ModusToolbox™ signer/combiner user guide

ModusToolbox™ tools package

About this document

A newer version of this document may be available on the web here

Scope and purpose

The ModusToolbox™ Signer/Combiner is a feature that brings a new experience in signing and manipulating binary and IntelHex files. Describe the instructions (commands) in the JSON file and run a single CLI command to execute the instructions.

Table of contents

1 Run config

Executes commands specified in the JSON file.

Command

```
run-config
```

Parameters

Name	Optional/Required	Description
-i, --input	required	Path to the JSON file containing commands.
-s, --set	optional	Value for variable interpolation. Must be specified as `[variable] [value]`. Every `[variable]` in the provided JSON file will be interpolated with `[value]`. The parameter may be specified more than once. Only alphanumeric and underscore characters may be in the `[variable]` name.
--symbol	optional	Path to a symbol file containing variable definitions. May be specified multiple times. Each file should provide a mapping of variable names to values.
--symbol-search	optional	Path to a directory to search for symbol files. May be specified multiple times. All valid symbol files found in the specified directories will be used for variable interpolation.

Usage example

```
# Run config from file commands.json
$ edgeprotecttools run-config --input commands.json

# Run config from file commands.json and set variables to interpolate
$ edgeprotecttools run-config --input commands.json --set VARIABLE_1 0x12345678 --set
VARIABLE_2 "hello, world" --set VARIABLE_3 "42"
```

2 Variable interpolation

Variable interpolation is a technique used to dynamically insert the value of variables into strings or text.

The variable name in the JSON file must adhere to the string format, enclosed by double curly braces, and consist solely of alphanumeric and underscore characters (for example, `"{{ VARIABLE_1 }}"`).

```
{
    "address": "{{ VARIABLE_1 }}"
}
```

Setting variables via CLI

Variables can be set directly from the command line using the `--set` option. The variable name in the CLI command must correspond to the variable name specified in the JSON file. The value may contain any printable characters; however, the usage of quotes should be cautious.

```
$ edgeprotecttools run-config --input commands.json --set VARIABLE_1 0x12345678
```

The interpolation will result in the following:

```
{
    "address": "0x12345678"
}
```

***Note**: The variable interpolation does not modify the input JSON file. It will interpolate the variables only in the runtime.*

Using symbol files for variable interpolation

In addition to the `--set` option, variables can be provided via symbol files using the `--symbol` and `--symbol-search` options:

- `--symbol <file>`: Specify one or more symbol files containing variable definitions. Each file should provide a mapping of variable names to values.
- `--symbol-search <directory>`: Specify one or more directories to search for symbol files. All valid symbol files found in the specified directories will be used for variable interpolation.

You can specify one or more symbol files using the `--symbol` option, or provide a directory containing multiple symbol files using the `--symbol-search` option. All variables from all discovered symbol files will be used for interpolation, provided their names are unique across all sources.

Symbol file structure

A symbol file is a JSON file that defines a mapping of variable names to their values. Each key in the file represents a variable name, and its value is the string to be interpolated. Variable names must consist only of alphanumeric characters and underscores.

Example symbol file:

```
{
    "version":1,
    "symbolInformation":[
        {
            "name":"CYMEM_CM33_0_S_extended_boot_reserved_START",
            "value":"0x22000000"
        },
        {
            "name":"CYMEM_CM33_0_S_extended_boot_reserved_C_START",
            "value":"0x02000000"
        }
    ]
}
```

Example usage:

```
# Set variables from a symbol file
$ edgeprotecttools run-config --input commands.json --symbol symbols.json

# Set variables from all symbol files in a directory
$ edgeprotecttools run-config --input commands.json --symbol-search ./symbols_dir
```

Combining variable sources

You may use `--set`, `--symbol`, and `--symbol-search` options simultaneously. All variables from these sources are merged for interpolation. Variable names must be unique across all sources.

This flexible approach allows you to manage variable interpolation efficiently, whether you prefer to define variables inline, in files, or by searching directories for symbol definitions.

3 JSON description

The commands for Run config command are accepted through JSON file interface. These include:

- Hierarchy
- JSON File
- Command Group
- Command
- Generating extra launch configurations

3.1 Hierarchy

JSON file introduces several layers of abstractions that serve different purposes.

Abstraction	Description
JSON File	Highest level of abstraction. Depicts a superset of use-cases.
Command Group	Use-case level. Depicts a series of commands that is used to achieve one outcome.
Command	Lowest level of abstraction. Depicts a single command to execute.

3.2 JSON File

Structure

```
{
    "schema-version": 1.0,
    "content": [
        // add command groups here
    ]
}
```

Properties

Name	Type	Optional/Required	Description
schema-version	number	required	Version of syntax used in JSON file.
content	list	required	List of command group objects.

3.3 Command Group

Structure

```
{
    "name": "Command Group Name",                // Command Group name
    "description": "Command Group Description", // Command Group description
    "enabled": true,                             // Enable or Disable this Command Group
    "commands": [
        // add commands here
    ]
}
```

Properties

Name	Type	Optional/Required	Description
name	string	required	Command Group name. Used for readability and debugging purposes.
description	string	optional	Command Group description. Used for readability purposes only.
enabled	boolean	optional	Enabling Command Group can be enabled or disabled. Disabled command group will not be executed. Default: false.
commands	list	required	List of command objects.

3.4 Command

Structure

```
{
    "command": "extract-payload",                // Command name
    "inputs": [                                  // Command inputs
        {
            "description": "Input description",
            "file": "input.hex"
        }
    ],
    "outputs": [                                 // Command outputs
        {
            "description": "Output description",
            "file": "output.bin"
        }
    ]
}
```

3 JSON description

Properties

Name	Type	Optional/Required	Description
command	string	required	Command name. Available values: add-signature, custom-script, extract-payload, hex-segment, merge, shift and sign.
inputs	list	required	Command inputs. Different commands may have different requirements for this field.
outputs	list	optional	Command outputs. Different commands may have different requirements for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Shift command example",
            "description": "Shifts a hex segment to a secondary address",
            "enabled": true,
            "commands": [
                {
                    "command": "shift",
                    "inputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application",
                            "file": "rram_3.hex",
                            "address" : "0x32030000"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application shifted
to secure region",
                            "address": "0x3202A000",
                            "file": "shifted_rram_3.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

3.5 Generating extra launch configurations

When you create a new ModusToolbox™ project, it contains set of common launch configurations. When customizing output hex files, you may need to generate additional Eclipse or VS Code Debug launch configurations for the specific hex files. To do this, insert a special "extra_config" section into the json file. Then, regenerate the launch configurations. For Eclipse, click the "Generate Launches for ..." link in the Quick Panel, or run "make eclipse" in a bash terminal (modus-shell for Windows). For VS Code, run "make vscode".

Extra_config properties

Name	Type	Optional / Required	Description
project	string	required	Project name that requires debugging. (This is used to determine the main elf file. The debug session will start from the beginning of main() defined in this elf). The launch config will be bound to this project.
debug_config_name	string	required	Specifies the output launch config name in the form <debug_config_name> Debug <probe>.
build_dependency	string	optional	Specifies whether the generated launch config should build a particular project or the entire application. Default is "project". Valid values are "project" or "application".
default	boolean	optional	If set to true, this hex file will be used for make program and the Program Application config. Default is "false".
extra_elf_path	string	optional	Allows you to set additional elf file(s) to be used for debugging. It requires the path to the extra elf file relative to the Application-level directory. Supported for TrustZone-enabled devices.

3 JSON description

Extra_config example

```
{
  "schema-version": 1.0,
  "content": [
    {
      "name": "Shift command example",
      "description": "Shifts a hex segment to a secondary address",
      "enabled": true,
      "commands": [
        {
          "command": "shift",
          "inputs": [
            {
              "description": "CM33 Secure + CM33 Non-Secure application",
              "file": "rram_3.hex",
              "address" : "0x32030000"
            }
          ],
          "outputs": [
            {
              "description": "CM33 Secure + CM33 Non-Secure application shifted to secure
region",
              "address": "0x3202A000",
              "file": "shifted_rram_3.hex"
            }
          ],
          "extra_config": [
            {
              "project": "proj_cm0p",
              "debug_config_name": "Debug_proj_cm0p",
              "build_dependency": "application",
              "default": true,
              "extra_elf_path": ["proj_cm33_ns/build/$(TARGET) /Debug/proj_cm33_ns.elf"]
            }
          ]
        }
      ]
    }
  ]
}
```

4 Commands description

This section contains descriptions for the following commands:

- Shift
- Hex relocate
- Hex segment
- Merge
- Sign image
- Extract payload
- Add signature
- Custom script
- Bin dump
- Hex dump
- Bin to hex
- Hex to bin
- Hash

4.1 Shift

Shift segment to a specific address.

Inputs

Name	Type	Optional/Required	Description
file	string	required	Input hex file containing the segment to shift.
address	string	optional	Address of the segment that has to be shifted.
description	string	optional	Description for this field.

Outputs

Name	Type	Optional/Required	Description
file	string	required	Path to the output file.
address	string	optional	The new address of the segment.
description	string	optional	Description for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Shift command example",
            "description": "Shifts a hex segment to a secondary address",
            "enabled": true,
            "commands": [
                {
                    "command": "shift",
                    "inputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application",
                            "file": "rram_3.hex",
                            "address" : "0x32030000"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application shifted
to secure region",
                            "address": "0x3202A000",
                            "file": "shifted_rram_3.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.2 Hex relocate

Relocates regions in the hex file to new address spaces. Relocates regions in the hex file to new address spaces. All segments within the specified region (from `start` to `start` + `size`) will be relocated to the new address (`dest`).

Inputs:

Name	Type	Optional/Required	Description
file	string	required	Input hex file containing the segment(s) to relocate.
regions	array	required	Regions to relocate. An array of objects, each containing `start`, `size`, and `dest` fields.
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	Path to the output hex file.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "hex-relocate command example",
            "description": "Relocates regions in hex file to new address spaces",
            "enabled": true,
            "commands": [
                {
                    "command": "hex-relocate",
                    "inputs": [
                        {
                            "file": "input.hex",
                            "regions": [
                                {
                                    "start": "0x08000000",
                                    "size": "0x04000000",
                                    "dest": "0x60000000"
                                }
                            ]
                        }
                    ],
                    "outputs": [
                        {
                            "file": "output.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.3 Hex segment

Extracts a segment from the hex file.

Inputs

Name	Type	Optional/Required	Description
file	string	required	Path to the input hex file.
address	string	required	Address of the segment.
description	string	optional	Description for this field.

Outputs

Name	Type	Optional/Required	Description
file	string	required	Path to the file where to save the extracted segment.

Name	Type	Optional/Required	Description
description	string	optional	Description for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Shift command example",
            "description": "Shifts a hex segment to a secondary address",
            "enabled": true,
            "commands": [
                {
                    "command": "shift",
                    "inputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application",
                            "file": "rram_3.hex",
                            "address" : "0x32030000"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application shifted
to secure region",
                            "address": "0x3202A000",
                            "file": "shifted_rram_3.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.4 Merge

Inputs

This command requires two or more inputs of the same format.

Name	Type	Optional/Required	Description
file	string	required	Path to the file to merge.
description	string	optional	Description for this field.

Outputs

Name	Type	Optional/Required	Description
file	string	required	Path to the merged file.

Name	Type	Optional/Required	Description
format	string	optional	Format of the output file. Available values: ihex or bin.
address	string	optional	Start address of the output hex file.
description	string	optional	Description for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Merge command example",
            "description": "Merges two or more hex/bin files into a single file",
            "enabled": true,
            "commands": [
                {
                    "command": "merge",
                    "inputs": [
                        {
                            "description": "CM33 Secure  application",
                            "file": "./rram_1.hex"
                        },
                        {
                            "description": "CM33 Non-Secure application",
                            "file": "./rram_2.hex"
                        },
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application",
                            "file": "./rram_3.hex"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "CM33 Secure + CM33 Non-Secure application merged to
one file",
                            "format": "ihex",
                            "file": "./cmd_merge_3_files.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.5 Sign image

Signs the user application and converts it into MCUboot format. If the signing key is not provided, the result is the MCUboot formatted unsigned image. All the numeric values can be provided in a decimal (for example, `fill-value: 255`) or hexadecimal format (for example, `fill-value: "0xFF"`).

Inputs

Name	Type	Optional/Required	Description
file	string	required	Path to the file to be signed or converted into MCUboot format.
description	string	optional	Description for this field.
signing-key	string	optional	ECDSA or RSA private key used to sign the image.
header-size	string, integer	required	MCUboot header size.
slot-size	string, integer	required	Maximum slot size.
fill-value	string, integer	optional	Value read back from erased flash. Default: `0`. Available values: `0`, `0xFF`.
min-erase-size	string, integer	optional	Minimum erase size. Default: `0x8000`.
image-version	string	optional	Image version in the image header. Default: `0.0.0`.
security-counter	string, integer	optional	Value of security counter. Use the `auto` keyword to automatically generate it from the image version. Default: `auto`.
align	string, integer	optional	Flash alignment. Default: `8`. Available values: `1`, `2`, `4`, `8`.
pubkey-format	string	optional	Public key format in the image TLV: full key or hash of the key. Available values: `hash` or `full`. Default: `hash`.
pubkey-encoding	string	optional	Public key encoding in the image TLV. Applicable values: `der`, or `raw`. Default: `der`.
signature-encoding	string	optional	Image signature encoding. Applicable values: `asn1`, or `raw`. Default: `asn1`.
pad	boolean	optional	Adds padding to the image trailer. Pads the image from the end of the TLV area up to the slot size. boot_magic is always at the very end after the padding.
confirm	boolean	optional	Adds image OK status to the trailer. Pads the image from the end of the TLV area up to the slot size and sets the image OK byte to 0x01 (the eighth byte from the end). The padding is required for this feature and is always applied. boot_magic is always at the very end after the padding.
overwrite-only	boolean	optional	Use overwrite mode instead of swap.
boot-record	string	optional	Create CBOR encoded boot record TLV. Represents the role of the software component (for example, CoFM for coprocessor firmware). Maximum 12 characters.
hex-address	string, integer	optional	Adjust the address in the hex output file.

Name	Type	Optional/Required	Description
load-address	string, integer	optional	Load address for image when it should run from RAM.
rom-fixed	string, integer	optional	Set flash address the image is built for.
max-sectors	string, integer	optional	When padding allow for this amount of sectors. Default: `128`.
save-enctlv	boolean	optional	When upgrading, save encrypted key TLVs instead of plain keys. Enable when BOOT_SWAP_SAVE_ENCTLV config option was set.
dependencies	string	optional	Add dependency on another image. Format: `(<image_ID>,<image_version>), ...` .
encryption-public-key	string	optional	ECDSA public key used to generate the symmetric key for image encryption (ECIES schema). It must be the receiver's public key.
encryption-secret-key	string	optional	An encryption key.
encryption-key-role	string	optional	An encryption key role. Specifies what the key is used for: • Image Encryption: Encrypts images for decryption in the XIP mode • Key Wrapping: Acts as a Key Encryption Key (KEK) for AES Key Wrapping (AES-KW), ensuring secure key management. Available values: `XIP`, `AES-KW`. Default: `XIP`.
encryption-address	string	optional	Starting address for data encryption.
protected-tlv	list	optional	The custom TLV to be placed into a protected area (the signed part). Add the `0x` prefix for the value to be interpreted as an `integer`, otherwise it will be interpreted as a `string`.
tlv	list	optional	The custom TLV to be placed into a non-protected area. Add the `0x` prefix for the value to be interpreted as an `integer`, otherwise it will be interpreted as a `string`.
kdf	string	optional	Key derivation function name. Default: `HKDF`. Available values: `HKDF`, `KBKDFCMAC`.
remove-tlv	list	optional	Removes TLV with the specified ID.

Outputs

Name	Type	Optional/Required	Description
description	string	optional	Description for this field.
file	string	required	Path to the signed and/or converted to the MCUboot format image.
format	string	required	Format of the output file. Available values: `ihex` or `bin`.

Name	Type	Optional/Required	Description
unencrypted	string	optional	The path where to save unencrypted image payload (bin). Specify this option if the image is encrypted and provide the unencrypted image to HSM because the signature is calculated on the unencrypted data.
nonce-output	string	optional	The path to a file where to save the nonce.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name" : "sign command example",
            "description": "Signs the input file and converts it to the MCUboot format",
            "enabled": true,
            "commands": [
                {
                    "command": "sign",
                    "inputs": [
                        {
                            "description": "Secure application",
                            "file": "./rram_1.hex",
                            "signing-key": "./some-key-ec-p256.pem",
                            "overwrite-only": true,
                            "header-size": "0x400",
                            "slot-size": "0x1000",
                            "fill-value": "0x00",
                            "min-erase-size": "0x200",
                            "pad": true,
                            "hex-address": "0x32005000",
                            "protected-tlv": [
                                {
                                    "tag": "0x22",
                                    "value": "0x12345678"
                                }
                            ]
                        }
                    ],
                    "outputs" : [
                        {
                            "description": "Signed secured application",
                            "format" : "ihex",
                            "file": "./rram_1_signed.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.6 Extract payload

The MCUboot formatted image consists of a header, a payload, a protected TLV area, and a non-protected TLV area. A header, a payload, and a protected TLV are the data that must be signed. A non-protected TLV area must not be included in a signature. This command extracts a part of an image to be signed.

Inputs

Name	Type	Optional/Required	Description
file	string	required	Image with MCUboot metadata.
description	string	optional	Description for this field.

Outputs

Name	Type	Optional/Required	Description
file	string	required	Path where to save the image to be signed (bin).
description	string	optional	Description for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Extract-payload command example",
            "description": "Adds a signature to the MCUboot formatted image",
            "enabled": true,
            "commands": [
                {
                    "command": "extract-payload",
                    "inputs": [
                        {
                            "description": "Path to the input hex file",
                            "file": "./hexs/rram_1_meta.hex"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Name of output file",
                            "file": "./hexs/extracted.bin"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.7 Add signature

Adds signature to the existing MCUboot format image.

Command name

```
add-signature
```

Inputs

This command requires two input files, where one is the MCUboot formatted image and the second is the signature.

Name	Type	Optional/Required	Description
file	string	required	Path to the image with MCUboot metadata
description	string	optional	Description for this field.

Name	Type	Optional/Required	Description
file	string	required	Binary file containing signature.
algorithm	string	required	Signature algorithm. Available values: ECDSA-P256, RSA2048, RSA4096.
description	string	optional	Description for this field.

Outputs

Name	Type	Optional/Required	Description
file	string	required	Path to the output file.
format	string	required	Format of the output file. Available values: ihex, bin.
description	string	optional	Description for this field.

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Add-signature command example",
            "description": "Adds a signature to the MCUboot formatted image",
            "enabled": true,
            "commands": [
                {
                    "command": "add-signature",
                    "inputs": [
                        {
                            "description": "Unsigned bootloader in MCUboot format",
                            "file": "./hexs/bootloader_meta.hex"
                        },
                        {
                            "description": "Signature returned by HSM",
                            "file": "./hexs/signature.bin",
                            "algorithm": "ECDSA-P256"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Final image signed with HSM",
                            "format": "ihex",
                            "file": "./hexs/bootloader_signed.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.8 Custom script

Executes system commands or other programs.

Command name

```
custom-script
```

Inputs

Name	Type	Optional/Required	Description
command-line	string	required	The command line to be executed.
shell	boolean	optional	Specifies whether the command will be executed through the shell. On POSIX with shell=True, the shell defaults to /bin/sh. Invoking via the shell does allow you to expand environment variables and file globs according to the shell's usual mechanism. On Windows with shell=True, the COMSPEC environment variable specifies the default shell. The only time you need to specify shell=True on Windows is when the command you wish to execute is built into the shell (for example, **dir** or **copy**). You do not need shell=True to run a batch file or console-based executable.
description	string	optional	The command description.

Outputs

None

Example

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "Custom-script command example",
            "description": "Test template for the custom-script command",
            "enabled": true,
            "commands": [
                {
                    "command": "custom-script",
                    "inputs": [
                        {
                            "description": "List detailed information about files and
directories in the current directory",
                            "command-line": "ls -la",
                            "shell": false
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.9 Bin dump

Creates binary file from the hex string or random bytes.

Command name: `bin-dump`

Inputs:

Name	Type	Optional/Required	Description
data	string	optional	The hex string. Either `data` or `random` property must be specified.
random	string, integer	optional	Generate random binary of the specified length.
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	Path to the output bin file.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "bin-dump command example",
            "description": "Creates binary file",
            "enabled": true,
            "commands": [
                {
                    "command": "bin-dump",
                    "inputs": [
                        {
                            "description": "Hex string or random value",
                            "data": "00112233445566778899AABBCCDDEEFF",
                            "random": null
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Output file",
                            "file": "output.bin"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.10 Hex dump

Extracts data from the hex file and saves it to a binary file.

Command name: `hex-dump`

Inputs:

Name	Type	Optional/Required	Description
file	string	required	The path to the hex file.
address	string, integer	required	Address of the data in the hex file.
size	string, integer	required	Size of the data.
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	Path to the output bin file.

Name	Type	Optional/Required	Description
fill-value	string, integer	optional	Value to fill the spaces between the segments. Available values: `0x00-0xFF`. Default: `0`.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "hex-dump command example",
            "description": "Extracts data from the hex file",
            "enabled": true,
            "commands": [
                {
                    "command": "hex-dump",
                    "inputs": [
                        {
                            "description": "Input hex file",
                            "file": "input.hex",
                            "address": "0x00",
                            "size": "0xFF"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Output bin file",
                            "file": "output.bin",
                            "fill-value": 0
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.11 Bin to hex

Converts binary file to hex.

Command name: `bin2hex`

Inputs:

Name	Type	Optional/Required	Description
file	string	required	The path to the bin file.
offset	string, integer	optional	Starting address offset for loading bin.
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	Path to the output hex file.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "bin2hex command example",
            "description": "Converts binary image to hex",
            "enabled": true,
            "commands": [
                {
                    "command": "bin2hex",
                    "inputs": [
                        {
                            "description": "Input bin file",
                            "file": "input.bin",
                            "offset": 0
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Output hex file",
                            "file": "output.hex"
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.12 Hex to bin

Converts hex file to binary.

Command name: `hex2bin`

Inputs:

Name	Type	Optional/Required	Description
file	string	required	The path to the input hex file.
start	string, integer	optional	Start of address range.
end	string, integer	optional	End of address range.

Name	Type	Optional/Required	Description
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	The path to the output bin file.
size	string, integer	optional	Size of the resulting file in bytes.
fill-value	string, integer	optional	Value to fill the spaces between the segments. Available values: `0x00`-`0xFF`. Default: `0xFF`.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "hex2bin command example",
            "description": "Converts hex file to binary",
            "enabled": true,
            "commands": [
                {
                    "command": "hex2bin",
                    "inputs": [
                        {
                            "description": "Input hex file",
                            "file": "input.hex",
                            "start": "0x32000000",
                            "end": "0x32000200"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Output bin file",
                            "file": "output.bin",
                            "size": "0x400",
                            "fill-value": 0
                        }
                    ]
                }
            ]
        }
    ]
}
```

4.13 Hash

Calculates hash of a file.

4 Commands description

Command name: `hash`

Inputs:

Name	Type	Optional/Required	Description
file	string	required	The path to the input file.
algorithm	string	required	Hash algorithm.
description	string	optional	Description for this field.

Outputs:

Name	Type	Optional/Required	Description
file	string	required	Output binary or text file.
format	string	required	Format of the output file. Available values: `bin`, `txt`. Default: `bin`.
description	string	optional	Description for this field.

Example:

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name": "hash command example",
            "description": "Calculates hash of a file",
            "enabled": true,
            "commands": [
                {
                    "command": "hash",
                    "inputs": [
                        {
                            "description": "Calculate hash of binary file",
                            "file": "output.bin",
                            "algorithm": "SHA256"
                        }
                    ],
                    "outputs": [
                        {
                            "description": "Output binary or text file",
                            "file": "hash.txt",
                            "format": "txt"
                        }
                    ]
                }
            ]
        }
    ]
}
```

5 JSON comments

Even though JSON standard does not support comments, this JSON parser is able to handle comments akin to those found in C programming language. Therefore, users can include comments for better documentation and readability without affecting the JSON's structure or the data's integrity.

The parser can recognize and properly ignore both `// single-line comment` and `/* multi-line comment */` comments, allowing users to annotate their JSON files just like they would in a C source file.

```
{
    // this is a single-line comment
    "address": "0x12345678",
    /*
      this
      is
      a
      multi-line
      comment
    */
    "file": "hello.world"
}
```

6 Use cases

This section contains the following use cases:

- Use case 1: Shifting non-secure app to secure region and signing
- Use case 2: Signing two images located in the internal flash and in the external flash
- Use case 3: Signing hex containing EEPROM segment
- Use case 4: Signing encrypted image with HSM

6.1 Use case 1: Shifting non-secure app to secure region and signing

There are two segments in the image. The first one is located in the secure memory region. The second one in the non-secure. The following JSON shows how to shift data from the non-secure memory region to secure memory region.

```
{   "schema-version": 1.0,
    "content": [
        {
            "name" : "CM33_Secure_CM33_Non_Secure",
            "description": "Use case 1: Shifting non-secure app to secure region and signing",
            "enabled": true,
            "commands": [
                {   "command": "shift",
                    "inputs": [
                        {   "description": "CM33 Secure + CM33 Non-Secure application",
                            "file": "app-path/cm33s_cm33ns.hex",
                            "address" : "0x2201F000"
                        }
                    ],
                    "outputs" : [
                        {   "description": "Application shifted to secure region",
                            "address" : "0x3201F000",
                            "file" : "out-path/cm33s_cm33ns_shifted.hex"
                        }
                    ]
                },
                {
                    "command": "sign",
                    "inputs": [
                        {   "description": "Shifted CM33 Secure + CM33 Non-Secure application",
                            "file": "out-path/cm33s_cm33ns_shifted.hex"
                        }
                    ],
                    "outputs" : [
                        {   "description": "Signed CM33 Secure + CM33 Non-Secure",
                            "signer-key" : "/key-path/signing_private_key.pem",
                            "format" : "ihex",
                            "file" : "out-path/cm33s_cm33ns_signed.hex",
                            "header-size": "0x400",
                            "fill-value": "0x00"
                        }
                    ]
                }
            ]
        }
    ]
}
```

6.2 Use case 2: Signing two images located in the internal flash and in the external flash

This use case shows how to sign both non-contiguous segments. The first one is located in the internal memory, and the second one in the external memory. The signing result may be a very large file because the

data are converted to a binary format. Following JSON shows how to split the segments, sign each of them, and then combine back to a single file.

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name" : "Signed Internal and External Flash Applications",
            "description": "Use case 2: Signing two images located in the internal flash and in
the external flash",
            "enabled": true,
            "commands": [
                {   "command": "hex-segment",
                    "inputs": [
                        {   "description": "Extract Internal Flash segment",
                            "file": "app-path/ble_app_psoc64.hex",
                            "address" : "0x10000000"
                        }
                    ],
                    "outputs" : [
                        {   "description": "Save Flash segment",
                            "format" : "ihex",
                            "file" : "out-path/internal.hex"
                        }
                    ]
                },
                {
                    "command": "hex-segment",
                    "inputs": [
                        {   "description": "Extract External Flash segment",
                            "file": "app-path/ble_app_psoc64.hex",
                            "address" : "0x18000000"
                        }
                    ],
                    "outputs" : [
                        {   "description": "Save EEPROM segment",
                            "format" : "ihex",
                            "file" : "out-path/external.hex"
                        }
                    ]
                },                {
                    "command": "sign",
                    "inputs": [
                        {   "description": "Signs Internal Flash Application",
                            "file": "out-path/internal.hex"
                        }
                    ],
                    "outputs" : [
                        {   "description": "Signed Internal Flash Application",
                            "signer-key" : "/key-path/signing_private_key.pem",
                            "format" : "ihex",
                            "file" : "out-path/internal_signed.hex"
                        }
```

```
                ]
            },
            {
                "command": "sign",
                "inputs": [
                    {   "description": "Signs External Flash Application",
                        "file": "out-path/external.hex"
                    }
                ],
                "outputs" : [
                    {   "description": "Signed External Flash Application",
                        "signer-key" : "/key-path/signing_private_key.pem",
                        "format" : "ihex",
                        "file" : "out-path/external_signed.hex"
                    }
                ]
            },
            {
                "command": "merge",
                "inputs": [
                    {   "description": "Signed Internal Flash Application",
                        "file": "out-path/internal_signed.hex"
                    },
                    {   "description": "Signed External Flash Application",
                        "file": "out-path/external_signed.hex"
                    }
                ],
                "outputs" : [
                    {   "description": "Combine signed Internal & External Flash Apps",
                        "format" : "ihex",
                        "file" : "out-path/ble_app_psoc64_signed.hex"
                    }
                ]
            }
        ]
    }
]
}
```

6.3 Use case 3: Signing hex containing EEPROM segment

There are two segments: the internal flash segment and the EEPROM segment. We need to sign the internal flash application only.

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name" : "EEPROM",
            "description": "Use case 2: Signing hex containing EEPROM segment",
            "enabled": true,
            "commands": [
                {
                    "command": "hex-segment",
                    "inputs": [
                        {
                            "description": "Extract Flash segment",
                            "file": "app-path/ble_app_psoc64.hex",
                            "address" : "0x10000000"
                        }
                    ],
                    "outputs" : [
                        {
                            "description": "Save Flash segment",
                            "format" : "ihex",
                            "file" : "out-path/flash.hex"
                        }
                    ]
                },
                {
                    "command": "hex-segment",
                    "inputs": [
                        {
                            "description": "Extract EEPROM segment",
                            "file": "app-path/ble_app_psoc64.hex",
                            "address" : "0x14000000"
                        }
                    ],
                    "outputs" : [
                        {
                            "description": "Save EEPROM segment",
                            "format" : "ihex",
                            "file" : "out-path/eeprom.hex"
                        }
                    ]
                },
                {
                    "command": "sign",
                    "inputs": [
                        {
                            "description": "Signs Flash application",
                            "file": "out-path/flash.hex"
```

```
                    }
                ],
                "outputs" : [
                    {
                        "description": "Signed Flash application",
                        "signer-key" : "/key-path/signing_private_key.pem",
                        "format" : "ihex",
                        "file" : "out-path/flash_signed.hex"
                    }
                ]
            },
            {
                "command": "merge",
                "inputs": [
                    {
                        "description": "Signed Flash segment",
                        "file": "out-path/flash_signed.hex"
                    },
                    {
                        "description": "Unsigned EEPROM segment",
                        "file": "out-path/eeprom.hex"
                    }
                ],
                "outputs" : [
                    {
                        "description": "Combined signed Flash app and unsigned EEPROM",
                        "format" : "ihex",
                        "file" : "out-path/ble_app_psoc64_signed.hex"
                    }
                ]
            }
        ]
    }
]
}
```

6.4 Use case 4: Signing encrypted image with HSM

In this case, we are creating an unsigned encrypted image, then sign it with HSM. To create an unsigned image use the sign command, but without the signing-key property. The result of the Sign image command are two unsigned images in the MCUboot format - encrypted and decrypted. The encrypted one is the image we are going to attach the signature generated by HSM. The decrypted one is the image we have to provide to HSM for signing. The signature is calculated from the non-encrypted data.

- The Extract payload command extracts from the decrypted image the part to be signed (header, body, protected TLV).

- The Custom script command signs the payload with HSM and saves the signature to the signature.bin file.
- The Add signature command adds the signature returned by HSM to the encrypted MCUboot format image.

```
{
    "schema-version": 1.0,
    "content": [
        {
            "name" : "BootloaderHSM",
            "description": "Use case 3: Signing image with HSM",
            "enabled": true,
            "commands": [
                {
                    "command": "sign",
                    "inputs": [
                        {
                            "description": "Path to the input hex file",
                            "file": "bootloader_path/bootloader.hex"
                        }
                    ],
                    "outputs" : [
                        {
                            "description": "Save encrypted bootloader and binary payload to
sign on HSM",
                            "header-size": "0x400",
                            "fill-value": "0x00",
                            "slot-size": "0x20000",
                            "pad": true,
                            "encryption-key": "key-path/public_key.pem",
                            "format" : "ihex",
                            "file" : "out-path/encryptedBoot.hex",
                            "decrypted" : "out-path/decryptedBoot.bin"
                        }
                    ]
                },                {
                    "command": "custom-script",
                    "inputs": [
                        {
                            "description": "Signing with HSM command. The command does not have
the 'outputs' property. If necessary, the outputs are handled by the command line.",
                            "command-line": "cxitool.exe Dev=3001@127.0.0.1
LogonPass=USR_0000,2222 Group=SLOT_0000 Spec=2 InFile=decryptedBoot.bin
Signature=signature.bin,raw Sign=SHA256,on_hsm,PSS"
                        }
                    ]
                },
                {
                    "command": "add-signature",
                    "inputs": [
                        {
                            "description": "Encrypted unsigned bootloader",
                            "file": "out-path/encryptedBoot.hex"
                        },
                        {
```

```
                        "description": "Signature returned by HSM",
                        "file": "signature.bin"
                    }
                ],
                "outputs" : [
                    {
                        "description": "Final image signed with HSM",
                        "format" : "ihex",
                        "file" : "out-path/boot_encrypted_signed.hex"
                    }
                ]
            }
        ]
    }
]
}
```

Revision history

Document revision	Date	Description of changes
**	2024-10-02	New document.
*A	2025-03-21	Updated section 3.5 with details to generate launch configs.
*B	2025-09-05	Updated "Sign image" section.
*C	2025-12-10	Minor back-end updates.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2025-12-10

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ymk1749831010551

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.