Universal Serial Bus
Content Security Method 4
Elliptic Curve Content Protection Protocols

CERTICOM

USB 1.0 Release Candidate

Revision 0.9

January 31, 2000

Revision History

Revision	Date	Filename	Author	Description
0.9	1/25/2000	CSM4_v0_9		USB DWG promotion to .9
0.8a	12/23/1999	Csm4_v0_8a		Added detailed USB interface descriptions.
0.8	11/1/1999	Csm4_v0_8		Promoted to .8 at 10/22/1999 USB DWG meeting.
0.7	9/22/1999	Csm4_v0_7		Separated CSM Appendices into individual CSM specification per Sept 1999 CSWG meeting

Contributors

Simon Blake-Wilson	Certicom
Peter de Rooij	Certicom

Please send comments via electronic mail to pderooij@certicom.com

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Purpose

This paper describes the USB services, functions, and processes required for the following two Elliptic Curve Content Protection Protocols (ECCPP) to be used with the "USB Device Class Definition for Content Security Devices" [CS]:

1. A method known as 'the Basic Elliptic Curve Content Protection Protocol' which performs device authentication using the elliptic curve digital signature algorithm (ECDSA) as specified in ANSI X9.62 [9.62].
2. A method known as 'the Enhanced Elliptic Curve Content Protection Protocol' which performs mutual host-device authentication and content encryption using the elliptic curve Diffie-Hellman protocol.

Both these methods use elliptic curve cryptography to provide security. Elliptic curve cryptography (ECC) is increasingly becoming the algorithm of choice for providing security in constrained environments. ECC has recently been approved by ANSI for securing financial transactions and is also being standardized within IEEE, ISO, NIST, and several other standards bodies.

The low computational cost of ECC, combined with minimized protocol overhead, make the methods proposed here particularly efficient compared to other possibilities. Both the methods will require assignment of a 128-bit GUID to enable their inclusion in the USB content protection specification.

1.2 Scope

This appendix describes the USB command structure and Content Security Interface (CSI) services necessary to perform the mentioned Elliptic Curve Content Protection Protocols (Basic and Enhanced ECCPP).

The Content Security Class specification allows Content Security Methods (CSM) to define additional requests as needed. Four additional Requests are defined to transfer commands and responses between the Host and Device. In addition, USB notification values are defined for the USB Content Security Notification format.

1.3 Related Documents

[CS] Universal Serial Bus Device Class Definition for Content Security Devices.

[USB 1.1] Universal Serial Bus Specification Version 1.1.

[USB CCS] USB Common Class Specification Version 1.0.

[X9.62] ANSI X9.62-1999, *Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)*, American Bankers Association, 1999.

1.4 Terms and Abbreviations

CS	Content Security
CSC	Content Security Class; refers to: *Universal Serial Bus Device Class Definition for Content Security Devices [CS]*
CSI	Content Security Interface
CSM	Content Security Method
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Standard, see ANSI X9.62.

2 USB Content Security Class Additions

The USB Device Class Definition For Content Security Devices allows Content Security Methods to define additional services as needed. Basic and Enhanced ECCPP require two additional USB Requests to transfer the ECCPP commands and responses. An Interrupt IN notification service is needed to allow USB devices to initiate authentication.

2.1 Requests

The Elliptic Curve Content Protection Protocols (ECCPP) require two additional USB requests to transfer the commands (host requests, including request data) and responses rather than defining a unique USB request for each individual request and corresponding response.

There are two additional requests that provide for the transport of encrypted data over the control endpoint.

This section details the structure of these requests.

2.1.1 Request Format

The General Request format ECCPP for is as follows:

Offset	Field	Size	Value	Description
0	*bmRequestType*	1	Bitmap	Characteristics of request: D7: Data transfer direction 0 = Host-to-device 1 = Device-to-host D6..5: Type 1 = Class D4..0: Recipient 1 = Interface 2 = Endpoint
1	*bRequest*	1	Value	USB ECCPS (CSM-4) Requests *PUT_CMD*, *GET_RESP*
2	*wValue*	2	Value	The high byte of *wValue* is reserved and set to a value of 0. The low byte is the *bMethod*[CSM-4] in the CS Standard Descriptor section 2.3.4, where CSM-4 refers to the index *n* corresponding to CSM-4
4	*wIndex*	2	Value	The high byte is the Channel ID. The low byte is the Interface number of the Content Security Interface (CSI).
6	*wLength*	2	Count	Byte length of the request or response frame.

Table 1. General Request Format

2.1.2 Command Request PUT_CMD

The *PUT_CMD* is used to transfer an ECCPP command with the corresponding parameters from the Host to the Device.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
0 01 00001B	*PUT_CMD* (0x80)	High Byte: 00 reserved Low Byte: bMethod[CSM-4]	High Byte: Channel ID Low Byte: CSI Interface Number	Length of command	command (including parameters)

Table 2. PUT_CMD Command Request

2.1.3 Command Request GET_RESP

The *GET_RESP* is used to transfer ECCPP response data from the Device to the Host.

bmRequestType	bRequest	wValue	wIndex	wLength	Data
1 01 00001B	*GET_RESP* (0x81)	High Byte: 00 reserved Low Byte: bMethod[CSM-4]	High Byte: Channel ID Low Byte: CSI Interface Number	Length of response	response

Table 3. GET_RESP Command Request

2.2 Content Security Interrupt IN Notification (IINS)

The USB Interrupt IN service is somewhat of a misnomer; it is implemented such that the Host periodically polls the USB Device. This provides the Device with an opportunity to send a notification to the Host. Recall that USB is designed so that the Host has total control over whom and when a compliant Device may access and use the USB.

The ECCPP methods use the standard IINS format as described in [CS].

2.3 USB ECCPP Descriptors

This section describes information relevant to the instantiation and use of USB ECCPP Content Security Class descriptors and associated USB descriptors. At the end of this section is a subsection that describes processes and attributes that are common to the descriptor outlined in this section.

2.3.1 Device Descriptor

Fields of Note: **bDeviceClass** is set to zero in order to cause loading of all descriptors.

2.3.2 Configuration Descriptor

Determined and Set by Device Manufacturer.

2.3.3 Content Security Interface Descriptor

Field	Value	Description
bLength	0x09	Size of this descriptor, in bytes: 9
bDescriptorType	0x04	Specified by Table 9-5 of USB 1.1
bInterfaceNumber	Number	Number of interface. A zero-based value identifying the index in the array of concurrent interfaces

Field	Value	Description
		supported by this configuration. bAlternateSetting Number Value used to select an alternate setting for the interface identified in the prior field.
bNumEndpoints	Number	Number of endpoints used by this interface (excluding endpoint 0) are CSM dependent.
bInterfaceClass	Class	Content Security Interface Class codes (assigned by the USB).
bInterfaceSubClass	Number	1 for Basic ECCPP, 2 for Enhanced ECCPP.
bInterfaceProtocol	0x00	Not used. Must be set to 0.
iInterface	Index	SBM, Index of a string descriptor that describes this interface.

Table 4. Content Security Interface Descriptor

2.3.4 CS Channel Descriptor

Field	Value	Description
bLength	Number	Byte length of this descriptor.
bDescriptorType	0xXX	CHANNEL_DESCRIPTOR, Specified by Table A.2 of USB DCD CSC
bChannelID	Number	Number of the Channel, must be a zero-based value that is unique across the device.
bmAttributes	Number	D7..D5: Reserved and set to zero D4..D0: Recipient Type 0 = Not used 1 = Interface 2 = Endpoint 3..31 = Reserved
bRecipient	Number	Identifier of the target recipient.
		If the Recipient field of bmAttributes = 1, then the value in the bRecipient field is an interface number. If the Recipient field of bmAttributes = 2, then the value in the bRecipient field is an endpoint address, where: D7..D5: Direction 0 = OUT 1 = IN D6..D4: Reserved and set to zero D3..D0: Endpoint number
bMethod[0]	Index	Index of a class-specific CSM descriptor that describes one of the Content Security Methods offered by the device. Must be a one-based value that is unique across the device. The value of 0 (zero) is reserved and must not be used in this field.
…		…
bMethod[N]	Index	Index of a class-specific CSM descriptor that describes one of the Content Security Methods offered by the device. Must be a one-based value that is unique across the device. The value of 0 (zero) is reserved and must not be used in this field.

Table 5. CS Channel Descriptor

2.3.5 USB ECCPP Content Security Method Descriptor

Field	Value	Description
bLength	Number	Byte length of this descriptor.

bDescriptorType	0xXX	CSM_DESCRIPTOR, Specified by Table A.2 of USB DCD CSC [CS]
bMethodID	CSM	Index of this class-specific CSM descriptor, must be a one-based value that is unique across the device. The bMethodID value of 0 is reserved (for more information, see the definition of the Get Channel Settings and Set Channel Settings requests in section 6).
ICSMDescriptor	Index	Index of string descriptor that describes the Content Security Method.
bcdVersion	0x0010	CSM Descriptor Version in Binary-Coded Decimal (i.e., version 2.10 is 0x0210).
guidMethod	→	128-bit GUID Assigned by CSC [CS, Appendix A]: *A12278E1-5572-11d3-B939-00A0C9BA4C6C* *[Note PdR: the GUID does not appear in CS anymore! Who assigns them now?]*

Table 6. USB ECCPP Content Security Method Descriptor

2.3.6 USB ECCPP String Descriptor

Field	Value	Description
bLength	Number	Byte length of this descriptor.
bDescriptorType	0x03	Specified by Table 9-5 of USB 1.1
bString	→	The value of this field is as follows and contained within the square brackets **[Elliptic Curve Content Protection Protocol Version 1.00]**

Table 7. USB ECCPP String Descriptor

2.3.7 General Descriptor Implementation Details

There are several descriptor descriptions containing data that talks about an "Index of". This Index maybe either a byte offset from a Device specific address or it may be the natural numbers 0, 1, 2, 3, …, which are by some Device specific method correlated to the associated object. This index may not be sequential or the same across devices.

3 Basic Elliptic Curve Content Protection Protocol

This appendix describes a protocol which provides device authentication and can be implemented on a low-cost USB Content Security Device. The basic authentication protocol is only rigorous enough to be suitable for low-cost consumer devices in the typical consumer environment, but does provide after-the-fact evidence of criminal intent.

The protocol is identified by a 128-bit GUID value that is specified in [CS].

3.1 Overview

The protocol is based on the elliptic curve digital signature algorithm (ECDSA). In short, to execute the protocol the host sends a random challenge to the device being authenticated, and the device signs the challenge using ECDSA and returns the signature along with its certificate to the host.

This protocol is designed to be efficient on low-cost devices. It is estimated that ~2K of ROM and ~300 bytes of RAM are required to implement the ECDSA signing performed during the protocol by the device. Used in a slow 16-bit micro-controller device, ECDSA is approximately 4 times faster than DSA, and the protocol takes in the order of hundreds of milliseconds to complete. Used in a slow 8-bit micro-controller device without a multiplier, where use of DSA or RSA may be infeasible, the protocol takes in the order of a second to complete.

The host always initiates the Basic ECCPP; either because it has determined itself that content security is required, or because it is notified by the device that content security is required.

The basic elliptic curve content protection protocol is shown in the following diagram.

Host		Device
Generate 128-bit random R	$\xrightarrow{R}$	Sign R using $K_{PrivDev}$ (the device private key).
Validate $Cert_{TA}(K_{PubDevice})$ and verify *Signature*.	$\xleftarrow{Signature,\ Cert_{TA}(K_{PubDevice})}$	

Figure 1. The Basic Elliptic Curve Content Protection Protocol

The protocol itself consists of a device authentication stage. The protocol must be preceded by an initialization stage which is performed during the setup of each device and host. These stages are described in more detail in the subsequent sections.

3.2 Stage 0: Initialization

Initialization is described for informational purposes only. Its execution is outside the scope of USB.

Each device generates an ECDSA key pair consisting of a private key $K_{PrvDevice}$ and a public key $K_{PubDevice}$ and obtains from the Trust Authority an X.509 certificate $Cert_{TA}(K_{PubDevice})$ containing its public key.

Each host obtains the ECDSA public key K_{PubTA} of the Trust Authority.

All signatures (including signatures on certificates) will be performed using ECDSA as specified in ANSI X9.62 and IEEE P1363. Devices will use NIST's 163-bit elliptic curve sect163k1 when they produce signatures, and the TA will use NIST's 283-bit elliptic curve sect283k1 when it signs certificates. (See http://csrc.nist.gov/encryption for the NIST curves.)

3.3 Stage 1: Authentication

To authenticate a device, the host first generates a 128-bit random challenge R_{Host} and sends it to the device, in the parameter to the *PUT_CMD* request. In response to this request, the device signs the challenge using ECDSA with its private key $K_{PrvDevice}$.

The host retrieves this signature, along with the device certificate $Cert_{TA}(K_{PubDevice})$ in the response, using the *GET_RESP* request.

PUT_CMD	0x01, 16, R
GET_RESP	0x01, $\|\|Cert_{TA}(K_{PubDevice})\|\|$, $Cert_{TA}(K_{PubDevice})$, $\|\|Signature\|\|$, $Signature$

Table 8. Parameters to Requests in Basic ECCPP, Authentication Stage

Finally, the host validates the device's certificate, optionally checks that the device's certificate has not been revoked, and verifies the device's signature on its challenge.

If all these checks are successful, the host completes authentication of the device and subsequently transfers content in the clear to the device.

4 Enhanced Elliptic Curve Content Protection Protocol

This appendix describes a protocol which provides mutual host-device authentication and content encryption.

The protocol is identified by a 128-bit GUID value that is specified in [CS].

4.1 Overview

The protocol is based on the elliptic curve Diffie-Hellman protocol (ECDH). In short, to execute the protocol the host and device first exchange certificates and random challenges. The host and the device may then exchange MACs computed using a MAC key derived from the Diffie-Hellman shared secret. Finally the host and the device exchange content encrypted using encryption keys derived from the Diffie-Hellman shared secret. If MACs are exchanged, the protocol provides explicit mutual authentication, while if MACs are not exchanged, the protocol provides implicit mutual authentication since only bona fide devices will be able to decrypt content.

The enhanced elliptic curve content protection protocol is shown in the following diagram. Dashed lines in the figure indicate optional messages; optional operations are enclosed in square brackets.

Host		Device
Generate 128-bit random R_{Host}	R_{Host}, $Cert_{TA}(K_{PubHost})$ $\longrightarrow$	Generate 128-bit random R_{Device}
Validate $Cert_{TA}(K_{PubDevice})$ Derive shared key from Diffie-Hellman shared secret and both random numbers.	R_{Device}, $Cert_{TA}(K_{PubDevice})$ $\longleftarrow$	Validate $Cert_{TA}(K_{PubHost})$. Derive shared key from Diffie-Hellman shared secret and both random numbers.
[Generate forward MAC]	[*forward MAC*] $\dashrightarrow$	[Verify forward MAC]
[Verify forward MAC]	[*backward MAC*] $\dashleftarrow$	[Generate backward MAC]
Exchange encrypted content	$\longleftrightarrow$	Exchange encrypted content

Figure 2. The Enhanced Elliptic Curve Content Protection Protocol

The protocol itself consists of a certificate exchange stage, followed by a key derivation stage, followed by an optional MAC exchange stage, followed by a content exchange phase. The protocol must be preceded by an initialization stage which is performed during the setup of each device and host. These stages are described in more detail in the subsequent sections.

4.2 Stage 0: Initialization

Initialization is described for informational purposes only. Its execution is outside the scope of USB.

Each device generates an ECDH key pair consisting of a private key $K_{PrvDevice}$ and a public key $K_{PubDevice}$ and obtains from the Trust Authority an X.509 certificate $Cert_{TA}(K_{PubDevice})$ containing its public key.

Each host generates an ECDH key pair consisting of a private key $K_{PrvHost}$ and a public key $K_{PubHost}$ and obtains from the Trust Authority an X.509 certificate $Cert_{TA}(K_{PubHost})$ containing its public key.

Each device and host obtains the ECDSA public key K_{PubTA} of the Trust Authority.

All devices and host will use static ECDH with the cofactor Diffie-Hellman primitive as specified in ANSI X9.63 and IEEE P1363 when they agree MAC and encryption keys. Devices and hosts will use NIST's 163-bit elliptic curve sect163k1 when they agree keys (see http://csrc.nist.gov/encryption for the NIST curves). All certificates will be signed using ECDSA as specified in ANSI X9.62 and IEEE P1363. The TA will use NIST's 283-bit elliptic curve sect283k1 when it signs certificates.

4.3 Stage 1: Certificate Exchange and Key Derivation

During certificate exchange, the host first generates a 128-bit random challenge R_{Host} and sends it to the device along with its certificate $Cert_{TA}(K_{PubHost})$, as the parameters to the *PUT_CMD* request. In response to this request, the device validates the host's certificate and optionally checks that the host's certificate has not been revoked.

Next, the host issues a *GET_RESP* request. This causes the device to generate a 128-bit random challenge R_{Device}, and to derive keys as detailed in 0 below. The device sends the challenge R_{Device} to the host along with its certificate $Cert_{TA}(K_{PubDevice})$ in the response. Upon receipt of this response, the host validates the device's certificate and optionally checks that the device's certificate has not been revoked. Next, it derives keys as detailed in 0 below

PUT_CMD	0x01, 16, R_{Host}, $\|Cert_{TA}(K_{PubHost})\|$, $Cert_{TA}(K_{PubHost})$
GET_RESP	0x01, 16, R_{Device}, $\|Cert_{TA}(K_{PubDevice})\|$, $Cert_{TA}(K_{PubDevice})$

Table 9. Parameters to Requests in Enhanced ECCPP, Certificate Exchange and Key Derivation Stage

4.3.1 Key Derivation

At the end of certificate exchange, both the host and the device generate session keys consisting of MAC keys $Kforward_0$ and $Kbackward_0$ and encryption keys $Kforward_i$ and $Kbackward_i$ for i between 1 and n. These keys are derived from the Diffie-Hellman shared secret Z (which is the x-coordinate of the point $hK_{PrvDevice}K_{PubHost} = hK_{PrvHost}K_{PubDevice}$) using the hash function SHA-1 as follows:

$$Kforward_i = H(Z,i,0,R_{Host},R_{Device},K_{PubHost},K_{PubDevice})$$

$$Kbackward_i = H(Z,i,1,R_{Host},R_{Device},K_{PubHost},K_{PubDevice})$$

Forward keys will be used to secure messages sent from the host to the device, and backwards keys will be used to secure messages sent from the device to the host. The certificate exchange stage and the key derivation phase represent an execution of the static ECDH protocol as specified in ANSI X9.63 and IEEE P1363 by the host and the device.

4.4 Stage 2: MAC Exchange

Subsequent to key derivation, the host and device may exchange MACs as follows if they want to achieve mutual explicit authentication (as well as mutual implicit authentication). The host first generates a MAC on a message containing the flow number 1, R_{Host}, R_{Device}, and the identities of the parties using the MAC scheme HMAC-with-SHA-1 under the key *Kforward*0, and sends the MAC to the device as the parameter to a *PUT_CMD* request. The device verifies if the MAC it received is valid.

Next, the host issues a *GET_RESP* request. This causes the device to generate a MAC on a message containing the flow number 2, R_{Host}, R_{Device}, and the identities of the parties using the MAC scheme HMAC-with-SHA-1 under the key *Kbackward*0. The device returns the MAC to the host in the response. Finally the host verifies whether the MAC it received is valid.

PUT_CMD	0x02, 20, *forward MAC*
GET_RESP	0x02, 20, *backward MAC*

Table 10. Parameters to Requests in Enhanced ECCPP, MAC Exchange Stage

4.5 Stage 3: Content Exchange

Subsequent to key derivation, the host and device are ready to exchange content. Initially the host sends content to the device encrypted using the block cipher DES or triple DES in CBC mode under the key *Kforward*1, and the device sends content to the host encrypted under the key *Kback*1. At any time the host or the device may tell the other party to update the keys they are using – this causes the parties, who were previously encrypting content using $Kforward_i$ and $Kbackward_i$, to begin encrypting content using $Kforward_{i+1}$ and $Kbackward_{i+1}$. The mechanism chosen for this is inclusion of the value of *i* used for the current block of content in the header of the content. For subsequent blocks, the same or a higgher value of *i* must be used (in both directions). If a value *i* smaller than the last value used is encountered, this is a fatal error. This key update procedure limits the exposure of individual keys.

Byte Index	Value
0	Index *i* of the encryption key used for content encryption
1	First byte of (encrypted) content
…	…
n	Last (*n*th) byte of (encrypted) content

Table 11. Format of encrypted payload