows the format of the M_V data structure. The V_D computation can be done before or in parallel with the DHKey computation.

Table 5-5: M_V Format

Offset	Field	Size	Value	Description
0	PK_D	384	Number	Device's public key
384	PK_H	384	Number	Host's public key
768	"displayed digest"	16	Constant	Hexadecimal value: 646973706c6179656420646967657374

Device computes number of digits to display to user $N = N_D$.

Device displays the verification number ($V_D \bmod 10^N$) to the user in decimal form, using only the numbers "0" through "9" (ASCII 30H - 39H). Exactly N digits must be displayed to the user and the number must be padded with ASCII 30H characters ("0") at the beginning if the verification number is not long enough (i.e., a verification number of "23" would be displayed as "0023" if N is 4).

Note: Section 5.3.5 occurs in parallel with section 5.3.6.

5.3.6 Host Derives Shared Secret and Verification Number

Host computes DHKey = SHA-256($PK_D^B \bmod p$).

Host computes the verification number V_H = first 32 bits of SHA-256(M_V). Table 5-5 shows the format of the M_V data structure. The V_H computation can be done before or in parallel with the DHKey computation.

Host computes number of digits to display to user $N = \min(N_D, 4)$.

Host does one of the following two actions:

1) Host displays the verification number ($V_H \bmod 10^N$) to the user in decimal form, using only the numbers "0" through "9" (ASCII 30H - 39H). Exactly N digits must be displayed to the user and the number must be padded with ASCII 30H characters ("0") at the beginning if the verification number is not long enough (i.e., a verification number of "23" would be displayed as "0023" if N is 4).
2) Host prompts the user to input the number shown on the device's screen.

Note: Section 5.3.6 occurs in parallel with section 5.3.5.

5.3.7 User Confirms Verification Number

The user verifies the association in one of two ways, depending on whether the host displayed the verification number in section 5.3.6:

1) If the host displayed the verification number (option 1), the user visually verifies that the numbers displayed on the host and device match (i.e., $V_D \bmod 10^N == V_H \bmod 10^N$) and presses a button on both the host and the device. If the numbers do not match or if the user does not respond within the timeout period, then the association has failed and the user either presses a "don't match" button or allows the association to time out.
2) If the host prompted the user to input the verification number (option 2), then the host will confirm that the value entered by the user (V_U, which should be $V_D \bmod 10^N$) matches the value derived from section 5.3.6 (i.e., $V_U == V_H \bmod 10^N$). If the values match, then the verification has succeeded and the host must indicate this to the user. The user must then press a button on the device to complete the association process. If the values do not match or if the user does not respond within the timeout period, then the association has failed and the host must indicate this to the user. The user then either presses a "don't match" button on the device or allows the association to time out.

The timeout period referred to in this section must be at least 20 seconds. There is no specified maximum timeout period (manufacturers may choose a value that best suits the needs of their products). Both the host and device begin this timeout period after section 5.3.4 is completed. The host and device maintain independent timers and the association process has failed if either the host or the device timer expires.

If section 5.3.7 fails for any reason, the host and device must erase all temporary values and restart the entire association process at section 5.3.1. The values that must be erased include DHKey, PK_D, PK_H, A, B, N, N_D, V_H, V_D, M_1, M_2, M_3, M_V, SHA-256(M_3), and SHA-256(M_V). Failure to follow this requirement may result in a compromise of the security of the system.

5.3.8 Host Sends Association Information to Device

After the user has confirmed the verification number, the host sends M_4 to the device by issuing the following control request:

bmRequestType	bRequest	wValue	wIndex	wLength	Data
00000000B	SET_SECURITY_DATA	4	0	Size of M_4	M_4

In order to prepare M_4 (Table 5-6), the host must provide its CHID and the CDID for the device. The host can choose whatever value it wants for the CDID.

The host derives the connection key CK by computing the first 128 bits of HMAC-SHA-256_{DHKey}("connection key").

The CK value must be stored in persistent memory along with the CHID and CDID on the host side. However, for security reasons, the CK value will not be sent to the device in the M_4 message. The device will derive the CK value on its own in the next step.

If the host already had an existing CC for this device (based on the CDID value provided by the device in the DN_Connect request at the beginning of the association), then the host must replace any existing CC data with the CHID, CDID, and CK from this section.

See section 3.8 for information about the attributes used in the M_4 data structure.

Table 5-6: M_4 Format

Offset	Field	Attribute Type Id	Attribute Length	Data
0	AssociationTypeId	0000H	0002H	0001H
6	AssociationSubTypeId	0001H	0002H	0001H
12	Length	0002H	0004H	Length of this data structure in bytes
20	AssociationStatus	0004H	0004H	Indicates whether the association was successful. A 0H value indicates success and a 1H value indicates a failure. All other values are reserved.
28	CHID	1000H	0010H	Connection Host ID
48	CDID	1001H	0010H	Connection Device ID
68	BandGroups	1004H	0002H	This bitmap reports the UWB band groups that are supported by the host. See [WUSB1] section 7.4.1.
74	LangID	0008H	0002H	UNICODE language ID code for the HostFriendlyName field
80	HostFriendlyName	000CH	Variable	Null-terminated UNICODE string (shortest-form UTF-8) used to hold the host-assigned host friendly name. Attribute length must be between 0000H and 0040H.

In order for the device to connect to the host automatically in the future without requiring re-association, the device must store the CHID, CDID, and CK in non-volatile memory.

5.3.9 Device Prepares and Stores CC

Device computes the connection key CK = first 128 bits of HMAC-SHA-256$_{DHKey}$("connection key")

After computing CK, the device must store the CC (CHID, CDID, and CK) in persistent memory.

5.3.10 Additional Keys For Other Applications

If any other applications need additional keys for whatever purpose, then a key derivation key KDK is computed on both the host and device as KDK = HMAC-SHA-256$_{DHKey}$("key derivation key"). The KDK value can then be used immediately or stored away for later use as the keying material for any other purposes.

In order to protect the security of DHKey and the CC, the KDK is the only value that is allowed to be used by other applications for any reason.

5.3.11 Host and Device Erase All Unneeded Variables

The host and device must erase all temporary values computed or generated in this section, including DHKey, PK_D, PK_H, A, B, N, N_D, V_D, V_H, M_1, M_2, M_3, M_4, M_V, SHA-256(M_3), and SHA-256(M_V). The CC and the KDK (if any) are the only values that should not be erased. Failure to follow this requirement may result in a compromise of the security of the system.

5.3.12 Numeric Association Is Complete

Once the procedures in sections 5.3.1 through 5.3.11 have been completed, the device has all of the information that it needs to communicate securely with the host. When the device is ready to communicate over Certified Wireless USB with the host, it connects to the host using the CC information established in section 5.3.9. The host and device then complete the four-way handshake and begin normal operation as specified in [WUSB1].

5.4 Test Vectors

This section provides test vectors for the mathematical calculations described in section 5.3. These test vectors have been verified by at least two independent implementations. Values shown are in hexadecimal format (most significant byte first) unless otherwise indicated. See section 2.6 for more information about how to interpret the text representing these numbers.

5.4.1 Random Numbers A and B

This section contains two sample 256-bit random numbers, A and B.

A =

```
4400 51d6 f0b5 5ea9 67ab 31c6 8a8b 5e37 d910 dae0 e2d4 59a4 8645
9caa df36 7516
```

B =

```
5dae c786 7980 a324 8ce3 578f c75f 1b0f 2df8 9d30 6fa4 52cd e07a
048a ded9 2656
```

5.4.2 Modular Exponentiation (g^A mod p, g^B mod p)

This section shows the computation of g^A mod p and g^B mod p. The values for g and p are from section 5.3 (Table 5-1). The values for A and B are from section 5.4.1.

$PK_D = g^A$ mod p =

```
5a0d 3d4e 049f aa93 9ffa 6a37 5b9c 3c16 a4c3 9753 d19f f7da 36bc
391e a72f c0f6 8c92 9bdb 4005 52ed 84e0 900c 7a44 c322 2fd5 4d71
4825 6862 886b fb40 16bd 2d03 c4c4 cf47 6567 c291 770e 47bd 59d0
aa53 23cf ddfc 5596 e0d6 558c 480e e8b0 c625 9983 4d45 81a7 96a0
1981 4687 8916 4504 afbd 29ce 9936 e86a 290c 5f00 f8ba 986b 4801
0f3e 5c07 9c7f 351d dca2 ee1f d508 46b3 7bf7 463c 2b0f 3d00 1b13
17ac 3069 cd89 e2e4 927e d3d4 0875 a604 9af6 49d2 dc34 9db5 995a
7525 d70a 3a1c 9b67 3f54 82f8 3343 bd90 d45e 9c39 62dc 4a4b f2b4
adb3 7e91 66b2 ddb3 1ccf 11c5 b9e6 c98e 0a9a 3377 abba 56b0 f428
3b2e aa69 f536 8bc1 07e1 c225 99f8 8dd1 924d 0899 c5f1 5346 2c91
1a82 9307 8aef ee9f b238 9a78 5483 3fce a61c fecb b49f 828c 361a
981a 5fed ecf1 3796 ae36 e36c 15a1 6670 af96 996c 3c45 a30e 900e
18c8 58f6 232b 5f70 72bd d9e4 7d7f c612 46ef 5d19 7657 39f3 8509
2843 79bc 319d 9409 e8fe 236b d29b 0335 a5bc 5bb0 424e e44d e8a1
9f86 4a15 9fda 907d 6f5a 30eb c0a1 7e36 28e4 90e5
```

$PK_H = g^B$ mod p =

```
dc14 c6f6 d85b 3d58 b54a bb30 6d55 6829 2ed7 85d3 9ed7 3643 666a
1b4a 4684 654f 88bb edf0 414c 59c7 0dd9 90b4 47b3 c325 0a4a 2367
3ea9 361a 79be 3376 0906 ef12 7627 fa9e 7f91 07e7 3675 9cff 990c
44fc e240 7e7c e1c7 d61a 83b8 5c82 85a9 bf94 7cc1 e582 642a 8a86
3e4e 0d57 f258 4b25 5229 c4d3 5355 1e86 ac2b bce4 13c7 e554 1cc2
e68d 7101 d578 30cd e1c9 1bd4 8c03 d190 1472 01f3 9697 f65c c2f4
45e8 5162 3bea 585c 8205 d8e8 ca91 b54d aefb 6fe5 ac46 e942 b5ea
6e04 495b d2f6 cb11 88c1 b44a 342e 5dab 2917 165e 0935 d743 69b7
6698 68c9 d4d5 b148 33f3 1e56 9499 1e73 353a 33f5 f4dc 61ff 5752
517b 7180 6da2 e47e fc78 d22d d8da c4f1 1501 9d57 5d60 b787 6140
4413 bff6 e314 329b f1e5 2b92 38f8 7964 a5a3 00c7 26c0 950f ac94
6459 3c30 6ece 4d92 813f d714 2e16 18b3 efbb 3fea 25f9 e177 0859
2507 d8be 73ef d569 761e 7ff4 b016 edd0 c5c3 85a8 ec16 1a44 f2d6
7c1c 6b39 7d8f 6c3f a797 bcd9 5e3f b8f4 ecba 7ebf 6620 570e f491
4e75 eaf9 752b a471 faf7 ccc5 5373 069c 2153 1194
```

5.4.3 M_3 and SHA-256(M_3)

This section shows the computation of the M_3 data structure used in sections 5.3.2 and 5.3.4.

This example assumes PK_D as shown in section 5.4.2 and $N_D = 2$.

$M_3 = PK_D \,||\, N_D$ =

```
5a0d 3d4e 049f aa93 9ffa 6a37 5b9c 3c16 a4c3 9753 d19f f7da 36bc
391e a72f c0f6 8c92 9bdb 4005 52ed 84e0 900c 7a44 c322 2fd5 4d71
4825 6862 886b fb40 16bd 2d03 c4c4 cf47 6567 c291 770e 47bd 59d0
aa53 23cf ddfc 5596 e0d6 558c 480e e8b0 c625 9983 4d45 81a7 96a0
1981 4687 8916 4504 afbd 29ce 9936 e86a 290c 5f00 f8ba 986b 4801
0f3e 5c07 9c7f 351d dca2 ee1f d508 46b3 7bf7 463c 2b0f 3d00 1b13
17ac 3069 cd89 e2e4 927e d3d4 0875 a604 9af6 49d2 dc34 9db5 995a
7525 d70a 3a1c 9b67 3f54 82f8 3343 bd90 d45e 9c39 62dc 4a4b f2b4
adb3 7e91 66b2 ddb3 1ccf 11c5 b9e6 c98e 0a9a 3377 abba 56b0 f428
3b2e aa69 f536 8bc1 07e1 c225 99f8 8dd1 924d 0899 c5f1 5346 2c91
1a82 9307 8aef ee9f b238 9a78 5483 3fce a61c fecb b49f 828c 361a
981a 5fed ecf1 3796 ae36 e36c 15a1 6670 af96 996c 3c45 a30e 900e
18c8 58f6 232b 5f70 72bd d9e4 7d7f c612 46ef 5d19 7657 39f3 8509
2843 79bc 319d 9409 e8fe 236b d29b 0335 a5bc 5bb0 424e e44d e8a1
9f86 4a15 9fda 907d 6f5a 30eb c0a1 7e36 28e4 90e5 02
```

SHA-256(M_3) =

```
02a8 af45 c0e7 1a9a 92f7 f46f ebde bb02 ca13 6d68 0db7 b9bb 74ff
06a1 9bbd 3d19
```

5.4.4 Diffie-Hellman Shared Secret and DHKey

This section shows the computation of $(g^A \bmod p)^B \bmod p$ and $(g^B \bmod p)^A \bmod p$ (these two expressions yield the same result). The values for g and p are from section 5.3 (Table 5-1) and the values for $g^A \bmod p$ and $g^B \bmod p$ are from section 5.4.2.

SharedSecret = $(g^A \bmod p)^B \bmod p = (g^B \bmod p)^A \bmod p$ =

```
bcec d344 c6f4 2f35 aced 542b 7ceb 684a 623b f9ad 3ebf 2a64 9afc
be7c 9fd2 127e 1d2b 08ba b247 3cdd bf44 fa3f 98a5 6ad7 5ee7 5a66
e0dc 0bfb c246 fb57 9a6d 5275 3222 ea82 e4fc ee51 fef5 3d24 af4c
5f00 fdba f7b3 c55a 0e4f 8b5f 2e27 51b5 ca3f 9898 8ca3 08b5 11bd
2e35 7767 84dc 852f 8519 9eb0 52aa 12a3 b4f5 e9ca be79 8610 11a6
c34e 9b11 6f06 fcb3 b59e e739 75cf 6529 118f 63b0 68f2 2422 cbac
11e1 18f1 fc3a 06c7 9787 f8c0 ee90 f878 64b9 fac6 5f75 6725 6abd
1da2 1122 d83e 4026 e9d4 835e 5e77 10cd 5ab4 7e88 7d10 dd75 56bf
5f27 679d 634a a1c2 f8a8 cfc3 1859 cb72 d0e0 8efa 9b01 a88b 213f
b604 63fa eb63 2449 7b77 4420 76cf 81b9 9556 34dc eeeb bcc1 9b17
1857 d823 d190 798f 391e 1910 b7ce eccc baa5 0856 32cf 7660 bb06
9b82 721f 7c33 61a4 512b 8a25 ac32 f16e a332 2e87 2f54 d2db 8ea7
b815 e125 cd47 b0c6 2a51 ae42 5f6c 6956 8ec4 3bb8 810f 62e8 447c
cb19 0f59 ad1c 212a 50aa 20f0 66c5 732c a60e 6728 ea2b c91a 82fe
cc80 6f81 3330 a694 4aff c69a 562f 3501 514c c70f
```

DHKey = SHA-256(SharedSecret) =

```
2d42 85c2 3196 26f2 c2c7 2c5a 2855 3f54 41d2 c521 8c0c fbb6 60cc
57a1 dfa1 a68f
```

5.4.5 M_V and Verification Numbers

This section uses DHKey from section 5.4.4 and PK_D and PK_H from section 5.4.2.

$M_V = PK_D \,||\, PK_H \,||$ "displayed digest" =

```
5a0d 3d4e 049f aa93 9ffa 6a37 5b9c 3c16 a4c3 9753 d19f f7da 36bc
391e a72f c0f6 8c92 9bdb 4005 52ed 84e0 900c 7a44 c322 2fd5 4d71
4825 6862 886b fb40 16bd 2d03 c4c4 cf47 6567 c291 770e 47bd 59d0
aa53 23cf ddfc 5596 e0d6 558c 480e e8b0 c625 9983 4d45 81a7 96a0
1981 4687 8916 4504 afbd 29ce 9936 e86a 290c 5f00 f8ba 986b 4801
0f3e 5c07 9c7f 351d dca2 ee1f d508 46b3 7bf7 463c 2b0f 3d00 1b13
17ac 3069 cd89 e2e4 927e d3d4 0875 a604 9af6 49d2 dc34 9db5 995a
7525 d70a 3a1c 9b67 3f54 82f8 3343 bd90 d45e 9c39 62dc 4a4b f2b4
adb3 7e91 66b2 ddb3 1ccf 11c5 b9e6 c98e 0a9a 3377 abba 56b0 f428
3b2e aa69 f536 8bc1 07e1 c225 99f8 8dd1 924d 0899 c5f1 5346 2c91
1a82 9307 8aef ee9f b238 9a78 5483 3fce a61c fecb b49f 828c 361a
981a 5fed ecf1 3796 ae36 e36c 15a1 6670 af96 996c 3c45 a30e 900e
18c8 58f6 232b 5f70 72bd d9e4 7d7f c612 46ef 5d19 7657 39f3 8509
2843 79bc 319d 9409 e8fe 236b d29b 0335 a5bc 5bb0 424e e44d e8a1
9f86 4a15 9fda 907d 6f5a 30eb c0a1 7e36 28e4 90e5 dc14 c6f6 d85b
3d58 b54a bb30 6d55 6829 2ed7 85d3 9ed7 3643 666a 1b4a 4684 654f
88bb edf0 414c 59c7 0dd9 90b4 47b3 c325 0a4a 2367 3ea9 361a 79be
3376 0906 ef12 7627 fa9e 7f91 07e7 3675 9cff 990c 44fc e240 7e7c
e1c7 d61a 83b8 5c82 85a9 bf94 7cc1 e582 642a 8a86 3e4e 0d57 f258
4b25 5229 c4d3 5355 1e86 ac2b bce4 13c7 e554 1cc2 e68d 7101 d578
30cd e1c9 1bd4 8c03 d190 1472 01f3 9697 f65c c2f4 45e8 5162 3bea
585c 8205 d8e8 ca91 b54d aefb 6fe5 ac46 e942 b5ea 6e04 495b d2f6
cb11 88c1 b44a 342e 5dab 2917 165e 0935 d743 69b7 6698 68c9 d4d5
b148 33f3 1e56 9499 1e73 353a 33f5 f4dc 61ff 5752 517b 7180 6da2
```

```
e47e fc78 d22d d8da c4f1 1501 9d57 5d60 b787 6140 4413 bff6 e314
329b f1e5 2b92 38f8 7964 a5a3 00c7 26c0 950f ac94 6459 3c30 6ece
4d92 813f d714 2e16 18b3 efbb 3fea 25f9 e177 0859 2507 d8be 73ef
d569 761e 7ff4 b016 edd0 c5c3 85a8 ec16 1a44 f2d6 7c1c 6b39 7d8f
6c3f a797 bcd9 5e3f b8f4 ecba 7ebf 6620 570e f491 4e75 eaf9 752b
a471 faf7 ccc5 5373 069c 2153 1194 6469 7370 6c61 7965 6420 6469
6765 7374
```

SHA-256(M_V) =

```
031f 77f7 70e1 2d25 4a21 dc4a fd7c d029 9940 2069 6ea5 e4b8 2bee
c796 b5ba 20ca
```

$V_D = V_H$ = First 32 bits of SHA-256(M_V) =

```
031f 77f7
```

$V_D \bmod 10^2 = V_H \bmod 10^2$ =

```
75 (decimal)
```

$V_D \bmod 10^3 = V_H \bmod 10^3$ =

```
975 (decimal)
```

$V_D \bmod 10^4 = V_H \bmod 10^4$ =

```
3975 (decimal)
```

5.4.6 Connection Key

CKDigest = $HMAC\text{-}SHA\text{-}256_{DHKey}$(“connection key”) =
$HMAC\text{-}SHA\text{-}256_{DHKey}$(`636f 6e6e 6563 7469 6f6e 206b 6579`) =

```
39f5 98ca 86a4 36ee 2017 7f30 1b5d 3ce6 2edb aee8 4d3e ac9a 54e4
dacd ea84 e6ed
```

CK = first 128 bits of CKDigest =

```
39f5 98ca 86a4 36ee 2017 7f30 1b5d 3ce6
```

5.4.7 Key Derivation Key

This section uses DHKey from section 5.4.4.

KDK = $HMAC\text{-}SHA\text{-}256_{DHKey}$("key derivation key") =
$HMAC\text{-}SHA\text{-}256_{DHKey}$(`6b65 7920 6465 7269 7661 7469 6f6e 206b 6579`) =

```
cde6 5a7d a070 aff6 d294 0c03 0a62 df2c 086c 10e9 39bc 0081 8a39
e856 3a82 cc27
```