

Implementing DICE with the Current CEC1702

We would like to work with Microsoft to implement a DICE prototype using the current CEC1702, without DICE code in ROM. This will allow us to determine if the security needs can be met with the current part.

In the existing CEC1702 the MCHP ECC Private Key can optionally be encrypted and stored in eFuse and then locked, which prevents access until the next power-on.

The DICE code can be signed and encrypted - the CEC1702 can authenticate as well as decrypt the code before execution.

A secret key can be stored in the user area of eFuse and then locked, specifically the DICE Encrypted User Key, the unique device identity used by the DICE code.

The CEC1702 provides APIs that the DICE code can use to load the application code, as well as authenticate and decrypt this code.

The following is a list of the keys to be used in this implementation.

Key	Location	Description	Protection
Customer Authentication Key	eFuse	This public key is used to perform ECDSA verification of the Binary Image	Locked by boot ROM code before passing control to DICE Code.
MCHP AES Symmetric Key	ROM	MCHP AES key to decrypt the MCHP ECC Private Key in eFuse	Locked by boot ROM code before passing control to DICE Code.
MCHP ECC Private Key	eFuse	Private Key used for Elliptic Curve Diffie-Hellman Key Exchange (ECDH). This is part of the Private/Public key pair used for ECDH. It is used in decrypting the Binary Image. It can be stored in encrypted form.	Locked by boot ROM code before passing control to DICE Code.
DICE Encrypted User Key	eFuse user region	This is the unique device identity used by the DICE code.	Locked by DICE code before passing control to application code.
DICE Decryption Key	DICE code	This is the key to decrypt the DICE Encrypted User Key in the eFuse.	Deleted by DICE code before passing control to application code.
Optional keys for authenticating/decrypting the application code	Option A: DICE code	Optional keys for authenticating/decrypting the application code	Deleted by DICE code before passing control to application code.
	Option B: eFuse user region	Optional keys for authenticating/decrypting the application code	Locked by DICE code before passing control to application code.

Outline of the authentication and decryption process of CEC1702

- Authenticate the header of the Binary Image from the flash:
 - The ROM calculates the SHA-256 hash of the first 64 bytes of the header.
 - The ROM performs ECDSA signature verification on the calculated hash digest, using the Customer Authentication Key (public key) stored in the eFuse (see Note 1).
 - If the ECDSA verification procedure returns an error, the load terminates.

Note 1: Header is signed by customer and public key is loaded into eFuse
- Authenticate the Binary Image from the flash:
 - The ROM calculates the SHA-256 hash of the image.
 - The ROM performs ECDSA signature verification on the calculated hash digest, using the Customer Authentication Key (public key) stored in the eFuse (see Note 2).

- If the ECDSA verification procedure returns an error, the load terminates.
- Note 2: Binary Image is signed by customer and public key is loaded into eFuse

3. Decrypt the Binary Image:

After the image is authenticated, if it is encrypted, then the ROM decrypts the image as follows.

- Obtain the MCHP ECC Private Key s from eFuse. If the private key is encrypted, derive s by AES256-CBC decryption using the 256-bit MCHP AES Symmetric Key stored in the secure region of ROM.
- Extract the point $R=[R_x,R_y]$ from Key Header (R is customer's ECDH public key)
- Check the point R to verify that it is on the NIST standard P-256 curve.
- Perform ECC point-multiply $s.R=K$, where $K=[K_x,K_y]$ on the curve P-256.
- Generate a 512-bit block to be hashed from the 256-bit K_x .
- Hash the 512-bit block using SHA-256. This creates a 256-bit digest that is used as the AES key.
- Generate another 512-bit block to be hashed from the 256-bit K_x .
- Hash the 512-bit block using SHA-256. This creates a 256-bit digest: the least significant 128 bits are used as the Initialization Vector for the encryption.

Decryption is AES256-CBC, using the AES key and Initialization Vector just calculated.

Options for DICE Implementation

Option 1: 2 stage loader. See concept diagram. This is the recommended solution.

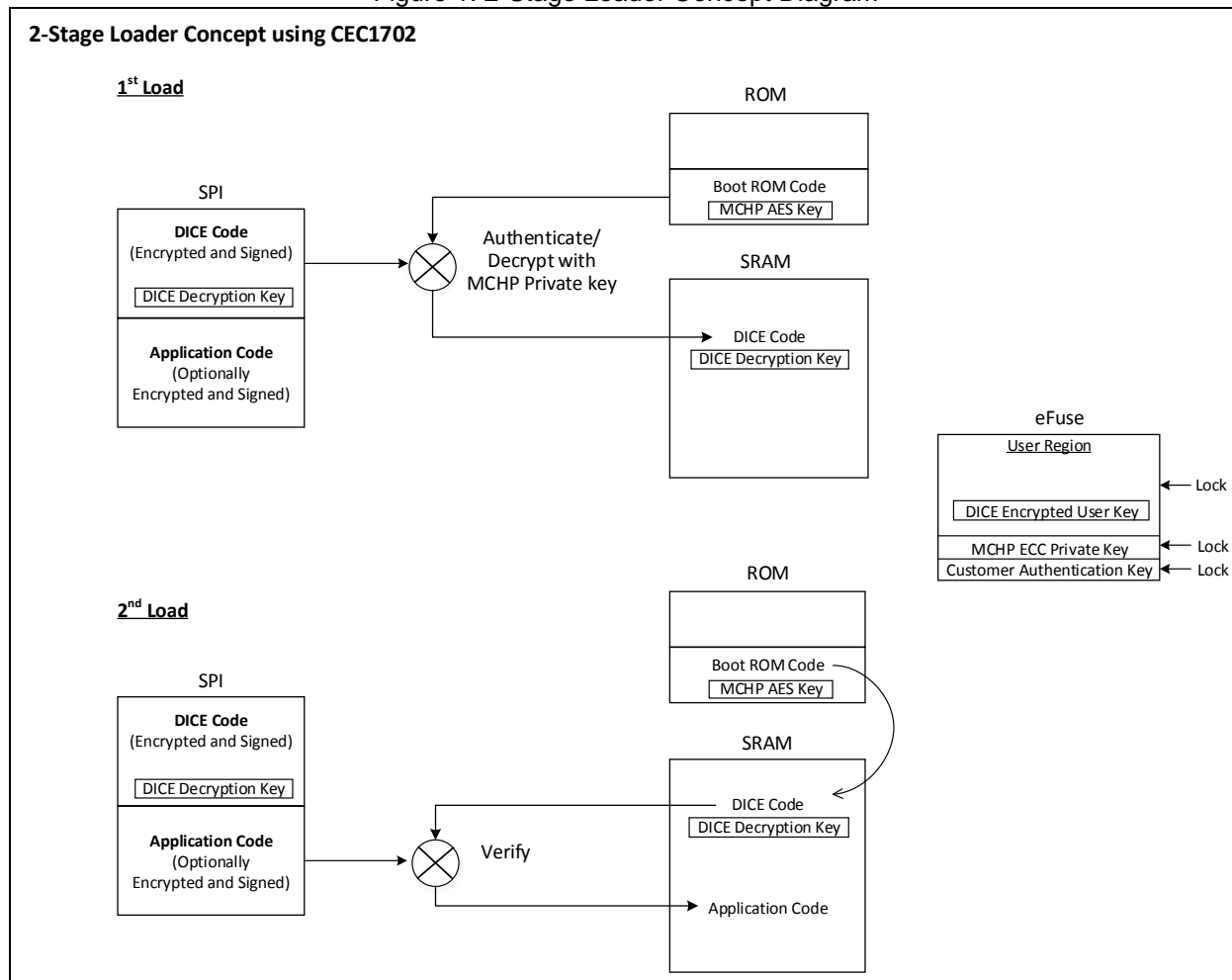
First the boot ROM loads the DICE code from the SPI flash into SRAM. This code should be encrypted and signed and the boot ROM will authenticate and decrypt it.

The boot ROM code then turns over control to the DICE code. Before doing so, the boot ROM code will lock the MCHP AES Symmetric Key in ROM, and the Customer Authentication Key and MCHP ECC Private Key in eFuse.

After the boot ROM gives control over to the DICE code, the DICE code uses the provided APIs to load the application code. The DICE code can optionally authenticate and decrypt the application code using APIs that are provided for the CEC1702. The DICE code contains the DICE Decryption Key to decrypt the DICE Encrypted User Key in the eFuse. This DICE Encrypted User Key is the unique device identity used by the DICE code.

The DICE code then turns over control to the application code. Before doing so, the DICE code will lock the DICE Encrypted User Key in eFuse by locking the eFuse user region, and it will delete/overwrite any keys it has in SRAM so the application code can't access them.

Figure 1: 2-Stage Loader Concept Diagram



Option 2: Load all code (DICE and Application) at once.

The boot ROM loads the DICE code and application code all at once from the SPI flash into SRAM. This code should be encrypted and signed and the boot ROM will authenticate and decrypt it.

The boot ROM code then turns over control to the DICE code. Before doing so, the boot ROM code will lock the MCHP AES Symmetric Key in ROM, and the Customer Authentication Key and MCHP ECC Private Key in eFuse.

The DICE code contains the DICE Decryption Key to decrypt the DICE Encrypted User Key in the eFuse. This DICE Encrypted User Key is the unique device identity used by the DICE code.

The DICE code then turns over control to the application code. Before doing so, the DICE code will lock the DICE Encrypted User Key in eFuse by locking the eFuse user region, and it will delete/overwrite any keys it has in SRAM so the application code can't access them.

Future Discussion

If full DICE support can be implemented in the CEC1702 with a ROM update, this is something we will consider.

- We can include the DICE code in the boot ROM and all the keys will be encrypted.
- The ROM will lock the keys and delete/overwrite any keys in SRAM before transitioning to the application code.