# Securing Titanium Applications

By design, an FPGA can be reconfigured to perform any function you program into it. But if left undefended, a malicious user could potentially subvert the FPGA to perform some other undesired function. To be secure from this type of attack, the FPGA needs to have an anti-tampering function. One method to prevent tampering is by authenticating the configuration bitstream to ensure that the FPGA can only use the bitstream file you created. Another potential issue is intellectual property theft. After working hard to create your hardware product, the last thing you want is for someone to reverse engineer the FPGA's functionality to steal your work and potentially create a competing product from it. To protect your IP, the FPGA needs to support bitstream encryption so you can be sure that your design is secure.

Titanium FPGAs have built-in authentication and encryption security features to help keep your applications safe. You can use one or the other or both to secure your bitstream.

*Titanium FPGAs have built-in security to help prevent tampering and to protect your design.*

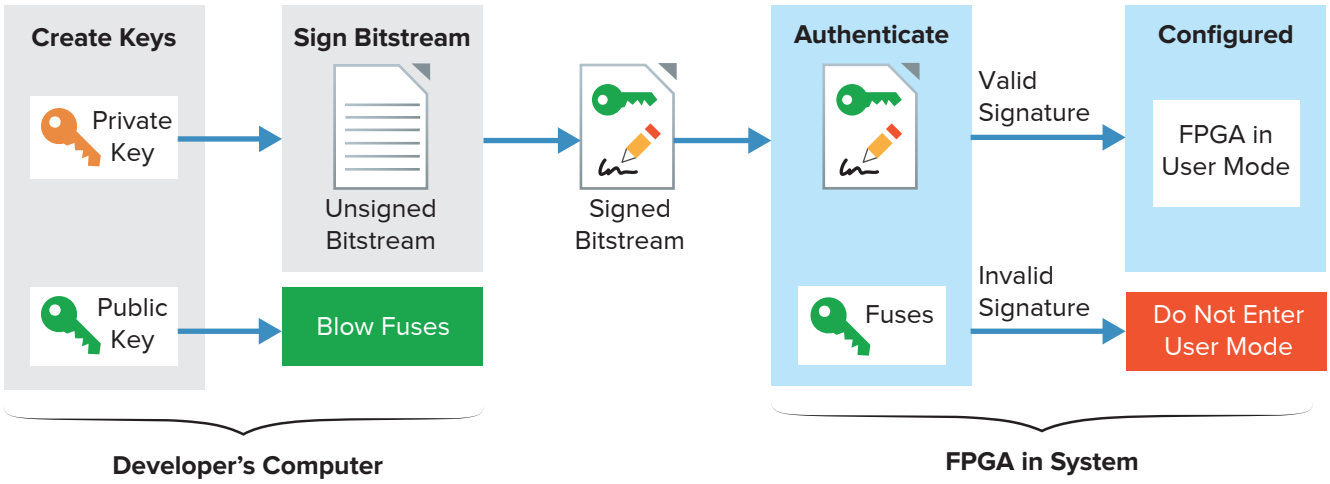
## Authentication

Titanium FPGAs support asymmetric bitstream authentication with the RSA-4096 algorithm. You create a public/private key pair and sign the bitstream with the private key. Then, you save the public key data into fuses in the FPGA. During configuration, the FPGA validates the signature on the bitstream using the public key.

If the signature is correct, the FPGA knows that the bitstream came from a trusted source and has not been altered by a third party. The FPGA continues configuring normally and goes into user mode. If the signature is not correct, the FPGA stops configuration and does not go into user mode.

### Figure 1 Bitstream Authentication

The FPGA has fuses that contain data about the RSA public key. The FPGA uses the public key to validate the signature. The private key stays private.



## Encryption

To ensure that your intellectual property remains safe, Titanium FPGAs use symmetric encryption with a 256-bit key and the AES-GCM-256 algorithm. You generate the key and encrypt the bitstream with it. Then, you save the key into the FPGA by blowing fuses. During configuration, the FPGA uses the stored key to decrypt the bitstream. The key cannot be extracted from the fuses, so a malicious user cannot recover the plaintext bitstream.

## Efinity® Software Makes It Easy

The Efinity® software includes built-in tools that make it easy for you to generate keys, blow fuses, and encrypt bitstreams.

### Generating Keys

The Efinity software has a built-in key generator tool that you use to generate the AES key (**.bin**) and/or RSA keys (**.bin** public and **.pem** private). Optionally, you can generate these keys using strong randomization if you need that feature. Additionally, because the RSA **.pem** file format is an industry standard, you can use an RSA private key file generated from a third-party tool. Make sure to save the keys in a safe place! The software also creates a serial vector format (**.svf**) file that you use to blow the fuses.

### Securing the Bitstream

To secure the bitstream, you simply point to the key files in the Efinity Project Editor (Bitstream Generation tab) and turn on authentication and/or encryption. During compilation, the Efinity software automatically secures the bitstream with the keys. You can then program the FPGA with the secured bitstream as you normally would.

### Blowing Fuses

You use a JTAG SVF player and the **.svf** file to blow the fuses. If you are working with an Efinix development kit, you use the Efinity software's built-in JTAG SVF Player and a USB cable. In the manufacturing environment, you can use any JTAG SVF player and just a JTAG cable. After you blow the fuses, power cycle the board or trigger CRESET_N for the fuse settings to take effect. Blowing fuses is a permanent action. You cannot change the key values once you have blown them into the fuses. So be very sure you are ready when you take that step!

## Verifying the Security

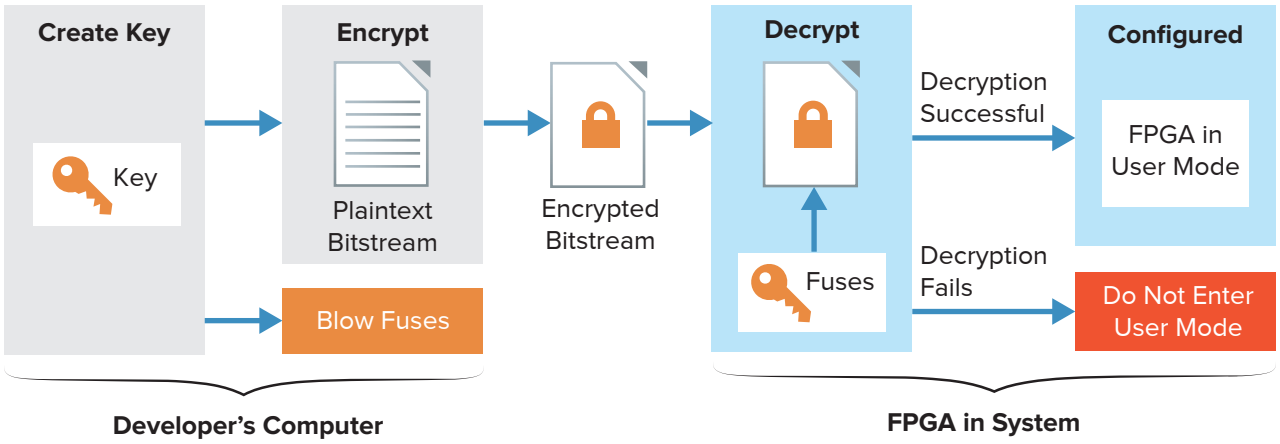
After you have completed this whole process, you may want to check that the security function is working correctly. With a secured bitstream and the correct keys, the FPGA simply goes into user mode as usual. But, you can test what might happen if there is a problem with the bitstream or keys. For example, you can sign the bitstream with an incorrect **.pem** file or encrypt the bitstream with an incorrect AES key. In either case, the FPGA should not go to user mode. Similarly, if you are using authentication, if you try to send an unsigned bitstream to the FPGA, it should not go into user mode. Finally, you can use the Efinity Programmer to check whether the FPGA security engine processed the RSA and/or AES algorithms, and whether the algorithms were successful.

## Your Application Is Secure

The Titanium family's embedded security engine uses simple and well-proven cryptographic techniques to ensure that your intellectual property is secure and that only good bitstreams from trusted sources can configure the FPGAs. In this way, systems containing Titanium FPGAs are kept secure and free from malicious attacks.

### Figure 2 Bitstream Encryption

The FPGA has fuses that store the 256-bit key used to decrypt the bitstream.








EFINIX®
900 Lafayette St, Suite 406
Santa Clara, CA 95050
www.efinixinc.com