Using Encryption and Authentication to Secure an UltraScale/UltraScale+ FPGA Bitstream

XAPP1267 (v1.8) May 22, 2025

Summary

IMPORTANT! *See AMD Design Advisory 68832 for important updates about eFUSE programming with AMD Vivado™ Design Suite 2016.4 and earlier versions.*

This application note describes a simple step-by-step process to generate an encrypted bitstream and encryption keys (both Advanced Encryption Standard Galois/Counter Mode (AES-GCM) and RSA authentication) using the AMD Vivado™ Design Suite. Steps to program the Advanced Encryption Standard Global System for Mobile communications (AES-GSM) encryption key and the hash of the RSA public key, along with the encrypted bitstream into an AMD UltraScale™ FPGA using the Vivado Design Suite are also included. This application note applies to both AMD UltraScale™ and AMD UltraScale+™ FPGAs. This document is not intended to discuss security issues, such as generating keys or selecting initialization vectors (IVs) to comply to National Institute of Standards and Technology (NIST) standards.

Note: This application note does not apply to AMD Spartan™ UltraScale+™ FPGAs.

Introduction

UltraScale devices have on-chip AES-GCM decryption and authentication logic to provide a high degree of design security. Encrypted UltraScale FPGA designs cannot be copied or reverse-engineered. The UltraScale FPGA AES system comprises software-based bitstream encryption and on-chip bitstream decryption with dedicated memory for storing the encryption key and encrypted bitstream. The encryption key and the encrypted bitstream are generated using the Vivado tools.

UltraScale devices store the encryption key internally in either dedicated RAM, backed up by a small, externally connected battery, known as battery backed RAM (BBRAM), or in the eFUSE. If using RSA authentication, the hash of the RSA Public key must be programmed into the eFUSE. The encryption key can only be programmed into the device through the JTAG port. Neither the BBRAM nor eFUSE can be read back. During configuration, the UltraScale device performs the reverse operation, decrypting the incoming bitstream. The UltraScale FPGA AES encryption logic uses a 256-bit encryption key. The on-chip AES decryption logic cannot be used for any purpose other than bitstream decryption; for example, the AES decryption logic is not available to the user design and cannot be used to decrypt any data other than the configuration bitstream.

Advanced Encryption Standard and Authentication

The UltraScale FPGA encryption system uses the AES-GCM authenticated encryption algorithm. AES is an official standard supported by the NIST and the U.S. Department of Commerce (see the *Advanced Encryption Standard (AES)* (FIPS PUB 197) and GCM specification *The Galois/Counter Mode of Operation (GCM)* (Specification) for more information).

An advantage of AES-GCM is that it also supports built-in authentication. The UltraScale FPGA AES encryption system uses a 256-bit encryption key (the alternate key lengths of 128 and 192 bits described by NIST are not implemented) to encrypt or decrypt blocks of 128 bits of data at a time. According to NIST, there are 1.1×10^{77} possible key combinations for a 256-bit key. For the most secure approach, it is recommended that you create this 256-bit key manually rather than use the pseudo-random key generator feature provided by Vivado.

Bitstream Authentication

The AES-GCM encryption standard supports built-in authentication, enhancing security and eliminating the need to specify a separate HMAC key as in the 7 series FPGAs. Without knowledge of the AES-GCM key, the bitstream cannot be modified or forged. Encryption provides the basic design security to protect the design from copying or reverse engineering, while authentication provides assurance that the bitstream provided for the configuration of the FPGA was the unmodified bitstream created by an authorized user. Authentication verifies both data integrity and authenticity of the bitstream.

Authentication covers the entire bitstream for all types of control and data. Any bitstream tampering including single bit flips are detected. If authentication passes, the configuration goes to completion through the startup cycle. If authentication fails the device will not start up if any changes to the bitstream are detected by the AES-GCM engine. If fallback is enabled the fallback bitstream is loaded after the entire device configuration has been cleared. If fallback is not enabled, the configuration logic disables the configuration interface, blocking any access to the FPGA. Pulsing the PROGRAM_B signal or power-on reset is required to reset the configuration interface. You will need to select one of two choices for bitstream authentication:

- If you are using bitstream encryption, you can rely on the authentication built into the AES-GCM standard.
- Additionally, you can rely on RSA-2048 Authentication to authenticate the bitstream before it is decrypted. RSA-2048 is discussed in the following paragraphs.

RSA Authentication

AES-GCM is a self-authenticating algorithm with a symmetric key, meaning that the key to encrypt is the same as the one to decrypt. This key must be protected as it is secret (hence storage to internal key space). The UltraScale architecture provides for an alternative form of authentication in the form of RSA-2048. RSA is an asymmetric algorithm, meaning that the key to verify is not the same key used to sign. The verification is done with a public key. This public key does not need to be protected and does not need special secure storage. If desired, this form of authentication can be used with encryption to provide both authenticity and confidentiality. RSA not only has the advantage of using a public key, it also has the advantage of authenticating prior to decryption. The hash of the RSA Public key must be stored in the eFUSE.

UltraScale FPGAs support RSA-2048 for the purpose of authenticating the bitstream data before it is sent to the decryptor. This method can be used to help prevent attacks on the decryption engine itself by ensuring that the data is authentic before performing any decryption. The RSA configuration control logic reads the encrypted bitstream, including a public key and bitstream signature, into the device memory. The RSA configuration control logic then instructs the RSA engine to calculate the expected digest based on the public key and signature.

After the bitstream is buffered and the RSA engine has calculated the expected digest, the actual digest is compared against that result. If RSA authentication passes and the configuration data was encrypted, then the FPGA is released for decryption of the bitstream. If RSA authentication fails, an error equivalent to an AES-GCM authentication error is generated. At this point the device either locks down or, if enabled, a fallback occurs.

A device configured with an RSA authenticated bitstream can take up to three times as long to configure as a standard uncompressed bitstream for that device. The actual time depends on the mode of configuration. RSA authentication cannot be used with bitstream compression, partial reconfiguration, or configuration over the PCIe® interface, including tandem solutions.

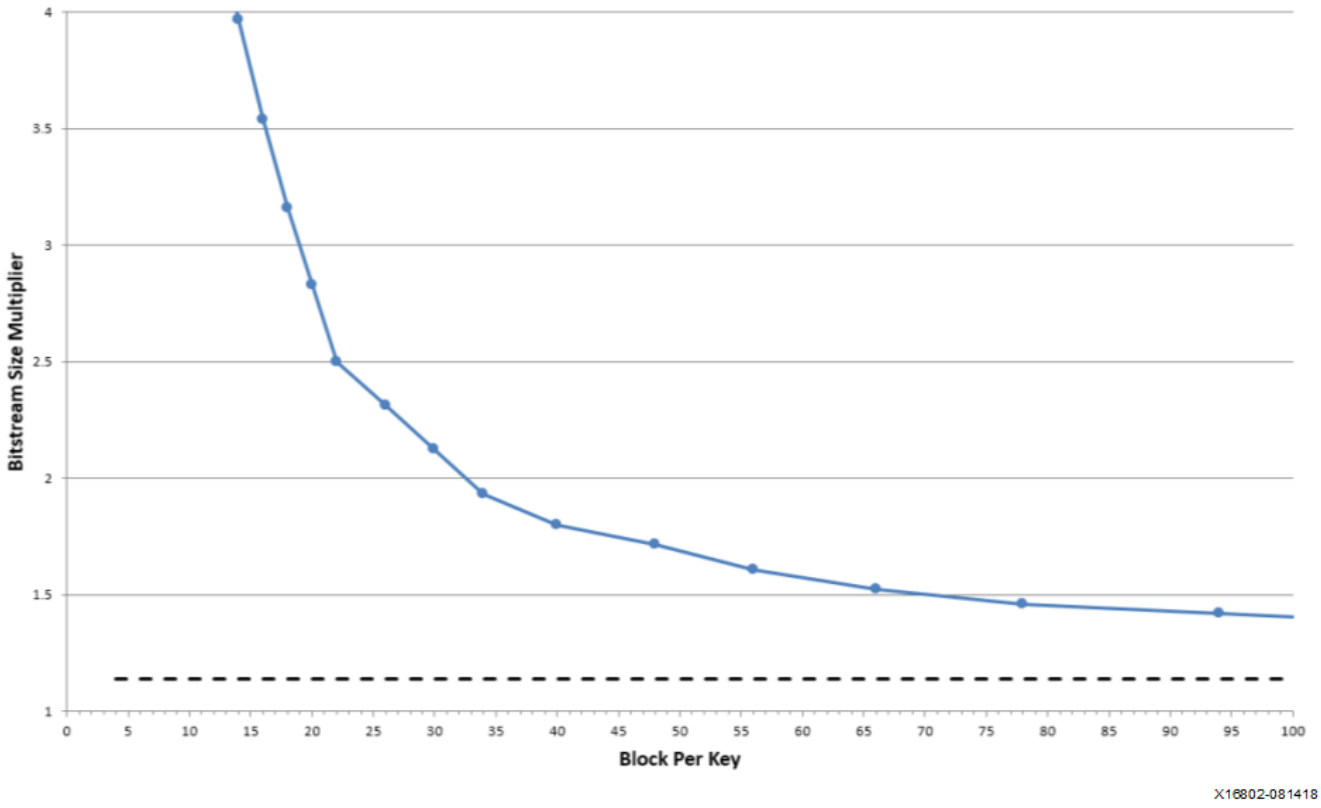
RSA authentication is supported in UltraScale and UltraScale+ devices with certain configuration modes and widths. For UltraScale FPGA devices and configuration modes that support RSA authentication, see the RSA Authentication section in the *UltraScale Architecture Configuration User Guide* (UG570).

IMPORTANT! *RSA authentication on AMD UltraScale™ and AMD UltraScale+™ FPGA series devices can be circumvented when encryption is not enforced using eFUSE programming. To avoid potential security issues, users requiring RSA authentication should also enforce encryption by configuring an AES key in either BBRAM or eFUSES and configure the device to require encrypted bitstreams by programming the FUSE_SHAD_SEC[0] eFUSE. Devices configured to enforce both RSA authentication and encryption are not susceptible to the RSA authentication issue. For further details, refer to AMD Design Advisory 000036039.*

Key Rolling

UltraScale FPGAs allow you to break up the bitstream into multiple AES encryption messages, each encrypted with its own unique key. With this feature, known as *rolling keys*, the initial key is stored on-chip, while keys for each successive message are encrypted (wrapped) in the previous message. Rolling keys increases security against side-channel attacks such as differential power analysis (DPA). The bitstream option BITSTREAM.ENCRYPTION.KEYLIFE defines the number of encryption blocks per key. Fewer encryption blocks per key offers greater security but greatly increases bitstream size and therefore configuration time. Selecting a value such as 1,024 or higher increases configuration size by about 15%, a value of 64 can increase bitstream size by 50%, and a value of 32 (default) can more than double the bitstream size. See the following figure for a graph showing bitstream size multiplier vs. block per key.

Figure 1: **Bitstream Size Multiplier vs. Block per Key**



AMD strongly recommends to create your own AES key, however if you choose to allow the Vivado software to generate your pseudo-random keys, you will see the number of Keys (Key0, Key1, Keyn...) included in the resulting NKY file. To define multiple custom keys you must provide them in a .NKY file and use the BITSTREAM.ENCRYPTION.KEYFILE write_bitstream property. For additional information regarding this write_bitstream property refer to Table 6: Write_bitstream Properties or see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

IMPORTANT! *See AMD Design Advisory 76171 for important updates about generating your own keys for fielded systems and providing the keys in the development tools.*

When using RSA authentication, certain block RAMs might be used to hold interim rolling keys, which impacts the ability to initialize those blocks. For any given block RAM column, each 36K block that resides in the bottom of a clock region is affected; essentially the first 36K block RAM starting at the bottom of a device and then every 12th 36K block RAM after that in a column (BRAM36_X*Y0, BRAM36_X*Y12, BRAM36_X*Y24, etc.). Those block RAMs cannot be initialized to user-defined values when using RSA authentication. Those block RAMs are always initialized to 0 after configuration. A DRC will trigger if you are using block RAM in your design affected by this RSA block RAM usage.

If you are using the more secure approach and defining your own keys, and have not provided the correct amount of keys determined by your BITSTREAM.ENCRYPTION.KEYLIFE option (could potentially be thousands of keys), Vivado will generate the necessary keys on your behalf and include them in your NKY file. AMD recommends that you always generate a complete set of your own keys.

Encrypted Bitstream Implementation Overview

The following is a list of seven fundamental steps needed to implement an encrypted design in an UltraScale FPGA:

1. Choose an AES key storage location: BBRAM or eFUSE; and corresponding security options. (See Developing Tamper-Resistant Designs with *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs* (XAPP1098) for trade-offs between BBRAM and eFUSE).
2. Choose if RSA authentication is being used: enable either AES-GCM or AES-GCM and RSA authentication. (See *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs* (XAPP1098) for trade-offs between AES-GCM and RSA Authentication).
3. Implement the hardware requirements in your board design based on your AES key storage location selection.
4. Using Vivado Design Suite software, generate an AES key or provide your own custom AES key to the software (which is always the most secure approach) and encrypt the bitstream:
   a. Generate/create the AES key.
   b. If RSA was chosen as the authentication method, generate an RSA public/private key pair using OpenSSL (www.openssl.org) or other key generation software.
5. Program the AES key into the FPGA using the JTAG interface.
6. Program the encrypted bit file into the FPGA via JTAG or another configuration mode such as SPI or BPI.

   ***Note*:** For UltraScale FPGA devices and configuration modes that support RSA authentication, see the RSA Authentication section in the *UltraScale Architecture Configuration User Guide* (UG570).
7. Perform hardware validation to ensure proper operation.

Hardware Board Requirements

There are a few basic hardware requirements for implementing an encrypted design flow:

- For programming ability and debugging capability: A JTAG connection to the FPGA.

- For BBRAM key storage: Battery to V_{BATT} (see the respective data sheet for battery voltage requirements).
- For eFUSE key storage: V_{BATT} or V_{CCAUX} is recommended to enable the ability to test with BBRAM flow prior to burning the one-time programmable (OTP) eFUSEs.

Software Requirements

Vivado Design Suite 2017.1 or newer is recommended.

AES Key Storage

There are two options when considering AES-GCM key storage; BBRAM or eFUSE.

RECOMMENDED: *When selecting the BBRAM or eFUSE storage options, it is highly recommended that you consider the advantages and disadvantages of each option and which option fits your design requirements best.*

The following sections detail each of their respective advantages and disadvantages. Additional information on each of these storage options can be found in the *UltraScale Architecture Configuration User Guide* (UG570).

BBRAM Storage Location

When an encryption key is stored in the FPGA's battery-backed RAM, the encryption key memory cells are volatile and must receive continuous power to retain their contents. During normal operation, these memory cells are powered by the auxiliary voltage input (V_{CCAUX}).

RECOMMENDED: *A separate V_{BATT} power input is needed to retain the key if and when V_{CCAUX} is removed. Therefore, it is recommended that the AES key be programmed in-system on a board that has the battery back-up. Otherwise, the key is lost when the power/battery is removed.*

IMPORTANT! *Program the BBRAM to a known state before attempting to configure with an encrypted bitstream that uses the BBRAM as the key source. If you attempt to download an encrypted bitstream on power-up before the BBRAM key is programmed, the FPGA might lock up. You must power-cycle the device and then load the BBRAM key before configuring with an encrypted bitstream.*

The following table identifies BBRAM storage location advantages and disadvantages.

Table 1: **BBRAM Storage Location Advantages and Disadvantages**

Advantages	Disadvantages
• Volatile and re-programmable • Passive and active key clearing (for example, the evidence can be removed) • Tamper resistant[1] • BBRAM can use either RSA authentication or Configuration Counting for DPA protection • Cannot readback the BBRAM key as there is no readback path	• Requires an external battery. • Many battery vendors do not specify operation at high temperatures and/or long lifetimes.

Notes:

1. There is no physical path to read the key out of BBRAM (write only access).

eFUSE Storage Location

eFUSE is a non-volatile one-time-programmable technology used for selected configuration settings. The fuse link is programmed (or burned or blown) by flowing a large current for a specific amount of time. User-programmable eFUSEs can be programmed with the AMD configuration tools.

IMPORTANT! *eFUSE bits are one-time programmable (OTP). After they are programmed, they cannot be un-programmed.*

For example, if access to a register is disabled, it cannot be re-enabled. The FPGA logic can access only the FUSE_USER register value. All other eFUSE bits are inaccessible from the FPGA logic. The following table identifies eFUSE storage location advantages and disadvantages.

Table 2: **eFUSE Storage Location Advantages and Disadvantages**

Advantages	Disadvantages
• No external battery is required • Only a bitstream encrypted with the eFUSE key can get loaded into the FPGA • Cannot readback the eFUSE key as there is no readback path • eFUSE requires RSA authentication for DPA protection	• Permanent: Key can NOT be cleared or updated. • Less secure than BBRAM solution because the key cannot be zeroized or updated.

Obfuscated Keys

UltraScale FPGAs enable you to load your AES key into the device in an obfuscated format. This enables you to give the obfuscated key to a contract manufacturer without having to expose your true AES-256 key to the contract manufacturer. When you set the BITSTREAM.ENCRYPTION.OBFUSCATEKEY property, Vivado write_bitstream software creates a new key, ObfuscateKey, in the output NKY file. This obfuscated key is created by encrypting your AES-256 key with a metalized family key stored in the silicon. The same key is used on all UltraScale devices and all UltraScale+ FPGAs. (The UltraScale FPGA family key is different from the UltraScale+ FPGA family key.)

AMD does not provide the family key as part of the Vivado tools. Customers must send a request for the family key to secure.solutions@amd.com. It will then be distributed to qualified customers through the Product Licensing site on www.amd.com.

To specify the location of the family key you must set the following write_bitstream property:

```
BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH C:/<any directory>/
familyKey_us.cfg [current_design]
```

You can give the obfuscated key to your contract manufacturer rather than the actual AES-256 key. When the key is programmed into either the eFUSE or BBRAM, if the NKY file contains an `KeyObfuscate` field, a flag is automatically set in the storage location indicating that this key is obfuscated. The resulting bitstream also contains additional instructions informing the chip to decrypt the appropriate AES-256 key storage location prior to using the key to decrypt the rest of the bitstream. The obfuscated key settings in the location that the bitstream selects must match the obfuscated key settings of the bitstream. The BITSTREAM.ENCRYPTION.OBFUSCATEKEY property is not compatible with the Configuration Counting DPA countermeasure for BBRAM key storage.

eFUSE Registers

An UltraScale FPGA has a total of six eFUSE registers: FUSE_RSA, FUSE_KEY, FUSE_DNA, FUSE_USER, FUSE_CNTL, and FUSE_SEC. For the purpose of this application note the focus is not on the FUSE_DNA register. All of the UltraScale eFUSE registers are described in the following table.

Table 3: **eFUSE Register Description**

Register Name	Size (Bits)	Contents	Description
FUSE_RSA	384	Bitstream authentication key [383:0] (bit 0 shifted first)	Stores a hash of the public key used for RSA bitstream authentication.
FUSE_KEY	256	Bitstream encryption key [255:0] (bit 0 shifted first)	Stores a key for use by AES-GCM bitstream decryption and authentication. The eFUSE key can be used instead of the key stored in battery-backed SRAM (BBRAM). The AES key is used by the UltraScale FPGA decryption engine to load encrypted bitstreams. Depending on the read/write access bits in the CNTL register, the AES key can be programmed through the JTAG port but cannot be read through the JTAG port.
FUSE_DNA	96	Device identifier programmed by AMD [95:0] (bit 0 shifted first)	Unique device identifier bits [95:0], corresponding to the 96-bit read-only DNA_PORTE2 primitive value known as Device DNA.

Table 3: **eFUSE Register Description** *(cont'd)*

Register Name	Size (Bits)	Contents	Description
FUSE_USER	32 or 128	User defined [31:0] or [127:0] (bit 0 shifted first)	Stores a 32-bit or 128-bit user-defined code. The 32-bit version of this register is readable from the FPGA logic using the eFUSE_USR primitive. (See Chapter 7, Design Entry in the UltraScale Architecture Configuration User Guide (UG570) [Ref 3] for a description of the eFUSE_USR primitive. Depending on the read/write access bits in the CNTL register, the code can be programmed and read through the JTAG port. The 128-bit version of this register stores a 128-bit user-defined code. This register is readable from the JTAG FUSE_USER_128 instruction. The JTAG FUSE_USER_128 data register length is 384 bits in UltraScale FPGAs or 176 bits in UltraScale+ FPGAs. Only bits [127:0] are supported for user code storage, and the remaining bits are reserved and can be any value.
FUSE_CNTL	21	Control Bits CNTL [20:0] (bit 0 shifted first)	Controls key use and read/write access to eFUSE registers. This register can be programmed and read through the JTAG port.
	24	Control Bits CNTL [23:0] (bit 0 shifted first)	In UltraScale+ FPGAs these bits control key use and read/write access to eFUSE registers. This register can be programmed and read through the JTAG port.
FUSE_SEC	32	Security Control Bits [31:0] (bit 0 shifted first)	Controls encryption and authentication options. Depending on the read/write access bits in the CNTL register, this register can be programmed and read through the JTAG port.

eFUSE Control Register (FUSE_CNTL) Bit Description

This register contains user programmable bits used to select AES key usage and set the read/write protection for other eFUSE registers. The following table provides bit descriptions and recommended settings.

Table 4: **eFUSE Control Register Bit (FUSE_CNTL) Description**

Bit	Bit Name	Description	Recommended Setting
0	R_DIS_Key	When programmed to 1, this bit disables the CRC check that verifies the AES key and programming of the AES key.	Yes (program to 1)
1	R_DIS_USER	When programmed to 1, this bit disables reading of the FUSE_USER user code. This does not disable reading the user code through EFUSE_USR component, although it disables reading the user code through the JTAG port.	No (keep at 0)
2	R_DIS_SEC	When programmed to 1, the bit disables reading of the FUSE_SEC security settings via JTAG.	Yes (program to 1)
3–4	Reserved	Reserved	–

Table 4: **eFUSE Control Register Bit (FUSE_CNTL) Description** *(cont'd)*

Bit	Bit Name	Description	Recommended Setting
5	W_DIS_CNTL	When programmed to 1, this bit disables programming of the FUSE_CNTL bits. **RECOMMENDED:** *Program this bit to 1 after programming the FUSE_CNTL register bits to prevent unintended changes to the FUSE_CNTL eFUSE bits.*	Yes (program to 1)
6	R_DIS_RSA	When programmed to 1, this bit disables reading of the FUSE_RSA authentication key.	Yes (program to 1)
7	W_DIS_KEY	When programmed to 1, this bit disables programming of the AES key and CRC check that verifies the key. **RECOMMENDED:** *Program this bit after programming the key to prevent unintended changes/corruption to the eFUSE AES key value.*	Yes (program to 1)
8	W_DIS_USER	Disable programming of the FUSE_USER user code.	No (keep at 0)
9	W_DIS_SEC	When programmed to 1, this bit disables programming of the FUSE_SEC register bits. **RECOMMENDED:** *Program this bit after programming the FUSE_SEC register to prevent unintended changes/corruption to the FUSE_SEC register.*	Yes (program to 1)
10–14	Reserved	Reserved	–
15	W_DIS_RSA	When programmed to 1, this bit disables programming of FUSE_RSA authentication key.	Pending customer security requirements
16	W_DIS_USER_128	When programmed to 1, this bit disables programming of FUSE_USER_128 user code.	–
17–23	Reserved	Reserved UltraScale bits extend through bit 20 and reserved UltraScale+ bits extend through bit 23.	–

eFUSE Security Register (FUSE_SEC) Description

This register contains user programmable bits used to select eFUSE security settings and to enable RSA Authentication, if desired. The following table provides bit descriptions and recommended settings.

Table 5: **eFUSE Control Register Bit (FUSE_SEC) Description**

Bit	Bit Name	Description	Recommend Setting
0	FUSE_SHAD_SEC[0] (CFG_AES_Only)	Only allow encrypted bitstreams. **IMPORTANT!** *If this bit is programmed to 1, the device cannot be used unless the AES key is known. Return material authorization (RMA) returns cannot be accepted and the Vivado Indirect SPI/BPI flash programming flow cannot be used if this bit is programmed. You must have external configuration memories programmed BEFORE you blow this fuse if you intend to use Vivado for this programming.*	Yes (program to 1)
1	FUSE_SHAD_SEC[1]	Force use of AES key stored in eFUSE (BBRAM keys disabled). When this bit is NOT programmed, encryption and the key source can be selected via bitstream options – the FPGA can be configured using an unencrypted bitstream, or a bitstream encrypted with a key value stored in battery-backed RAM (BBRAM) or eFUSE.	No (keep at 0)
2	RSA_AUTH	Force RSA Authentication. **IMPORTANT!** *If this bit is programmed to 1, the device cannot be used unless the RSA key is known. Return material authorization (RMA) returns cannot be accepted and the Vivado Indirect SPI/BPI flash programming flow cannot be used if this bit is programmed. You must have external configuration memories programmed BEFORE you blow this fuse if you intend to use Vivado for this programming.*	Pending customer security requirements
3	FUSE_SHAD_SEC[3]	Disables external JTAG pins.	Pending customer security requirements
4	SCAN_DISABLE	Disable AMD test access.	No (keep at 0)
5	CRYPT_DISABLE	Permanently disable the decryptor.	No (keep at 0)
6	FUSE_BKS_ENABLE	Enable key obfuscation.	Automatically set by Vivado Design Suite
7–31	Reserved	Reserved.	-

- When FUSE_SHAD_SEC[0:1] are NOT programmed:
  - Encryption can be enabled or disabled via the bitstream options.
  - The AES key stored in eFUSE or battery-backed SRAM (BBRAM) can be selected via the bitstream options.
- When FUSE_SHAD_SEC[1:0] are programmed.
  - Only bitstreams encrypted with the eFUSE key can be used to configure the FPGA through external configuration ports.

IMPORTANT! *When FUSE_SHAD_SEC[0] or RSA_AUTH is programmed, only AES encrypted or RSA authenticated bitstreams, respectively, can be used to configure the FPGA through external configuration ports. This precludes device configuration from AMD test bitstreams and AMD pre-built bitstreams. Thus, AMD does not accept return material authorization (RMA) requests or support indirect flash programming for devices that have the FUSE_SHAD_SEC[0] or RSA_AUTH bit programmed.*

Creating an Encryption Key and Encrypted Bitstream

The bitstream generator (write_bitstream), provided with the Vivado tools, can generate encrypted as well as non-encrypted bitstreams. For AES bitstream encryption, set the write_bitstream property to enable bitstream encryption. You can either specify a custom 256-bit key as an input to the bitstream generator, which is the AMD recommendation and the most secure approach, or you can have the Vivado tool generate a pseudo-random key for you (not recommended).

IMPORTANT!

For 3D IC devices, you must assign a NKY file which has unique keys for each SLR. Vivado Hardware Manager issues a critical warning when you try to assign a single AES key for all SLRs or a single AES/IV pair for all SLRs. Both of these situations introduce security vulnerabilities.

See Answer Record 71558, Answer Record 000036543, and Answer Record 000033700 for important updates on using the unique set of encryption keys. See below for an example of a 3D IC NKY file.

The bitstream generator, in turn, generates an encrypted bitstream file (.BIT) and an encryption key file (.NKY). The following table identifies the write_bitstream properties available to be defined in the XDC file and their corresponding descriptions. For a Vivado Integrated Design Environment (IDE) example of key creation and bitstream encryption, see the *Vivado Design Suite User Guide: Programming and Debugging* (UG908).

Table 6: **Write_bitstream Properties**

Write_bitstream Property	Default Values	Possible Values	Description
BITSTREAM.ENCRYPTION.ENCRYPT	No	No or Yes	Encrypts the bistream.
BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT	bbram	bbram or eFUSE	Determines the location of the AES encryption key to be used, either from the battery-backed RAM (BBRAM) or the eFUSE register. ***Note*:** This property is only available when the *Encrypt* option is set to *True*.
BITSTREAM.ENCRYPTION.KEYLIFE	32	4 to 2,147,483,647	The number of 128-bit encryption blocks over which a single key should be used for AES-GCM authenticated bitstreams. A default of 32 increases the bitstream size or bitstream length by about 2x. Used when BITSTREAM.AUTHENTICATION. AUTHENTICATE = No
BITSTREAM.ENCRYPTION.KEY0	Pick	Pick or <256 bit hex string>	Key0 sets the AES encryption key for bitstream encryption. To use this option, you must first set Encrypt to Yes.

Table 6: **Write_bitstream Properties** *(cont'd)*

Write_bitstream Property	Default Values	Possible Values	Description
BITSTREAM.ENCRYPTION.KEYFILE	None	<string>	Specifies the name of the input encryption file (with a .nky file extension). To use this option, you must first set Encrypt to Yes.
BITSTREAM.ENCRYPTION.RSAKEYLIFEFRAMES	8	8 to 2,147,483,647	Specifies how many configuration frames should be used for any given AES-256 key when RSA Public Key Authentication is specified. A value of 8 configuration frames is equivalent to using the key for 246 encryption blocks. Used when BITSTREAM.AUTHENTICATION. AUTHENTICATE = Yes
BITSTREAM.ENCRYPTION.STARTIV0	Pick	Pick or <128 bit hex string>	The initialization vector used to specify the initial GCM count value in the first AES-GCM message. 128-bit hex value. Only first 96 bits of 128 hex values are used. The last 32 bits are discarded.
BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE	Pick	Pick or <128 bit hex string>	Starting obfuscate initial vector (Obfuscate IV0) value. Only first 96 bits of 128 hex values are used. The last 32 bits are discarded.
BITSTREAM.AUTHENTICATION.AUTHENTICATE	No	Yes or No	Indicates whether or not to use RSA authentication. If No, then the authentication intrinsic to AES-GCM is used.
BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE	None	<string>	Specifies the OpenSSL .pem file that contains the key pairs that should be used to sign the RSA-2048 authenticated bitstream.
BITSTREAM.ENCRYPTION.OBFUSCATEKEY	Disable	Enable or Disable	Creates a bitstream whereby the key used to encrypt the bitstream is obfuscated before it is written to eFUSE or battery-backed RAM (BBRAM). This allows the user to provide the device programmer with an obfuscated key rather than the original customer key. The device programmer can then write the obfuscated key to the eFUSE or BBRAM and mark it as obfuscated using the obfuscated-key flag in the selected storage location.

RECOMMENDED: *See Answer Record 000033701 for bitstream parameters for ensuring an adequate level of Side-Channel Analysis (SCA) resistance.*

The following specific XDC file code snippet shows BBRAM key storage and a custom key defined. If you want the tools to generate a pseudo-random key then you need to comment out the `BITSTREAM.ENCRYPTION.KEY0` property. This also shows RSA authentication enabled and the RSA key life frames set to the default of 8. Comment out these to disable the RSA authentication features. All of the properties shown are also selectable and editable via the Edit Device Properties GUI.

```
#Encryption settings
set_property BITSTREAM.ENCRYPTION.ENCRYPT YES [current_design]
#set_property BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT EFUSE [current_design]
set_property BITSTREAM.ENCRYPTION.ENCRYPTKEYSELECT BBRAM [current_design]
set_property BITSTREAM.ENCRYPTION.OBFUSCATEKEY Enable [current_design]
set_property BITSTREAM.ENCRYPTION.KEYLIFE 32 [current_design]
set_property BITSTREAM.ENCRYPTION.KEY0
256'h1234567812345678123456781234567812345678123456781234567812345678
[current_design]
set_property BITSTREAM.ENCRYPTION.STARTIV0
```

```
128'h87654321876543218765432187654321
[current_design]
set_property BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE
128'hABCDABCDABCDABCDABCDABCDABCDABCD[current_design]
set_property BITSTREAM.ENCRYPTION.FAMILY_KEY_FILEPATH
C:/<any directory>/familyKey_us.cfg [current_design]

#Authentication settings
set_property BITSTREAM.AUTHENTICATION.AUTHENTICATE YES [current_design]
set_property BITSTREAM.ENCRYPTION.RSAKEYLIFEFRAMES 8 [current_design]
set_property BITSTREAM.AUTHENTICATION.RSAPRIVATEKEYFILE
{C:config/ultrascale/encryption/encryption_test/encryption_test.runs/impl_1/
rsa.pem}
[current_design]
```

The NKY file generation occurs at the same time as bitstream generation. This file takes the same top-level name as the bit file and is placed in the same implementation directory as the encrypted bitstream. The NKY file format for monolithic devices is:

```
Device <type>;
EncryptKeySelect <type>;
StartIvObfuscate 00000000000000000000000000000000;
RsaPublicKeyDigest
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000;
Key0 0000000000000000000000000000000000000000000000000000000000000000;
StartIV0 00000000000000000000000000000000;
```

For 3D IC devices, the NKY file format `Key` and `StartIV` statements are extended with a second value indicating the targeted SLR index.

```
Device <type>;
EncryptKeySelect <type>;
StartIvObfuscate 00000000000000000000000000000000;
RsaPublicKeyDigest
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000;
Key0 0000000000000000000000000000000000000000000000000000000000000000, 0;
StartIV0 00000000000000000000000000000000, 0;
Key0 0000000000000000000000000000000000000000000000000000000000000000, 1;
StartIV0 00000000000000000000000000000000, 1;
```

Table 7: **Vector Size**

Parameter	Vector Size (in hex)
StartIVObfuscate	128'h
RsaPublicKeyDigest	384'h
Key0	256'h
StartIV0	32'h

For example: (top.nky, monolithic device)

```
Device xcvu095;
EncryptKeySelect BBRAM;
StartIvObfuscate abcdabcdabcdabcdabcdabcdabcdabcd;
RsaPublicKeyDigest
6DF61AA1636F1FF8A35D064CED96918A6D4820D3CAEEFFA47F64F5C58AC2E539CA3A53103C86
450EC576AD595AB3A528;
Key0 1234567812345678123456781234567812345678123456781234567812345678;
```

```
StartIV0 876543218765432187654321000003f8;
Key1 44a619e399910767e68cb81bcbae831bd5d10a96e5a348420e9384eb0df06111;
StartIV1 b152ef23b9481138af45a21a000003d8;
.
.
.
Key4063 ddd9955fae4847a54d10a1c06a52171cf69e3593220018c2cac4ca56bf359f5c;
StartIV4063 7d94e42bf79ab77cce1a1d8300000382;
```

RSA authentication PEM file, example syntax:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAvCMmT6/MM9LxXs7ZxybE4wKACvp0S2EpWy/q+wFkjeev/
oT1EZkyRkeCLWKwLaTUeGxFYeWCVFhpHH7PU9d/5HudIsVr/uJ8k/V7GASsj/
8EL3O+RFoMdpsv6AFFD8desse3svR2d3yWlnrWLKfSd25DLqOg5fHMauV5DwDpsrbUvBf/
ZOW5JWd4iyi0oeK1/Dw/
91AYiJoRWmKt6s3IH1ZkX4OfoXMBJ+SnVgV9NIm59lOb0vd0ZZtNOqo1oX/
Ekn93jwoD1UbHAWN90TfZSIAqsv2c4aeC342jKrHUq4cykK
.
.
.
xuTbhBadZaq8u8TGsXO3oPvI+p2tee5sNNoleJj3/gnkPtF9od5bqo8=
-----END RSA PRIVATE KEY-----
```

Loading the Encryption Key

Both the BBRAM and eFUSE 256-bit symmetric keys can only be loaded onto a device through the JTAG interface using the Vivado Device Programmer tool. For UltraScale devices, this key loading path is write-only to the device. There is no physical data path to read back either key. When a key is written to the device via JTAG, a key integrity check is initiated by writing the expected CRC32 value via JTAG to the device. An actual CRC32 integrity check is calculated on the stored key by the device (internally) and compared to the expected CRC32 that was just received via the JTAG port. A pass/fail type result is then written out by the device to the JTAG port instead of the actual key data to signify integrity status. Removing the physical readback path for the key increases the security of the stored key.

BBRAM key programming solutions include:

- Use of Vivado Device Programmer tool and JTAG cable

  ***Note*:** For BBRAM-based keys, prior to writing the key, the existing key in BBRAM is zeroized (erased and verified).

eFUSE key programming solutions include:

- AMD programming
  - Fully automated in-line ATE flow at AMD test house
  - Secure programming for medium to high volumes – with uniform settings
- Avnet programming
  - Opportunities for security, handling, serialization, and other differentiators
  - Ideal for programming from moderate volumes, down through low volumes

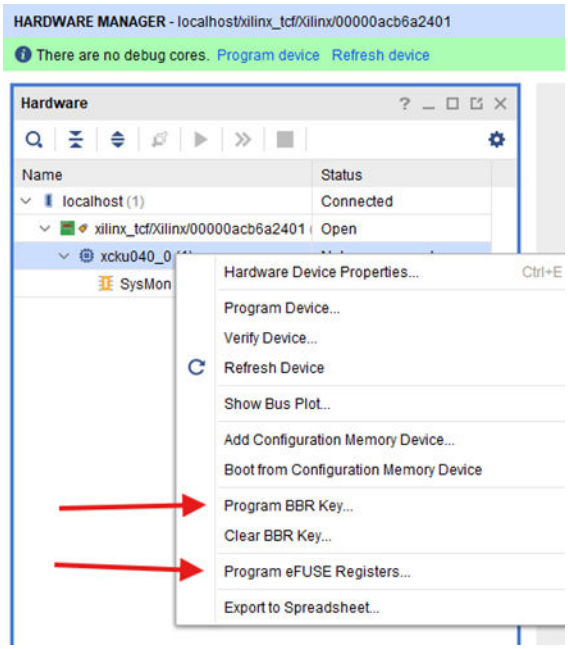
- Customer manufacturing flow
  - Third-party tool, or integrated in design *Internal Programming of BBRAM and eFUSEs* (XAPP1283), using AMD programming technology
  - Ideal for custom requirements including highly-secret information handling

RECOMMENDED: *For the eFUSE solution it is also recommended to take the following precautions for in-system programming of the AES key:*

- Prevent or clear the FPGA of a configured design to minimize power supply noise within the FPGA.
- If possible, stop board-level system clocks to minimize system power supply noise.

After connection to a valid HW target using Vivado Hardware Manager, right-click the UltraScale FPGA to allow selection of either **Program BBR Key...** or **Program eFUSE Registers...**, depending on which storage option you have previously selected (see the following figure).

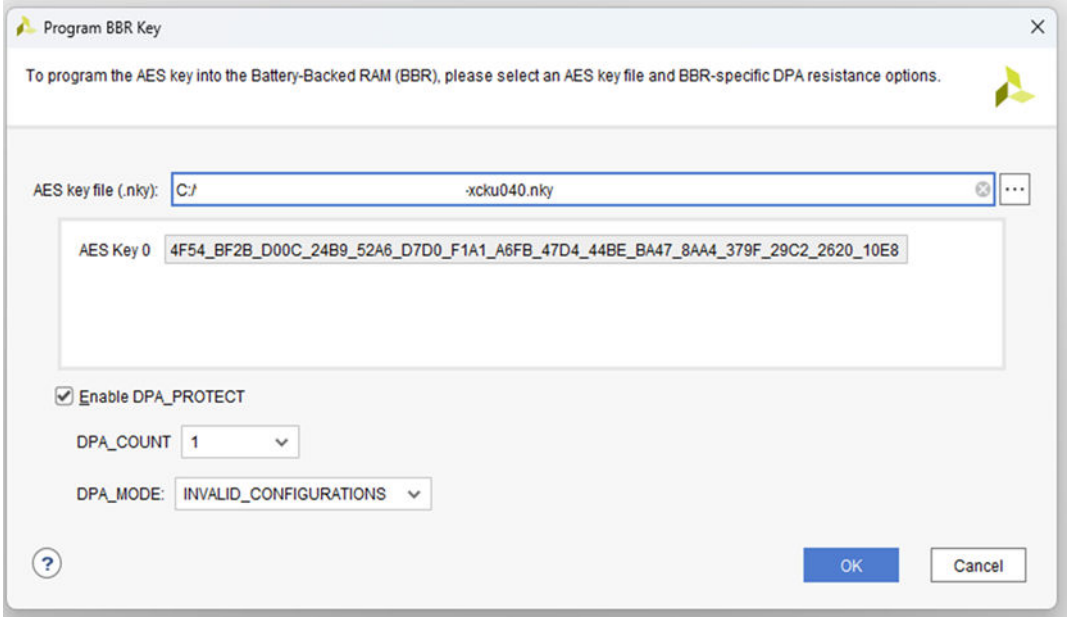
Figure 2: **Vivado HW_Manager Key Programming Selection**



BBRAM

When you select **Program BBR Key...**, you have the ability to browse to the recently generated NKY file in the project directory. After you add the NKY file you also have the ability to double check the key value and verify that this is the AES key you intend to program into the device. (See the following figure.)

Figure 3: **BBRAM Programming GUI**



Note: If the NKY file contains an `KeyObfuscate` field because the BITSTREAM.ENCRYPTION.OBFUSCATEKEY property was enabled prior to write_bitstream, then the obfuscated key flag in the eFUSE or BBRAM is automatically set by Vivado software during programming of the AES-256 key.

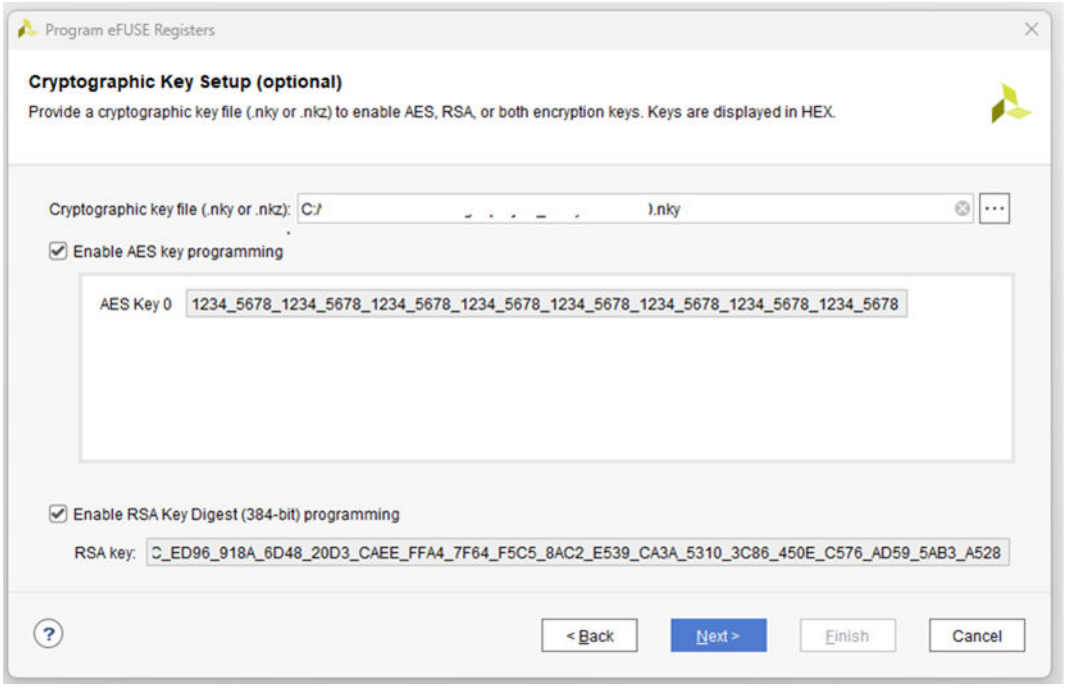
The Enable DPA_PROTECT check box enables the BBRAM Configuration Counting DPA Protection mechanism.

- DPA_COUNT specifies the initial load value for the configuration counter. Once the count reaches 0, the BBRAM is erased.
- DPA_MODE specifies under what conditions the DPA_COUNT should be decremented. The 2 choices are INVALID_CONFIGURATIONS, which is the typical DPA setting, and ALL_CONFIGURATIONS, which decrement the count on every configuration so that the device has a fixed number of configurations to be used.

eFUSE

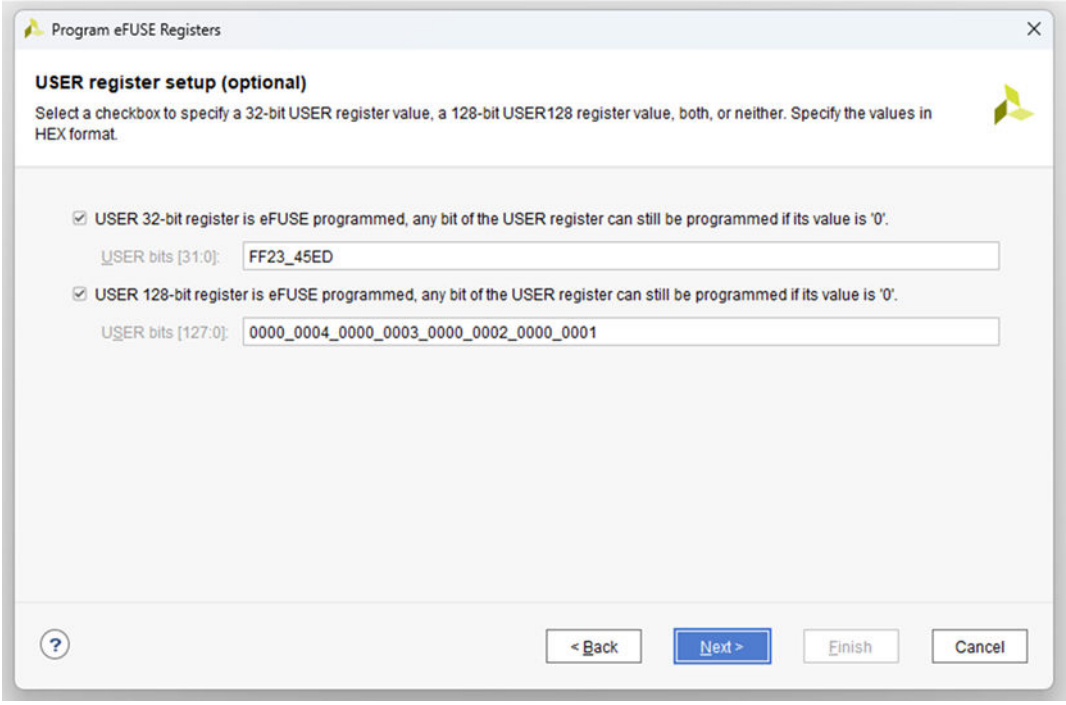
When you select **Program eFUSE Registers...**, a wizard appears and guides you through the process of selecting the NKY file and the various eFUSE registers you want to program. After you add the NKY or PEM file you also have the ability to double check the key values and verify that these are the AES and RSA keys that you intend to program into the device. (See the following figure.)

Figure 4: **eFUSE Programming Cryptographic Key Setup**



The User Register setup screen is shown in the following figure. This allows you to specify a unique 32-bit and/or a 128-bit value to program into the FUSE_USER register bits. These registers are readable from the FPGA logic using the eFUSE_USR primitive.

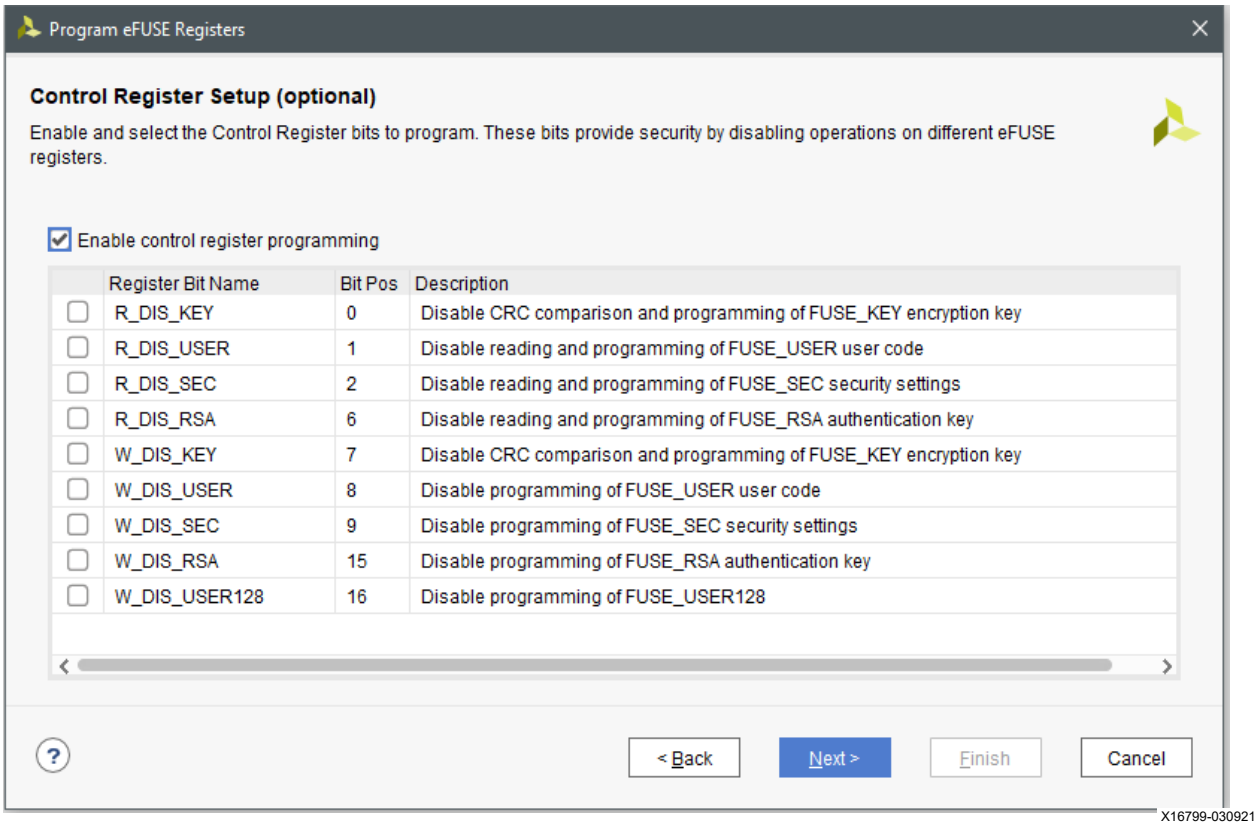
Figure 5: **eFUSE Programming USER Register Setup**



The Control Register setup screen is shown in the following figure. This allows you to select which FUSE_CNTL register bits to program. These bits provide security by disabling read and write operations on different eFUSE Control registers.

***Note*:** See Table 4: eFUSE Control Register Bit (FUSE_CNTL) Description for Control register bit descriptions and recommended settings.

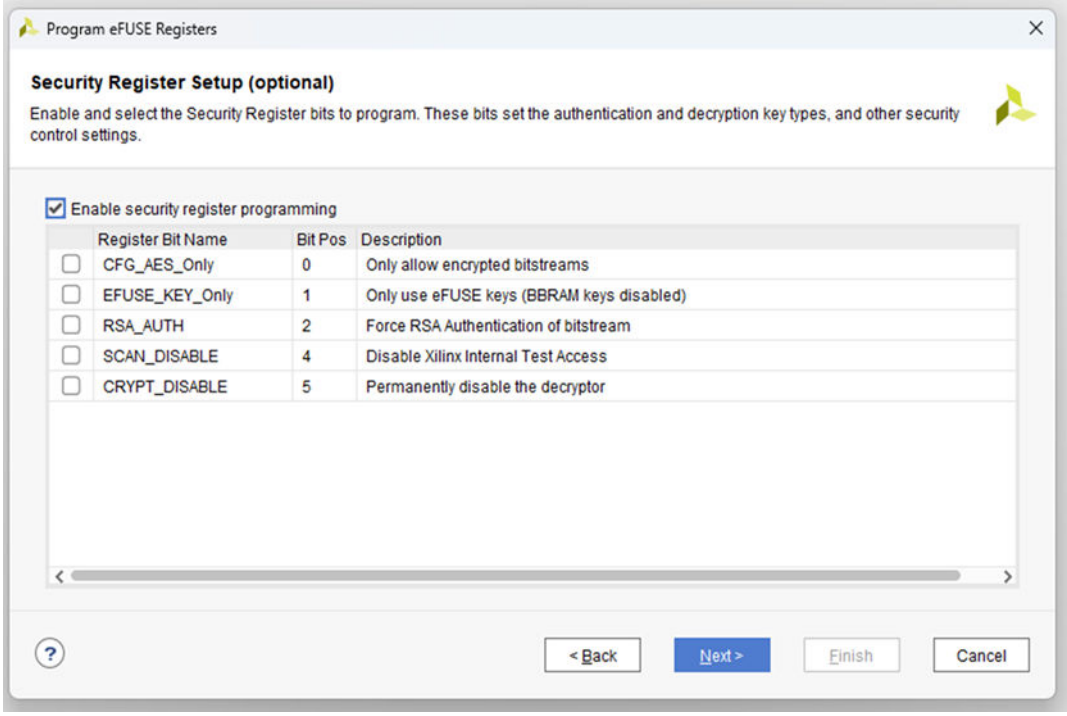
Figure 6: **eFUSE Programming Control Register Setup**



The Security Register setup screen is shown in the following figure. This allows you to select which FUSE_SEC register bits to program. These bits provide additional security by only allowing encrypted bitstreams or enabling RSA authentication.

***Note*:** See Table 5: eFUSE Control Register Bit (FUSE_SEC) Description for Security register bit descriptions and recommended settings.

Figure 7: **eFUSE Programming Security Register Setup**

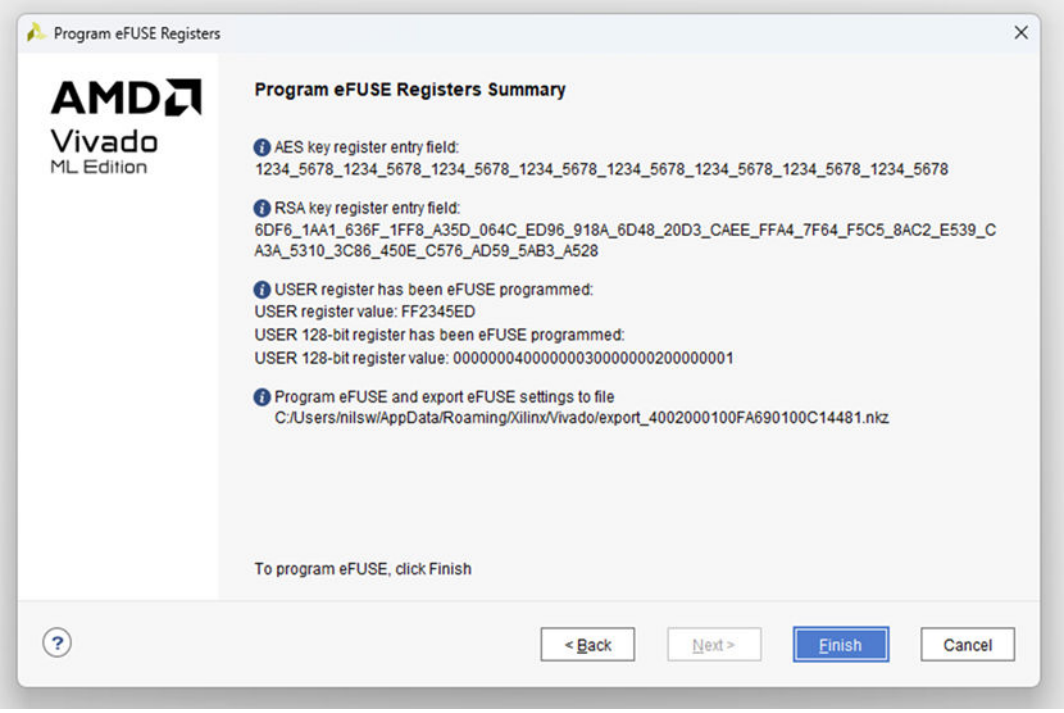


***Note*:** If the NKY file contains an `KeyObfuscate` field because the BITSTREAM.ENCRYPTION.OBFUSCATEKEY property was enabled prior to write_bitstream, then the obfuscated key flag in the eFUSE or BBRAM is automatically set by Vivado software during programming of the AES-256 key.

The last screen, shown in the following figure, is the Summary screen that you can use to verify that the options you have selected are the options that you intend to implement.

IMPORTANT! *Remember that eFUSE registers are one-time programmable and can NOT be changed at a later time.*

Figure 8: **Summary**



Loading the Encrypted Bitstream

After the device has been programmed with the correct encryption key, it can be configured with an encrypted bitstream. After configuration with an encrypted bitstream, it is not possible to read the configuration memory through JTAG or SelectMAP readback, regardless of the bitstream security setting. Although the device holds an encryption key, a non-encrypted bitstream can be used to configure the device (only if the FUSE_SHAD_SEC[0] bit is not programmed) and only after INIT_B or PROGRAM_B is asserted, thus clearing out the configuration memory. In this case the key is ignored. After configuring with a non-encrypted bitstream, readback is possible (if allowed by the write_bitstream security settings). However, the encryption key still cannot be read out of the device, preventing the use of Trojan Horse bitstreams to defeat the UltraScale FPGA encryption scheme.

None of the supported configuration methods are affected by encryption. UltraScale FPGAs do not allow bitstreams to be created with both compression and RSA authentication. An encrypted bitstream can be delivered through any configuration interface: JTAG, Serial, SPI, BPI, SelectMAP, or ICAPE3. After configuration, the device cannot be reconfigured without toggling the PROGRAM_B pin, cycling power, or issuing the IPROG or JPROGRAM instruction. The Fallback and IPROG reconfiguration are enabled even when encryption is turned on. The Fallback and IPROG reconfiguration images can either be encrypted or unencrypted images.Readback is available through the ICAPE3 primitive. None of these events resets the BBRAM key if V_{BATT} or V_{CCAUX} is maintained.

A mismatch between the key in the encrypted bitstream and the key stored in the device causes configuration to fail with the INIT_B pin pulsing Low and then back High if fallback is enabled, and the DONE pin remaining Low. Advanced configuration solutions such as tandem configuration and partial reconfiguration are supported with encrypted bitstreams. Partial bitstreams can be delivered unencrypted to the ICAP, or encrypted (with the same AES key) to any configuration port, as long as the latter has not been explicitly forbidden by the designer. Setting Security Level2 (via set_property BITSTREAM.READBACK.SECURITY Level2 [current_design]) or programming the FUSE_SHAD_SEC[0] "CFG_AES_Only" bit to a `1` prevents partial reconfiguration over external configuration ports.

IMPORTANT! *An RSA authenticated encrypted bitstream must be programmed from one of these configuration interfaces: SelectMAP, SPI or BPI. Direct JTAG programing of RSA authenticated bitstreams with Vivado HW Manager is not supported. For UltraScale FPGA devices and configuration modes that support RSA authentication, see the RSA Authentication section in the UltraScale Architecture Configuration User Guide (UG570).*

eFUSE Programming General Recommendations

Apply the following recommendations for eFUSE programming projects:

- Required: Use Vivado Design Suite 2017.1 or newer.
- Recommended: Set the FPGA configuration mode pins to the JTAG only setting during eFUSE programming, if the board design allows.
- Required: Use separate eFUSE programming operations, for example, separate passes through the Program eFUSE GUI wizard or separate Tcl commands, to program applicable eFUSE values and options in the following order:
  1. Program eFUSE operation pass #1: Program NKY values (AES, RSA) and FUSE_USER values.
  2. Program eFUSE operation pass #2: (If applicable) Program the Security Register (FUSE_SEC) options, except for JTAG disable.
  3. Program eFUSE operation pass #3: (If Applicable) Program the Control Register (FUSE_CNTL) options, except for the W_DIS_CNTL (write-disable control register).

     Note: If you need to program the Security Register JTAG Disable option in the final step (5), do not program the Control Register W_DIS_SEC option.
  4. Program eFUSE operation pass #4: (If Applicable) Program the Control Register W_DIS_CNTL (write-disable FUSE_CNTL register. See *Vivado Design Suite User Guide: Programming and Debugging* (UG908).
  5. Last program eFUSE operation pass: (If Applicable) Program the Security Register JTAG Disable. See *Vivado Design Suite User Guide: Programming and Debugging* (UG908).
- Recommended: For the first programmed device, validate the eFUSE results after each of the preceding steps, and then re-validate the eFUSE results after completing all steps to ensure that the final results from a complete eFUSE programming procedure is as expected.
- If AES and/or RSA values are programmed, then validate that the device loads an AES-encrypted and/or RSA-signed bitstream successfully.

- If FUSE_USER value is programmed, then validate that you read the correct JTAG FUSE_USER and/or EFUSE_USER primitive value.
- If FUSE_SEC settings are programmed, then validate the correct device behavior for the chosen settings.
- If FUSE_CNTL settings are programmed, then check the resulting REGISTERS.EFUSE.FUSE_CNTL value in Vivado to verify the settings, and check that the read-protected REGISTER.EFUSE.* registers in Vivado do not show your actual values.

  ***Note*:** It is expected that Vivado will show some FUSE_CNTL reserved bit locations which are previously programmed to '1' by the AMD factory.
- Verify that you can, or cannot, access the device JTAG, depending on your choice for step 5.

Hardware Verification

You will most likely want validation that the AES key was programmed into either the BBRAM or eFUSE bits properly. The following is a check-list of verification steps:

1. Generate bitstreams using Vivado 2017.1 or later: An unencrypted bitstream, an encrypted bitstream with your personalized key, an encrypted bitstream with an all-ones key, and an encrypted bitstream with an all-zeros key.
2. Review the generated bitstreams to validate that encryption occurred.
3. Check Hardware: Use Vivado Device Programmer to connect to the FPGA, download the unencrypted bit file via JTAG. Verify that the design functions as expected.
4. Test the FPGA decryptor: Download the encrypted .bit file with the all-zeros key (for eFUSE).
5. Program the AES key via JTAG following the recommendations in the previous section. (If using eFUSE, first perform steps 5 and 6 with the BBRAM key as a validation check; then, if working as expected, program the eFUSE for final test.)
6. Test key: Download the encrypted .bit file with your personalized key.
7. Test key: Download encrypted .bit file with all-zeros key (expect failure).
8. Test key settings: Download the unencrypted .bit file (results can vary depending on security settings).
9. Check key security: Check that the key is read-protected.

Conclusion

This application note describes the UltraScale FPGA AES encryption and authentication standards. It presents you with advantages and disadvantages of the different key storage options available. Most importantly, it is an easy *how* to guide to create an encrypted bitfile along with encryption and authentication keys, and to program these files into an UltraScale FPGA using Vivado software.

Finding Additional Documentation

Technical Information Portal

The AMD Technical Information Portal is an online tool that provides robust search and navigation for documentation using your web browser. To access the Technical Information Portal, go to https://docs.amd.com.

Documentation Navigator

Documentation Navigator (DocNav) is an installed tool that provides access to AMD Adaptive Computing documents, videos, and support resources, which you can filter and search to find information. To open DocNav:

- From the AMD Vivado™ IDE, select **Help** → **Documentation and Tutorials**.
- On Windows, click the **Start** button and select **Xilinx Design Tools** → **DocNav**.
- At the Linux command prompt, enter `docnav`.

***Note*:** For more information on DocNav, refer to the *Documentation Navigator User Guide* (UG968).

Design Hubs

AMD Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In DocNav, click the **Design Hubs View** tab.
- Go to the Design Hubs web page.

References

These documents provide supplemental material useful with this guide:

1. *Advanced Encryption Standard (AES)* (FIPS PUB 197)
2. *The Galois/Counter Mode of Operation (GCM)* (Specification)
3. *UltraScale Architecture Configuration User Guide* (UG570)
4. *Vivado Design Suite User Guide: Programming and Debugging* (UG908)
5. *Developing Tamper-Resistant Designs with UltraScale and UltraScale+ FPGAs* (XAPP1098)
6. *OpenSSL* (www.openssl.org)
7. *Internal Programming of BBRAM and eFUSEs* (XAPP1283)
8. *UltraScale FPGA RSA Authentication and Supporting Configuration Modes* (XCN15038)
9. *Design Advisory for UltraScale/UltraScale+ FPGA Series: RSA Authentication Vulnerability (JustSTART)* (000036039)

10. *Design Advisory for UltraScale RSA Authentication - UltraScale devices that use RSA authentication will fail bitstream authentication when smaller configuration interface widths are used* (AMD Design Advisory 65792)

Revision History

The following table shows the revision history for this document.

Section	Revision Summary
05/22/2025 Version 1.8	
eFUSE Security Register (FUSE_SEC) Description	• Updated Recommend Setting for Bit 0 in Table 5. • Added Bit 3 entry to Table 5.
06/24/2024 Version 1.7	
Summary	Added recommendation to enforce encryption to important note.
Bitstream Authentication	Clarified the list item about RSA-2048 Authentication.
RSA Authentication	• Removed descriptions of using RSA authentication independent of bitstream encryption. • Added important note.
Encrypted Bitstream Implementation Overview	Clarified authentication choices in Step 2.
Creating an Encryption Key and Encrypted Bitstream	• Changed instances of SSI to 3D IC devices. • Clarified note and changed to important note. Added reference to AR 000036543 to important note. • Updated NKY file format description to NKY file format for monolithic devices. • Added 3D IC NKY file format example. • Added monolithic device to vector size example.
Loading the Encryption Key	Updated Figure 2.
BBRAM	Updated Figure 3.
eFUSE	• Added important note. • Updated Figure 4, Figure 5, Figure 7, and Figure 8.
References	Added reference to Design Advisory 000036039.
02/10/2023 Version 1.6	
• Updated description for Bits[0:2, 6] in Table 4. • Cleaned up Encryption settings code in Creating an Encryption Key and Encrypted Bitstream. • Added Table 7.	
03/16/2022 Version 1.5	
• Added security issue description in Summary. • Added Design Advisory 76171 in Key Rolling. • Updated and added recommended notes in Creating an Encryption Key and Encrypted Bitstream. • Added 96 bits description to BITSTREAM.ENCRYPTION.STARTIV0 and BITSTREAM.ENCRYPTION.STARTIVOBFUSCATE in Table 6.	

Section	Revision Summary
03/26/2021 Version 1.4	
• Updated FUSE_RSA and FUSE_KEY contents and descriptions in Table 3. • Updated Table 4. • Updated bits 0-2 in Table 5. • Added recommended note and coding in Creating an Encryption Key and Encrypted Bitstream. • Updated Possible Value for BITSTREAM.ENCRYPTION.RSAKEYLIFEFRAMES in Table 6. • Updated Figure 6. • Updated Fallback and IPROG description in Loading the Encrypted Bitstream.	
10/12/2018 Version 1.3	
Clarified Obfuscated Keys. Changed BITSTREAM.ENCRYPTION.STARTIV0 hex value from 32 bits to 128 bits in Table 6 and in line 14 in the file code snippet. Added content to line 16 in the file code snippet. Clarified eFUSE programming solutions bulleted list under Loading the Loading the Encryption Key. Replaced [Ref 7] under References.	
08/15/2018 Version 1.2	
Updated eFUSE register descriptions in Table 3. Clarified R_DIS_Key description (bit 0) and added W_DIS_USER_128 (bit 16) in Table 4. Added FUSE_BKS_ENAB (bit 6) and reserved bits 7-31 in Table 5. Clarified write_bitstream properties in Table 6. Added eFUSE Programming General Recommendations.	
04/13/2017 Version 1.1	
Added reference to Design Advisory 68832 under Summary.	
06/02/2016 Version 1.0	
Initial release.	N/A

Please Read: Important Legal Notices

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.

Copyright