AN1509: Series 3 AXiP

Silicon Labs® Series 3 platform introduces new Authenticated eXecute in Place (AXiP) and Encrypted eXecute in Place (EXiP) features for protection of memory contents. While devices with in-package flash are delivered secure by default with AXiP enabled on two pre-defined regions, developers are able to configure up to eight code regions of customizable sizes with AXiP, EXiP, or no protection (code is stored in plaintext) through SE Manager APIs or DCI commands, allowing for flexibility in device setup. This application note will further discuss the importance of encrypted and authenticated memory contents, the fundamentals of AXiP and EXiP features, and will provide examples for customizing this feature.

KEY POINTS

- AXiP and EXiP overview
- Configuring code regions
- Development device considerations

Table of Contents

1. Series 3 Security Features 3

2. Introduction 4

3. Authenticated eXecute in Place 5

3.1 Algorithm Used 5

3.2 AXiP Key 5

3.3 IV Generation 5

3.4 AXiP Default Device Configuration 6

4. Encrypted eXecute in Place 7

4.1 Algorithm Used 7

4.2 EXiP Key 7

4.3 IV Generation 7

4.4 EXiP Device Configuration 7

5. Configuring Code Regions 8

5.1 Data Regions versus Code Regions 8

5.2 Physical versus Logical Mapping 9

5.3 Code Region Configuration File 10

5.4 Closing a Code Region 10

6. Considerations for Development Devices 11

6.1 Reading SE OTP Count 11

6.2 Skip Closing a Code Region 11

6.3 Transition to Development Device Command 12

7. Examples 13

7.1 Using AXiP Out of the Box 13

7.2 Custom Code Region Configuration 14

7.3 Closing a Code Region 15

8. Revision History 16

1. Series 3 Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Secure Engine on Series 2 products. Series 3 products continue to include and expand upon existing Secure Engine Technology The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys and to execute cryptographic functions and secure services.

User Assistance

In support of these products Silicon Labs offers the following application notes:

Document	Summary	Applicability
AN1190: Series 2 Secure Debug	How to lock and unlock Series 2 debug access, including background information about the SE	Secure Vault Mid and High
AN1218: Series 2 Secure Boot with RTSL	Describes the secure boot process on Series 2 devices using SE	Secure Vault Mid and High
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using SE during device production	Secure Vault Mid and High
AN1247: Anti-Tamper Protection Configuration and Use (this document)	How to program, provision, and configure the anti-tamper module	Secure Vault High
AN1268: Authenticating Silicon Labs Devices using Device Certificates	How to authenticate a device using secure device certificates and signatures, at any time during the life of the product	Secure Vault High
AN1271: Secure Key Storage	How to securely "wrap" keys so they can be stored in non- volatile storage.	Secure Vault High

Key Reference

Silicon Labs security implementations use asymmetric key pairs and symmetric keys. The table below clarifies key names, applicability, and relevant documentation.

Key Name	Customer Programmed	Purpose	Used in
Public Sign key	Yes	Secure Boot binary authentication and/or OTA upgrade payload authentication	AN1218 (primary), AN1222
Public Command key	Yes	Secure Debug Unlock or Disable Tamper command authentication	AN1190 (primary), AN1222, AN1247
OTA Decryption key (or GBL Decryption key)	Yes	Decrypting GBL payloads used for firmware up- grades	AN1222 (primary), UG266/UG489
Attestation key aka Private Device Key	No	Device authentication for secure identity	AN1268
AXiP Key	No	Used on Series 3 devices for encryption/decryption and authentication of firmware placed in flash	AN1509
EXiP Key	No	Used on Series 3 devices for encryption/decryption of firmware placed in flash	AN1509

2. Introduction

As embedded devices are increasingly used in critical applications such as healthcare devices, industrial controls, and IoT devices, security of these devices is paramount. Embedded devices in these systems often perform essential functions or contain sensitive data, making them targets for attacks. Silicon Labs introduced the Secure Engine as part of the new Series 2 portfolio, which introduced many new security features into the device offerings. Series 3 devices expand upon those offerings by including and improving previous security features, and introducing new security features, such as Authenticated eXecute in Place (AXiP) and Encrypted eXecute in Place (EXiP). These features help to provide protection for code placed in flash memory.

Three of the main principles of security are ensuring confidentiality, integrity, and authenticity. Confidentiality guarantees data cannot be accessed by unauthorized parties. Integrity guarantees data has not been tampered with or modified. Authenticity guarantees data is received from an authorized source. AXiP uses AES-GCM encryption, which helps to ensure integrity, confidentiality, and authenticity of code placed in external memory through encrypting the data, which ensures confidentiality, and including an authentication tag, to ensure the data placed in memory is from an authorized source, and has not been modified. EXiP uses AES-CTR, which helps to ensure confidentiality only, as only encryption is applied to data in flash memory.

While Silicon Labs will always recommend implementing the highest level of security available on Series 3 devices, it is ultimately up to end-device manufacturers to make their own risk analysis to determine the level of security to implement in their devices. As with any engineering solution, there will be tradeoffs that should be analyzed when implementing these features. For example, AXiP, the highest security option adds a 4-byte MAC to each 32-bytes of ciphertext, which will in turn, consume more memory. EXiP, will result in the same data size between plaintext and ciphertext, however, it will not come with the additional benefit of ensuring authenticity and integrity.

The following sections will continue to go further in depth on the AXiP and EXiP features of Series 3, including details on the functions of and how to implement each feature.

3. Authenticated eXecute in Place

Series 3 devices introduce the new AXiP feature which enhances the security of flash by providing encryption and authentication of external flash contents through the EXTMEM and SE subsystems. The SE is involved by providing the AXiP and EXiP keys, generating a unique per-region IV, and configuring code regions for AXiP, EXiP, or no protection. The EXTMEM subsystem handles memory access through a high speed QSPI interface with dedicated hardware to encrypt, decrypt, and authenticate data. Because all memory read and write operations on Series 3 devices are routed through the SE and EXTMEM subsystems, AXiP is seamlessly integrated into the firmware flashing and execution process. Each step is handled securely by these systems, requiring no user interaction to generate keys, manage encryption, or verify authenticity.

3.1 Algorithm Used

AXiP utilizes the AES-GCM encryption algorithm. AES-GCM is a widely adopted authenticated encryption mode that combines the confidentiality of AES-CTR with the integrity assurance of a Message Authentication Code (MAC).

The AES-GCM algorithm takes in 32-bytes of plaintext and encrypts it with the AXiP Key stored within the SE using an IV generated by the SE for each AXiP enabled code region. The output of this computation is 32-bytes of ciphertext appended with a 4-byte MAC. This 4-byte MAC is independently verifiable per 32-byte blocks of ciphertext. The MAC stored is not directly accessible by the host core, because it has no associated logical address. More details on the differences between physical and logical addresses are covered in 5. Configuring Code Regions.

On decryption, the MAC is verified per 32-bytes of data before decrypting and executing instructions. If the ciphertext is modified at any time, the MAC will no longer be valid, which will cause the MAC verification to fail. If MAC verification fails, a hard fault will occur. This helps to ensure strong integrity, but introduces an additional memory consumption, which should be considered when planning memory usage.

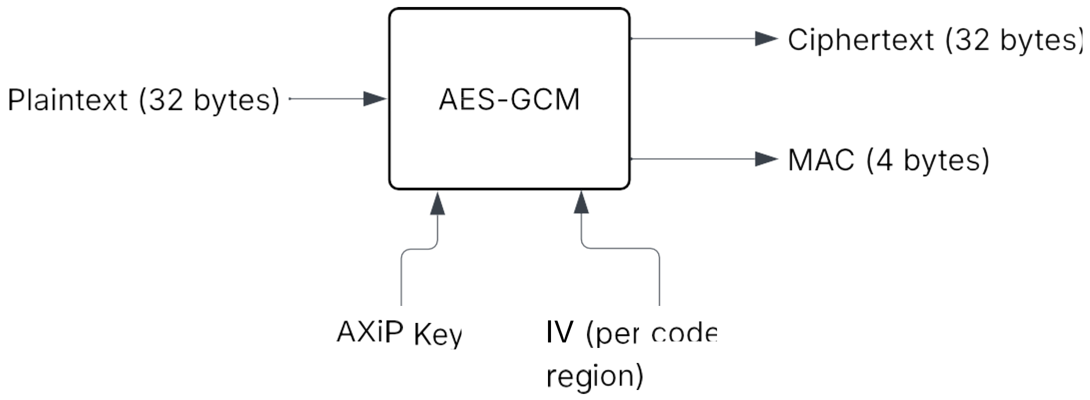


Figure 3.1. AES-GCM Algorithm Overview

3.2 AXiP Key

The AXiP feature utilizes a single 256-bit AES key that is derived at boot from a device-unique Physical Unclonable Function (PUF) seed. The PUF is only available while the device is powered on, and the AXiP key derived from this unique seed when needed. This ensures that the AXiP key has enhanced resistance to physical attacks. As this key is derived from the PUF, it is permanent for the lifetime of the device. The AXiP key is shared across all AXiP-configured code regions of the device.

3.3 IV Generation

Unique initialization vectors ensure that multiple encryptions of identical plaintexts yield different ciphertexts. In AXiP, IVs are generated using the True Random Number Generator (TRNG) and are stored within the SE's MTP. In order to prevent IV reuse, which could introduce predictable bit patterns, a new IV is generated when flashing a code region with AXiP enabled. For more details, refer to the 6. Considerations for Development Devices.

3.4 AXiP Default Device Configuration

Series 3 devices with in-package flash are delivered secured by default, with AXiP enabled out of the box on two code regions. No additional configuration is required to use AXiP on a factory-new in-package flash devices, users only need to flash a firmware image to the predefined code region to implement AXiP.

Unlike in-package flash devices, external flash devices require external flash initialization and SE firmware programming before AXiP is enabled. After the external flash is set up, two regions are configured for AXiP, similar to the default configuration of in-package flash devices. Beyond these external flash initialization steps, region configuration and usage are equivalent to in-package devices.

Typically, a factory-new in-package flash device is set up with Code Region 0 designated for bootloader use. This region is configured with AXiP enabled and a fixed size of 32 kB in logical address space. Code Region 1 is generally intended for application code. It is also AXiP-enabled and sized in multiples of 32 kB logical space, depending on application requirements. The data region is located after the final configured code region, and is a variable size dependent on code region configuration. The default code region sizes may vary among Series 3 OPNs, depending on the flash capacity of the in-package flash devices.

An example of default device configuration is shown below on a 2048 kB flash SixG301 device. Additional details on physical and logical flash mapping on devices with AXiP enabled can be found in 5.2 Physical versus Logical Mapping .

Note: By default, the initial 192 kB of flash memory is allocated for SE Firmware. This region is reserved and not accessible to the user.

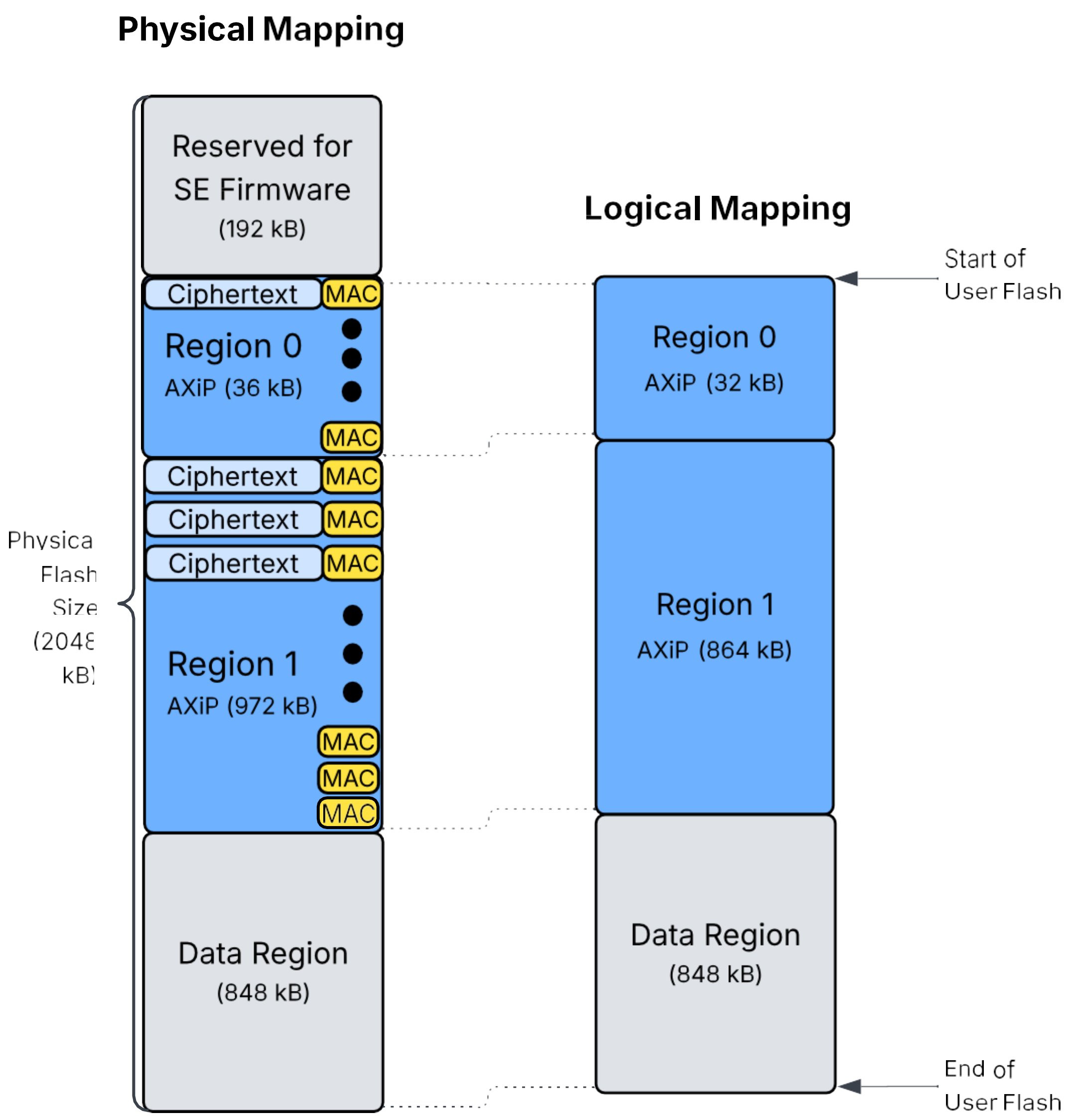


Figure 3.2. Example of Default Device Configuration on a 2048 kB SixG301 Device with In-Package Flash

4. Encrypted eXecute in Place

Series 3 devices introduce the new EXiP feature which enhances the security of external flash by providing encryption during programming and decryption of ciphertext during execution, through the SE and EXTMEM subsystems.

4.1 Algorithm Used

EXiP utilizes the AES-CTR encryption algorithm which provides confidentiality for flash contents. When EXiP is enabled, 16 bytes at a time will be taken to encrypt plaintext or decrypt ciphertext with the EXiP Key stored within the SE using an IV generated by the SE for each EXiP enabled code region.

While EXiP uses less memory than AXiP, it is considered a less secure solution, due to the underlying algorithm differences. Not only could an attacker modify ciphertext without detection, since there is no integrity or authenticity check on the data, AES-CTR is also vulnerable to bit-flipping attacks. This allows an attacker to XOR bits in the ciphertext, which results in the same bits being flipped in the plaintext, once decrypted.

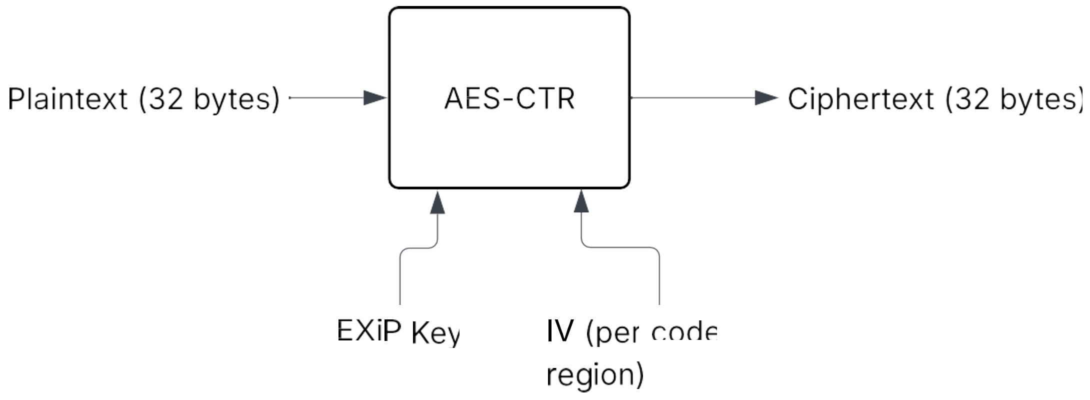


Figure 4.1. AES-CTR Algorithm Overview

4.2 EXiP Key

The EXiP feature utilizes a single 256-bit AES key that is derived at boot from a device-unique Physical Unclonable Function (PUF) seed. The PUF is only available while the device is powered on, and the EXiP key derived off of this unique seed when needed. This ensures that the EXiP key has enhanced resistance to physical attacks. As this key is derived from the PUF, it is permanent for the lifetime of the device. This EXiP key is shared across all EXiP-configured code regions of the device, and is a separate key from the AXiP key.

4.3 IV Generation

Unique initialization vectors ensure that multiple encryptions of identical plaintexts yield different ciphertexts. In EXiP, IVs are generated using the true random number generator and are stored within the SE's MTP. In order to prevent IV reuse, which could introduce predictable bit patterns, a new IV is generated when flashing a code region with EXiP enabled. For more details, refer to the 6. Considerations for Development Devices section.

4.4 EXiP Device Configuration

On Series 3 devices EXiP is not enabled by default. A user must configure their device for EXiP in order to utilize the feature. Refer to 5. Configuring Code Regions for more details on how to configure code regions for EXiP.

5. Configuring Code Regions

Devices with in-package flash are delivered secure by default, with AXiP enabled on two code regions, so no additional configuration is typically required. However, users working with external flash devices, or those with specific application needs, may choose to configure code regions for greater flexibility. Configuring code regions is highly application dependent, so some consideration and planning is needed. This section will serve as a guide to helping users understand how to plan out code region configurations.

There are two methods that can be used in order to set a custom code region configuration. One method is to use SE Manager APIs, such as `sl_se_code_region_get_config`, `sl_se_code_region_apply_config`, and `sl_se_code_region_close`, which respectively read, apply, and close the custom region configuration. More details on these APIs can be found in our SE Manager Documentation.

The second method is configuring code regions through the DCI on the device using Simplicity Commander. This is the main method used in upcoming sections of this document.

5.1 Data Regions versus Code Regions

Series 3 devices introduce the concept of separate code regions and data regions in flash. Code regions are made up of 32 kB erasable pages which are used to store firmware images. Each code region supports only one updatable firmware image, which is why it is standard practice to allocate one region for the bootloader and another for the application firmware. Alternatively, they can be combined within a single region, in which case both must be updated simultaneously and share the same version number.

Data regions are 4 kB erasable pages which are used to store data, and are most commonly used for NVM3 and GBLv4 storage. The XiP features (AXiP and EXiP) are only applicable to code regions, not to data regions. This does not mean that data in the data region cannot be protected; NVM3 and GBLv4 data can be encrypted before it is written to the data region.

It is important to take into account the size of the data region required by an application when configuring code regions, as resizing code regions impacts the size of the data region. Configuring code regions via OTA updates is not supported. Thus, users should ensure enough space is reserved for the bootloader, application firmware, NVM3, GBLv4, and any other required components, for the lifetime of the device.

5.2 Physical versus Logical Mapping

Differences in physical and logical flash mapping must be considered when resizing regions to ensure no code or data regions are overwritten. Physical flash space is allocated to regions based on configuration (AXiP, EXiP, plaintext) and region type (code, data, or reserved for SE firmware).

One reserved region is allocated to physical flash is a 192 kB region reserved for SE Firmware. This is placed at the beginning of physical flash, and ends at the start of user accessible flash, or the start of code region 0.

When AXiP is enabled, an extra 4-bytes of MAC data is stored for every 32-bytes of ciphertext. Thus, this results in 4 kB of MAC data stored for every 32 kB ciphertext data. This MAC is not directly accessible, because it has no associated logical address, but must be accounted for when calculating code and data region sizes.

For EXiP, plaintext code regions, and data regions, one-to-one mapping is used between physical and logical flash maps, independent of other code regions. Address translation between physical and logical flash mapping is handled by the EXTMEM subsystem.

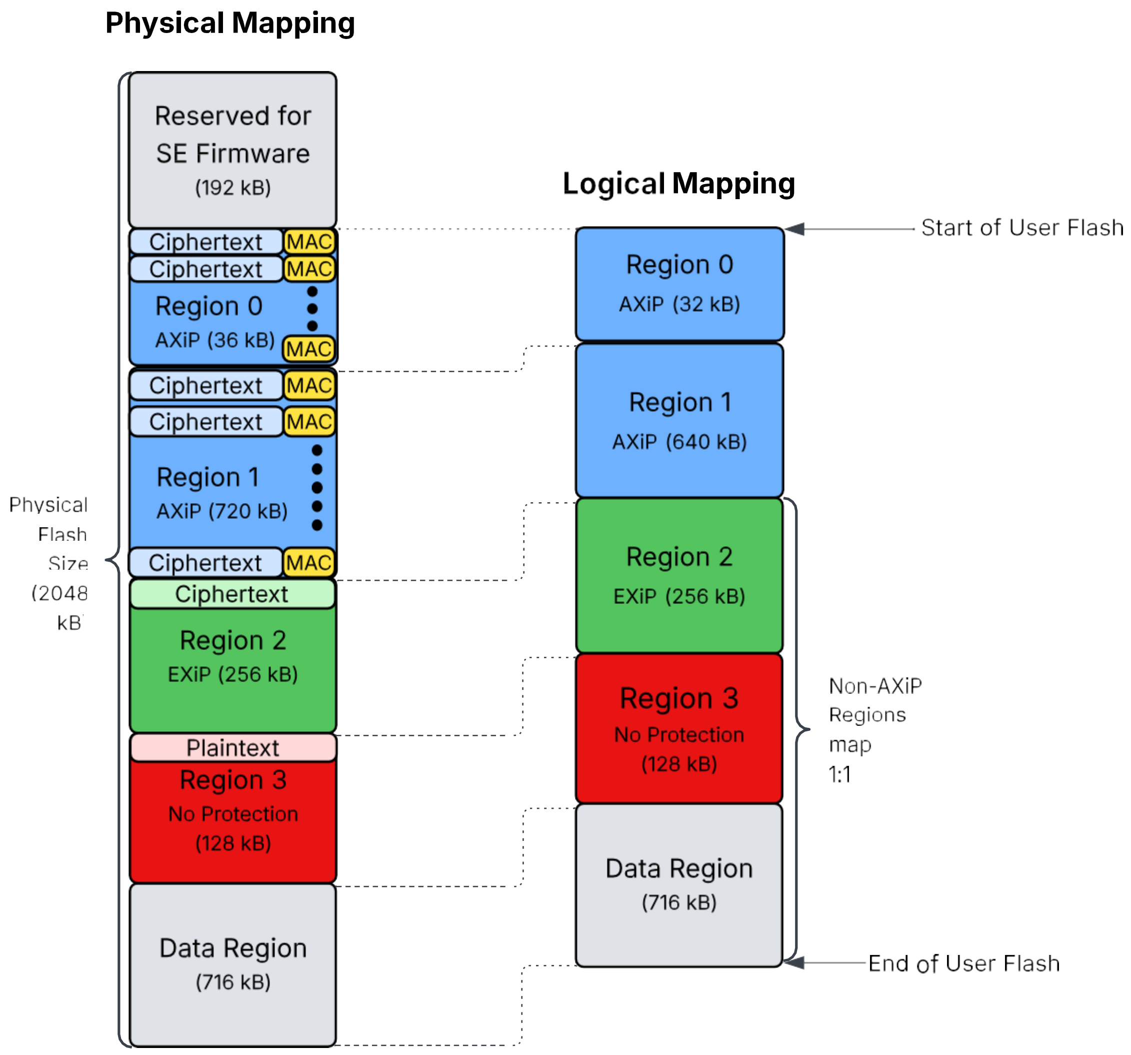


Figure 5.1. Physical versus Logical Mapping of Custom Configured SixG301 Device

An example of a SixG301 device with a custom code region configuration is shown above. The image illustrates a side-by-side comparison of physical and logical address space. In this example, AXiP is enabled on Region 0 and 1, resulting in 4 kB of MAC data stored per 32 kB of ciphertext. The EXiP, plaintext, and data regions map one to one.

To determine the remaining flash available for the data region, subtract the total size of all configured code regions (including MAC overhead) and the reserved SE firmware area from the total flash size. The result is the available space for the data region. For example: Data Region Size = (Total Flash) – (Reserved SE Firmware Region) – (Total Code Region Size, including MAC overhead)

5.3 Code Region Configuration File

To configure code regions via the DCI, Commander takes in a YAML format configuration file, which can be generated by reading the current configuration from the device. The examples section of this document contains step-by-step instructions on how to use this file to reconfigure code regions. An example code region configuration file in YAML format is shown below.

```
regions:
  - size_kb: 32
    protection: encrypted_authenticated
  - size_kb: 1408
    protection: encrypted_authenticated
```

Listed below are the required fields for the configuration file as well as their acceptable values.

Regions

- Up to 8 regions can be configured
  - Each region is listed as a bullet point, starting from Region 0 at the top of the file, and incrementing one region at a time
  - Each region must have a size and protection type set

Size

- Each region must be sized as a multiple of 32 kB

Protection Type

- Setting the protection to `encrypted_authenticated` enables AXiP on the code region
- Setting the protection to `encrypted` enables EXiP on the code region
- Setting the protection to `none` disables all protection on the code region, resulting in the code stored in plaintext

For example, the code region configuration for the custom configuration shown in is listed below.

```
regions:
  - size_kb: 32
    protection: encrypted_authenticated
  - size_kb: 640
    protection: encrypted_authenticated
  - size_kb: 256
    protection: encrypted
  - size_kb: 128
    protection: none
```

Note: When code regions are modified, firmware images must also be updated to ensure the linker files align with the code region configuration. Refer to Platform Software Documentation for details on modifying linker files to match code region configurations.

5.4 Closing a Code Region

Once code regions are configured and firmware is programmed to the device, code regions **must** be closed. Closing a code regions locks the code region to prevent writes to the region, as well as further region configuration. Each time a code region is closed, the SE's MTP is updated, which causes an SE OTP bit to be used. More details on SE OTP bits is covered in the following section.

Note: To open a closed code region, an erase of the region must be performed.

6. Considerations for Development Devices

When developing with AXiP or EXiP-enabled devices, certain considerations must be taken into account. Specifically, SE OTP bits are consumed each time an IV is generated and when a code region is closed. This occurs because each of these operations triggers an update to the SE MTP memory. The OTP bits serve to prevent rollback attacks by ensuring that the SE MTP cannot be reverted to a previous state.

By default, IVs are regenerated each time a code region is erased to prepare the region for reprogramming. Additionally, flashing a firmware image using Simplicity Commander automatically triggers the closure of the associated code region. During development, where devices are frequently flashed and erased, this default behavior can lead to exhaustion of SE OTP memory.

The SE OTP counter is at least 2 kB in size and may be allowed to overflow into the next available OTP region if space permits. However, once all available OTP space is exhausted, the device enters an EOL (End of Life) state and can no longer be reflashed. This section outlines best practices to avoid SE OTP exhaustion during development.

6.1 Reading SE OTP Count

The current number of OTP bits consumed can be read through the use of the following command. Once SE OTP is consumed, the device will enter an EOL state. In this state, requested operations such as reflashing a device will not be able to be carried out.

```
commander security otprollbackcount -d sixg301
```

```
Number of used OTP rollback bits: 9
DONE
```

6.2 Skip Closing a Code Region

When using Simplicity Commander, code regions are closed automatically when using the `commander flash` command. To avoid closing a code region when running this command, a user can specify the `--noclose` flag, which will leave the region open. This option is only intended for use to prevent exhaustion of OTP bits on development devices when continuous reflashing is done.

Note: Closing a code region is a prerequisite for Secure Boot, as it enables the SE to access Region 0 to perform signature validation on the bootloader firmware. Therefore, on development devices with secure boot enabled, code regions must be closed in order for firmware to be executed by the SE.

```
commander flash example.hex --noclose -d sixg301
```

```
Parsing file example.hex...
Writing 168756 bytes starting at address 0x01000000
Erasing range 0x01000000 - 0x01007FFF (1 sector, 32 KB)
Erasing range 0x01008000 - 0x010B7FFF (1 sector, 704 KB)
Programming range 0x01000000 - 0x01001FFF (8 KB)
...
Programming range 0x010B6000 - 0x010B7FFF (8 KB)
Flashing completed successfully!
DONE
```

On production devices, a code region must be closed after firmware programming to prevent further modification of the region. This can be done through the flashing command, without specifying the `--noclose` flag, or through a standalone DCI command in Simplicity Commander, shown in 7.3 Closing a Code Region.

6.3 Transition to Development Device Command

In order to avoid resource exhaustion on development devices due to IV rolling, the transition to development command (also referred to as the Skip IV Roll command) can be used to issue a one-time command which will enforce the same IV to be used for the lifetime of the device. As this flag is enabled in OTP, this setting cannot be reversed once enabled. **Devices with this option set must only be used for development purposes.** Once Skip IV Rolling is enabled, devices should be considered non-secure and should not enter production, as reusing IVs on devices in the field can introduce vulnerabilities which could lead to data confidentiality being impacted. When this option is not set, IVs will continue to be rolled on each flash erase, which will maintain security of the encrypted data.

To permanently disable IV rolling on a development device, issue the following command. This command was added in Simplicity Commander version 1.19.2 release.

```
commander security transitiontodevelopment -d sixg301
```

```
==============================================================================
THIS IS A ONE-TIME command which permanently changes the security properties of the device.
Once done, the device permanently uses the same IV for encryption of flash, which is not
secure (using the same IV and same key can lead to exposure of the encrypted data).
This should only be enabled on development devices to prevent exhausting OTP rollback
bits after repeated flash/erase cycles.

Type 'continue' and hit enter to proceed or Ctrl-C to abort:
==============================================================================
continue
The device has been permanently transitioned to NOT SECURE development mode
DONE
```

Note: This command prevents OTP bits from being consumed during IV generation only. OTP consumption will still occur during code region closures and firmware version updates; therefore, OTP exhaustion remains possible even when this command is used.

7. Examples

Prerequisites

At the time of publication, this document was written using Simplicity Commander version 1.19.2 and SE Firmware version 3.3.1. It is a requirement to upgrade to at least these versions before proceeding, but it is recommended to upgrade to the latest version of both products, if a newer version is available. Install the latest version of Simplicity Commander delivered via silabs.com, to ensure you have the latest features and bugfixes. Additionally, it is recommended to upgrade devices to the latest SE Firmware version, delivered as part of the Simplicity SDK, as support for new Commander instructions, bug fixes, features, and vulnerability patches may be included in these updated versions.

Note: Before continuing with the steps listed in this section, refer to 6. Considerations for Development Devices for important information on using AXiP and EXiP on development devices.

7.1 Using AXiP Out of the Box

When using an in-package flash device with default configuration, all that needs to be done is to flash a valid image to the device. This section shows how to read the default region configuration on the device and program a firmware image with automatic closing of code regions.

On devices with external flash, AXiP is not enabled out of the box. Once the external flash is initialized and SE firmware is programmed, AXiP is configured to the default two-region setup (bootloader and application), matching the behavior of in-package flash devices.

Reading Default Code Region Configuration

1. To read the region configuration details from a device, issue the following command. This command is not required to be run for use AXiP out of the box, it is only intended to display the current configuration status.

```
commander security readregionconfig -d sixg301
```

```
Index        : 0
Size         : 32 kB
Protection   : Encrypted and authenticated
Closed       : False

Index        : 1
Size         : 1408 kB
Protection   : Encrypted and authenticated
Closed       : False

DONE
```

Flashing a Device with AXiP Enabled

1. To use AXiP on this device, simply flash the firmware image to the device using the following command. This command will automatically close the code regions after flashing is completed successfully.

```
commander flash example.hex -d sixg301
```

```
Parsing file example.hex...
Writing 168756 bytes starting at address 0x01000000
Erasing range 0x01000000 - 0x01007FFF (1 sector, 32 KB)
Erasing range 0x01008000 - 0x010B7FFF (1 sector, 704 KB)
Programming range 0x01000000 - 0x01001FFF (8 KB)
...
Programming range 0x010B6000 - 0x010B7FFF (8 KB)
Closing region 1
Closing region 0
Flashing completed successfully!
DONE
```

7.2 Custom Code Region Configuration

Modifying the region configuration can be done on devices with open code regions. If code regions are closed, they cannot be modified or reconfigured until they are erased, which will reopen the closed code region.

Note: The following steps begin with a device which is in the default configuration, where all configured code regions are open. If a user is reconfiguring a device with closed code regions, issue a `commander device masserase` pr a `commander device recover` command to the device before proceeding.

1. To begin modifying a device's code region configuration, a configuration YAML file must be generated. The recommended way to do this is to read out the current configuration into a YAML file using the following command.

```
commander security readregionconfig --outfile region_config.yaml -d sixg301
```

```
Writing parsed configuration to file region_config.yaml...
DONE
```

   Below is a copy of the contents of the `region_config.yaml` file. This was read out from a device in the default code region configuration state.

```
regions:
  - size_kb: 32
    protection: encrypted_authenticated
  - size_kb: 1408
    protection: encrypted_authenticated
```

2. Modify the code `region_config.yaml` file and save the modified file. More details on the allowed fields for this file can be found in the 5.3 Code Region Configuration File section of this document. An example of a modified `region_config.yaml` file is shown below.

```
regions:
  - size_kb: 32
    protection: encrypted_authenticated
  - size_kb: 256
    protection: encrypted
  - size_kb: 64
    protection: none
```

3. Write the new configuration file to the device using the following command.

```
commander security writeregionconfig region_config.yaml -d sixg301
```

```
Reading configuration from file region_config.yaml...
Writing region configuration to device...
DONE
```

4. Read out the configuration of the device to verify settings were applied successfully.

```
commander security readregionconfig -d sixg301
```

```
Index         : 0
Size          : 32 kB
Protection    : Encrypted and authenticated
Closed        : False

Index         : 1
Size          : 256 kB
Protection    : Encrypted
Closed        : False

Index         : 2
Size          : 64 kB
Protection    : Plaintext
Closed        : False

DONE
```

5. Once the code regions are configured, firmware can be programmed to the regions using the `commander flash` command. In this example, the `--noclose` flag is used to later illustrate how to close a code region independently from flashing.

```
commander flash bootloader.hex --noclose -d sixg301
```

```
Parsing file bootloader.hex...
Writing 18788 bytes starting at address 0x01000000
Erasing range 0x01000000 - 0x01007FFF (1 sector, 32 KB)
Programming range 0x01000000 - 0x01000FFF (4 KB)
Programming range 0x01001000 - 0x01001FFF (4 KB)
Programming range 0x01002000 - 0x01002FFF (4 KB)
Programming range 0x01003000 - 0x01003FFF (4 KB)
Programming range 0x01004000 - 0x01004FFF (4 KB)
Programming range 0x01005000 - 0x01005FFF (4 KB)
Programming range 0x01006000 - 0x01006FFF (4 KB)
Programming range 0x01007000 - 0x01007FFF (4 KB)
Flashing completed successfully!
DONE
```

7.3 Closing a Code Region

1. The following command can be used to close a code region. The command takes in a region index, starting from 0, and an optional code region version number. This command was run following the previous step where a bootloader was flashed to a device. Closing a code region is required after regions are configured and firmware is programmed.

   Note: The code region version can only be set when a code region is closed. This functionality is supported exclusively through the DCI using this command.

```
commander security closeregion 0 --codeversion 2 -d sixg301
```

```
Succesfully closed code region 0 (version 0x00000002)
DONE
```

2. To verify the code region was closed, read out the current code region configuration.

```
commander security readregionconfig -d sixg301
```

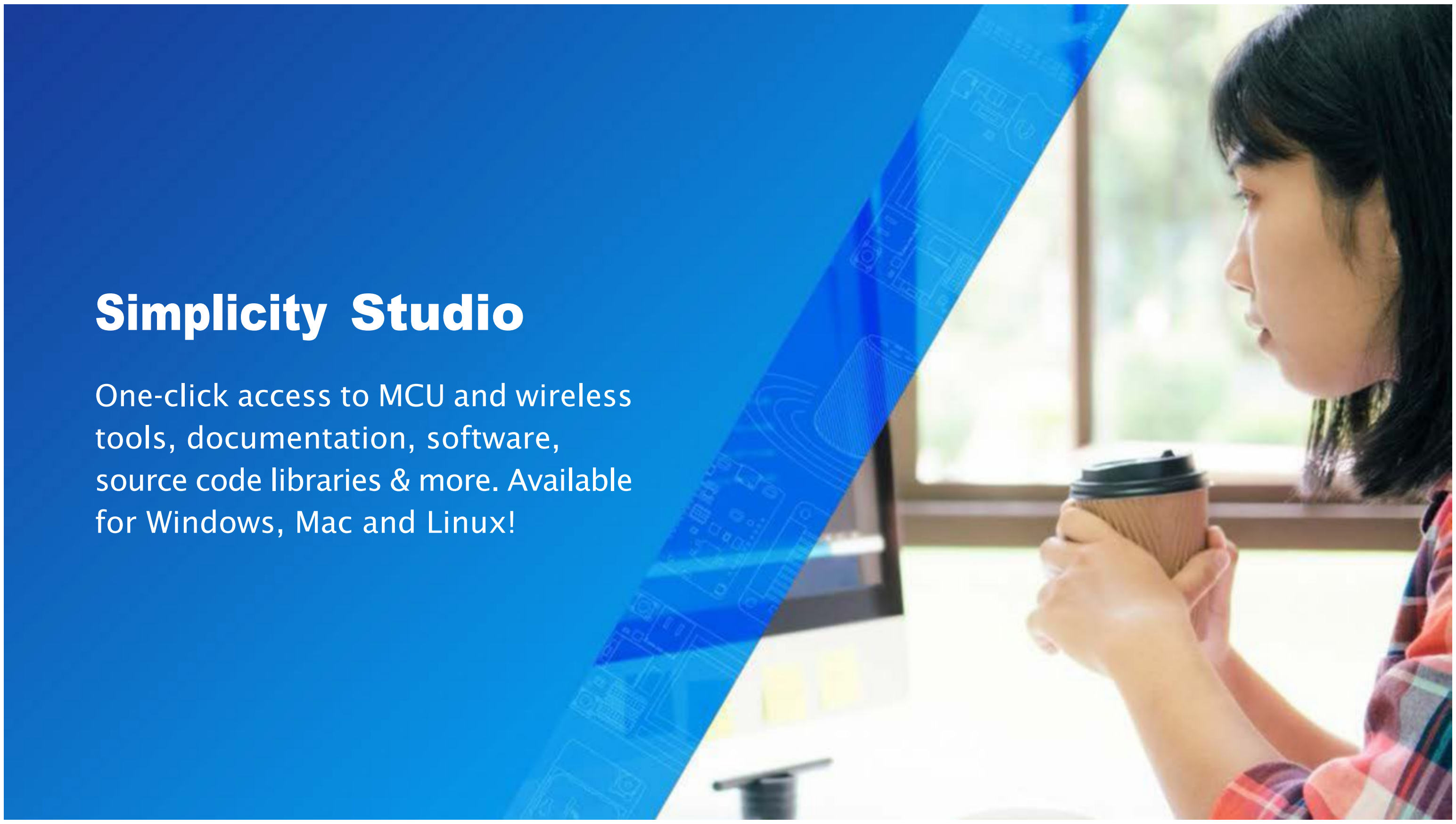
```
Index         : 0
Size          : 32 kB
Protection    : Encrypted and authenticated
Closed        : True

Index         : 1
Size          : 256 kB
Protection    : Encrypted
Closed        : False

Index         : 2
Size          : 64 kB
Protection    : Plaintext
Closed        : False

DONE
```

8. Revision History

Revision 0.1

September, 2025

- Initial revision


Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com