ARM® Cortex®-M23

32-bit Microcontroller

NuMicro® Family
Crypto Tool
User Manual

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

Table of Contents

1 OVERVIEW 3
2 CRYPTO TOOL INTRODUCTION 4
2.1 Key Generator 4
2.2 ECDSA 6
2.3 ECDH 9
2.4 AES 11
2.5 SHA 13
2.6 Packer 15
3 REVISION HISTORY 16

1 OVERVIEW

The CryptoTool is an assist tool for M2351 for cryptography calculation. Its features are listed below:

- Generate private key with 256 bits width
- Support ECC NIST P-256 includong public key calculation, ECDSA and ECDH
- Support AES CFB mode with 256 bits width
- Support SHA-256
- Pack firmware for OTA firmware package

2 CRYPTO TOOL INTRODUCTION

CryptoTool include CryptoTool.exe and libcrypto-1_1.dll, user should check these two files are in the same layer directory. Just double click "**CryptoTool.exe**" can open the CryptoTool. The following will demonstrate how to use the CryptoTool.

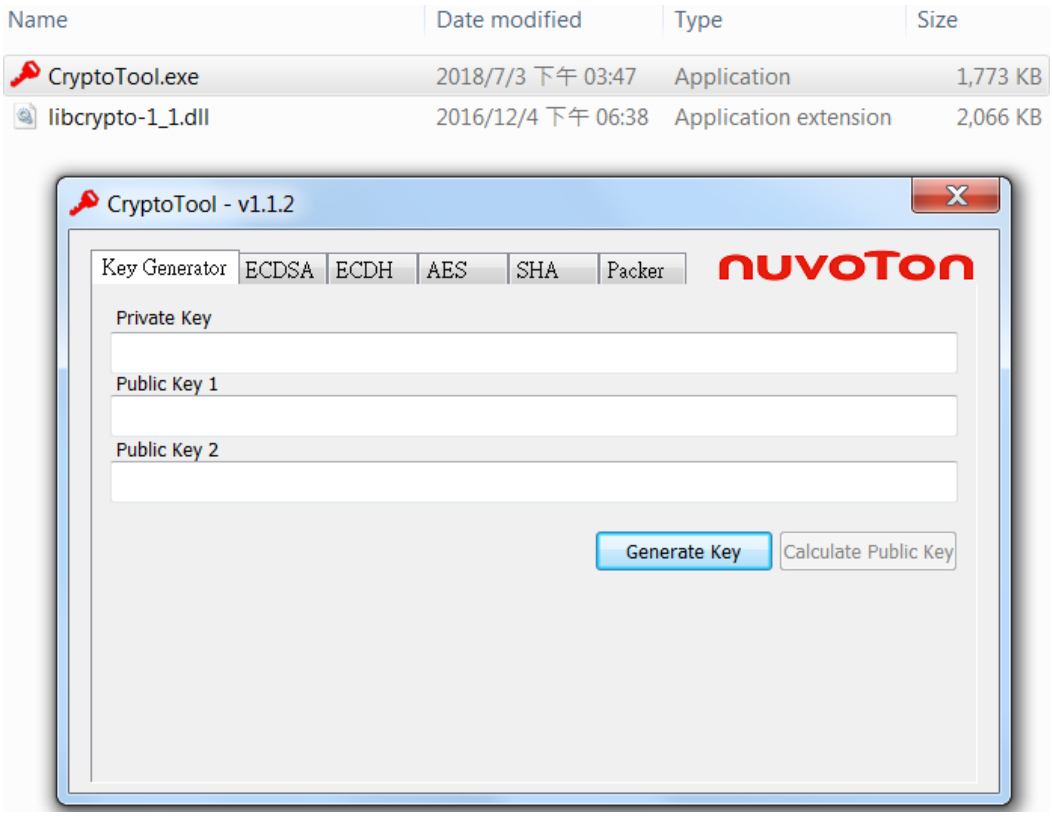


Figure 2-1 Directory of CryptoTool

2.1 Key Generator

Choose "**Key Generator**" in the CryptoTool. In this tab, tool can generate random key pair by ECC calculation. Just click "**Generate Key**" button if you want to generate both private key and public keys automatically. Then the private key and the public keys will show in "**Private Key**", "**Public Key 1**" and "**Public Key 2**" respectively. The public keys are calculated from the specified private key with ECC NIST P-256 curve.

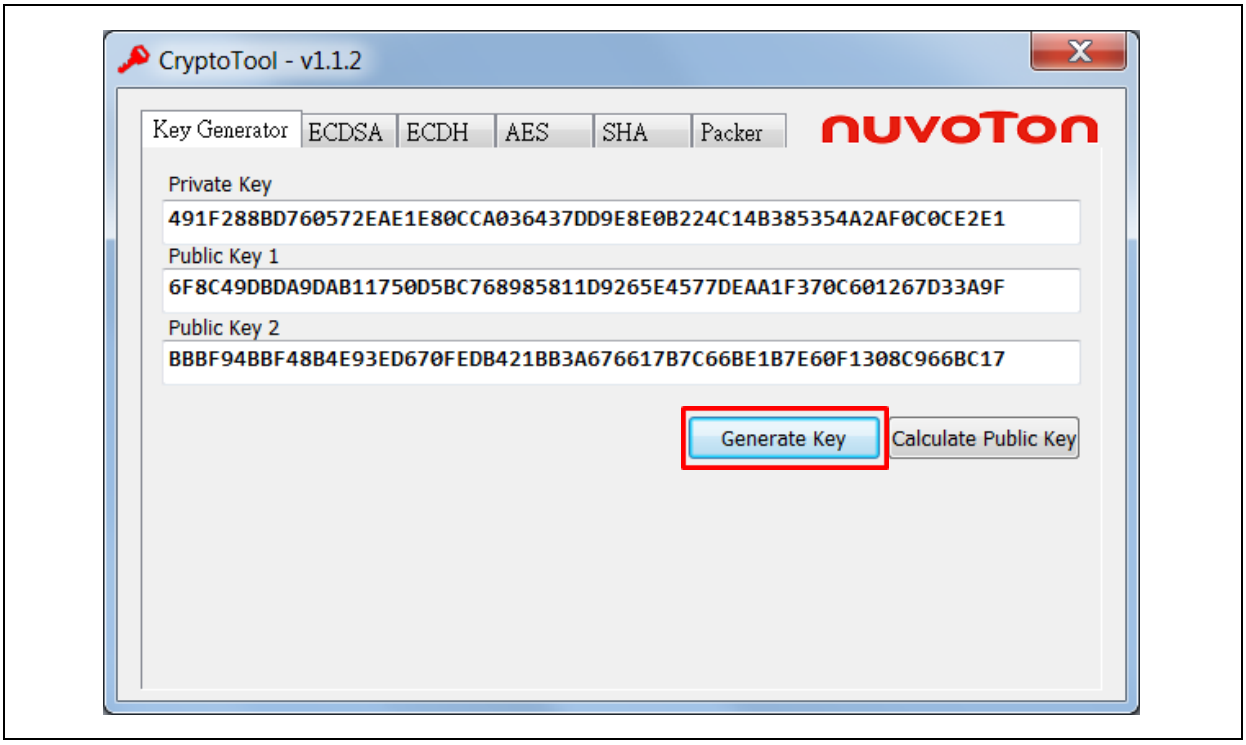


Figure 2-2 Generate Private Key and Public Keys

If you have your own private key, you can enter it into "**Private Key**" field and click "**Calculate Public Key**" button, then the public keys will be calculated with NIST P-256 curve and show in "**Public Key 1**" and "**Public Key 2**".

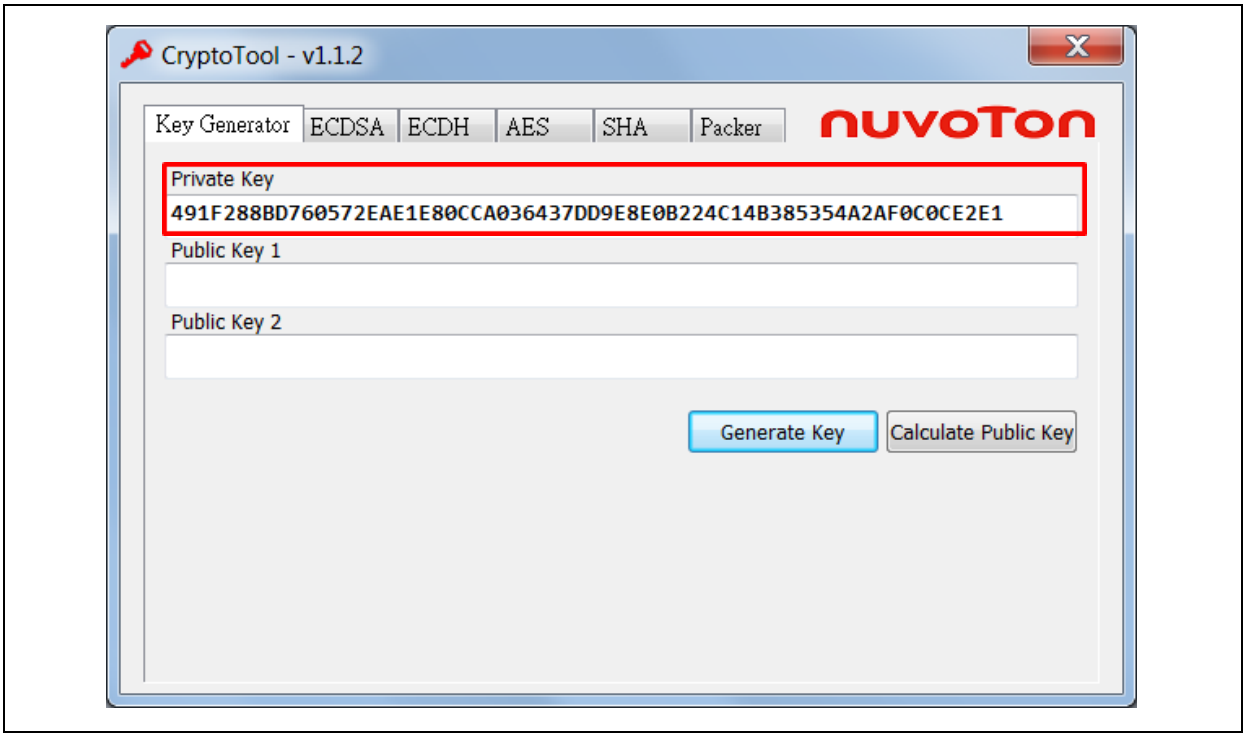


Figure 2-3 Enter Private Key Manually

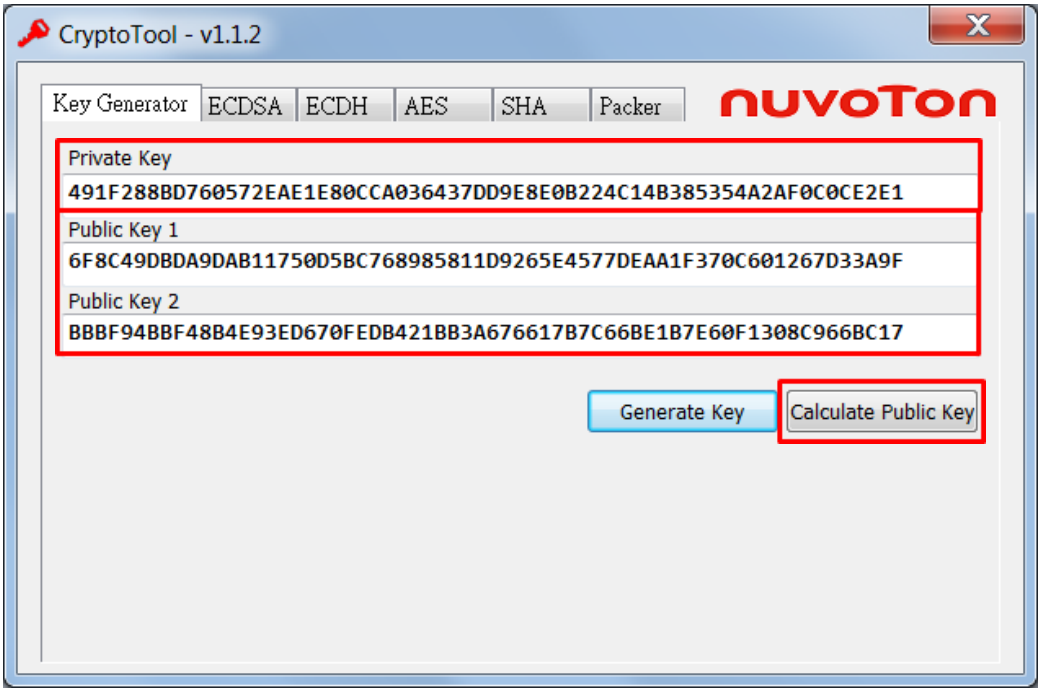


Figure 2-4 Calculated Public Keys by Specified Private Key

2.2 ECDSA

Choose "**ECDSA**" in the CryptoTool. In this tab, tool can calculate a signature according to the private key and the message, and can verify the signature to confirm that whether the message is legal or not. To sign a message signature, user needs to provide private key, and message hash value. To verify the message signature, user needs to enter the public keys, message hash value and the message signature. The NIST P-256 curve is used in ECDSA calculation here.

For example, if *A* wants to send a message to *B*, *A* needs to specify its own private key in "**Private Key (256 bits)**" and put the hash value that is derived from calculating message by SHA in "**Message Digest (SHA256)**" and then click "**Sign**" button, then the signature will show in "**R (256 bits)**" and "**S (256 bits)**".

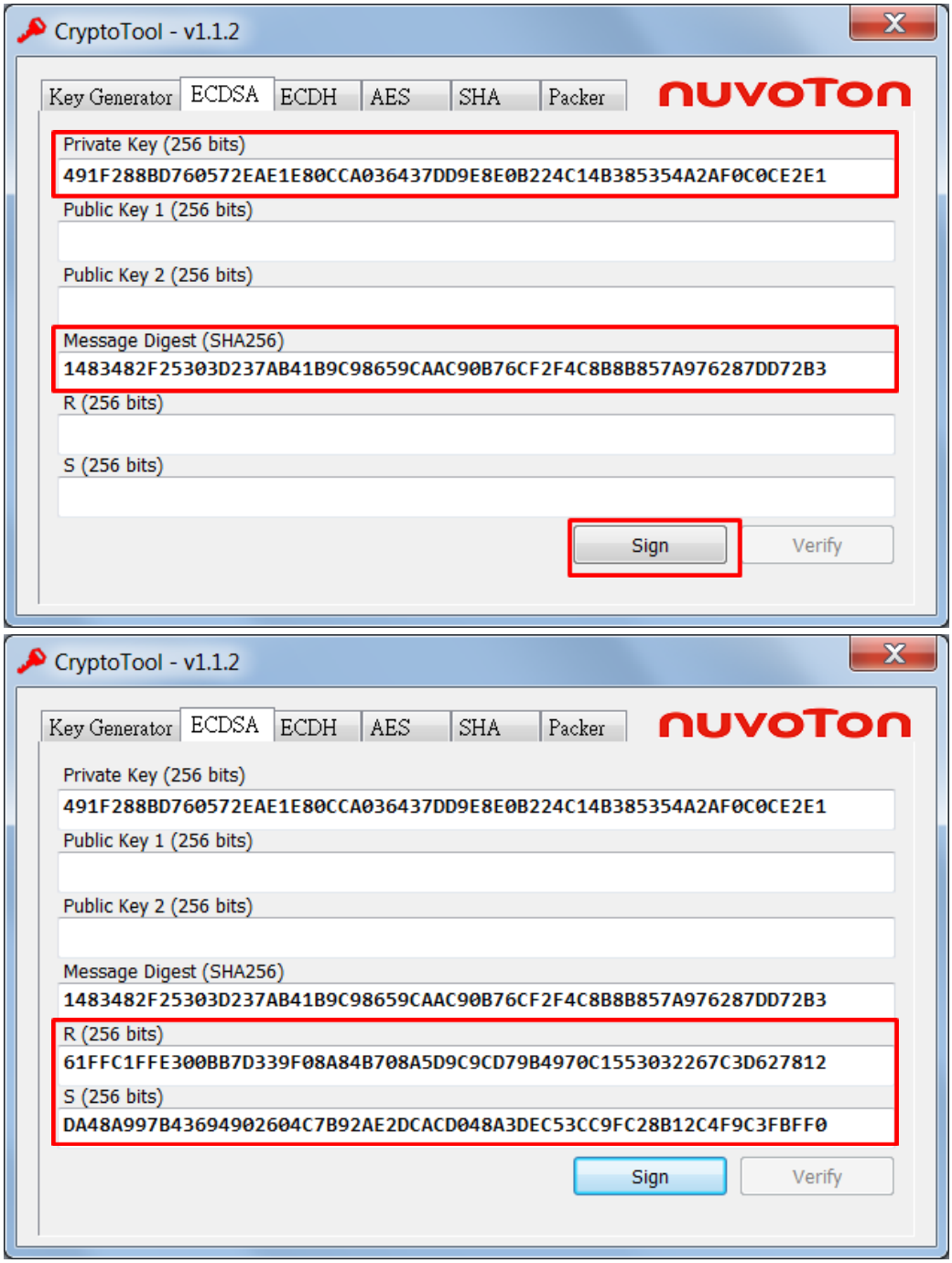


Figure 2-5 Sign a Signature

After that, *B* can specify the public keys obtained from *A* in "**Public Key 1 (256 bits)**" and "**Public Key 2 (256 bits)**", put the received hash value in "**Message Digest (SHA256)**" and specify the received signature in "**R (256 bits)**" and "**S (256 bits)**" respectively, then click "**Verify**" button to check the signature verification is passed or failed. The signature verification is passed will show "**PASS!**" and will show "**FAIL!**" while the signature verification is failed.

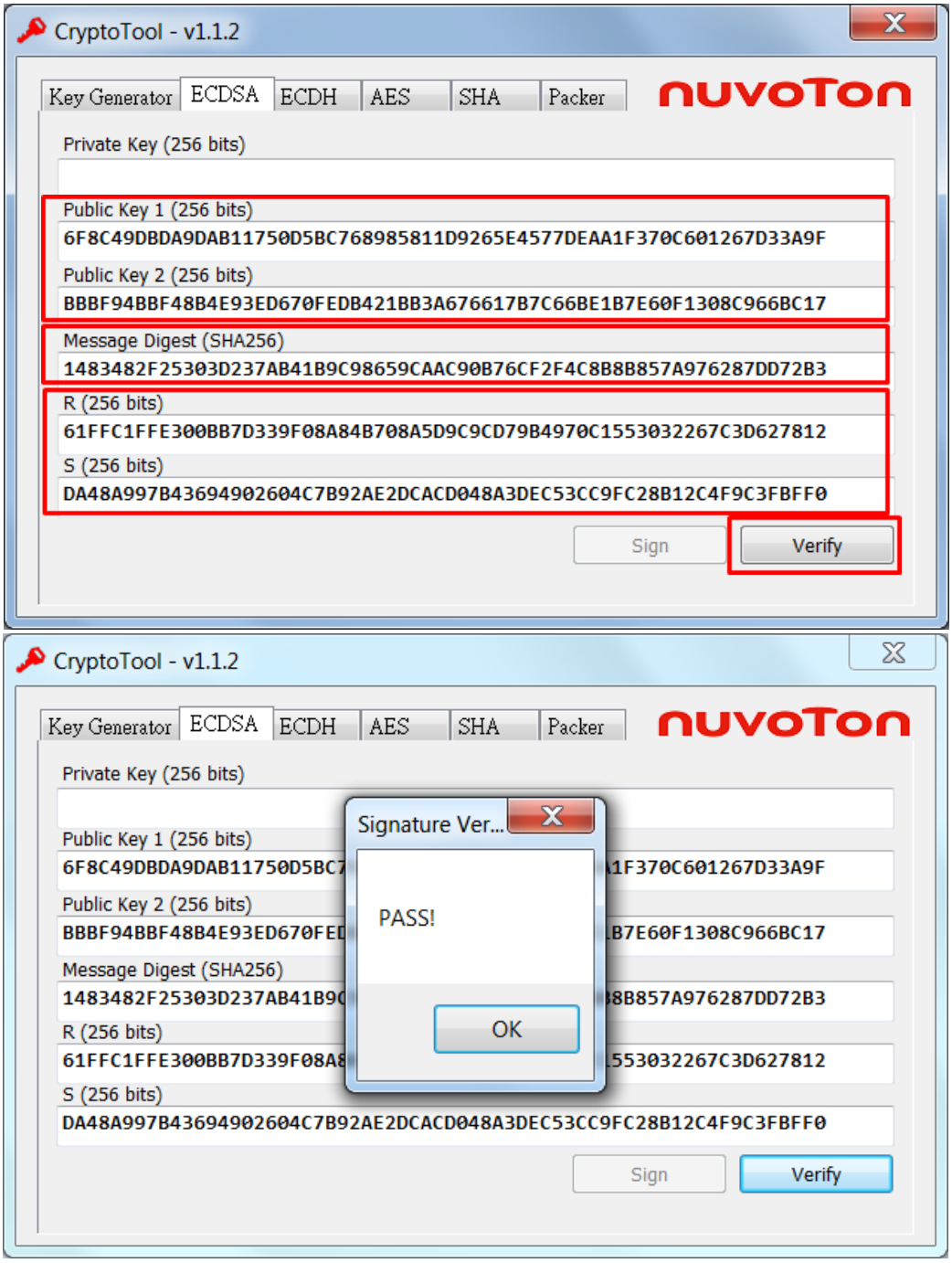


Figure 2-6 Signature Verification is Passed

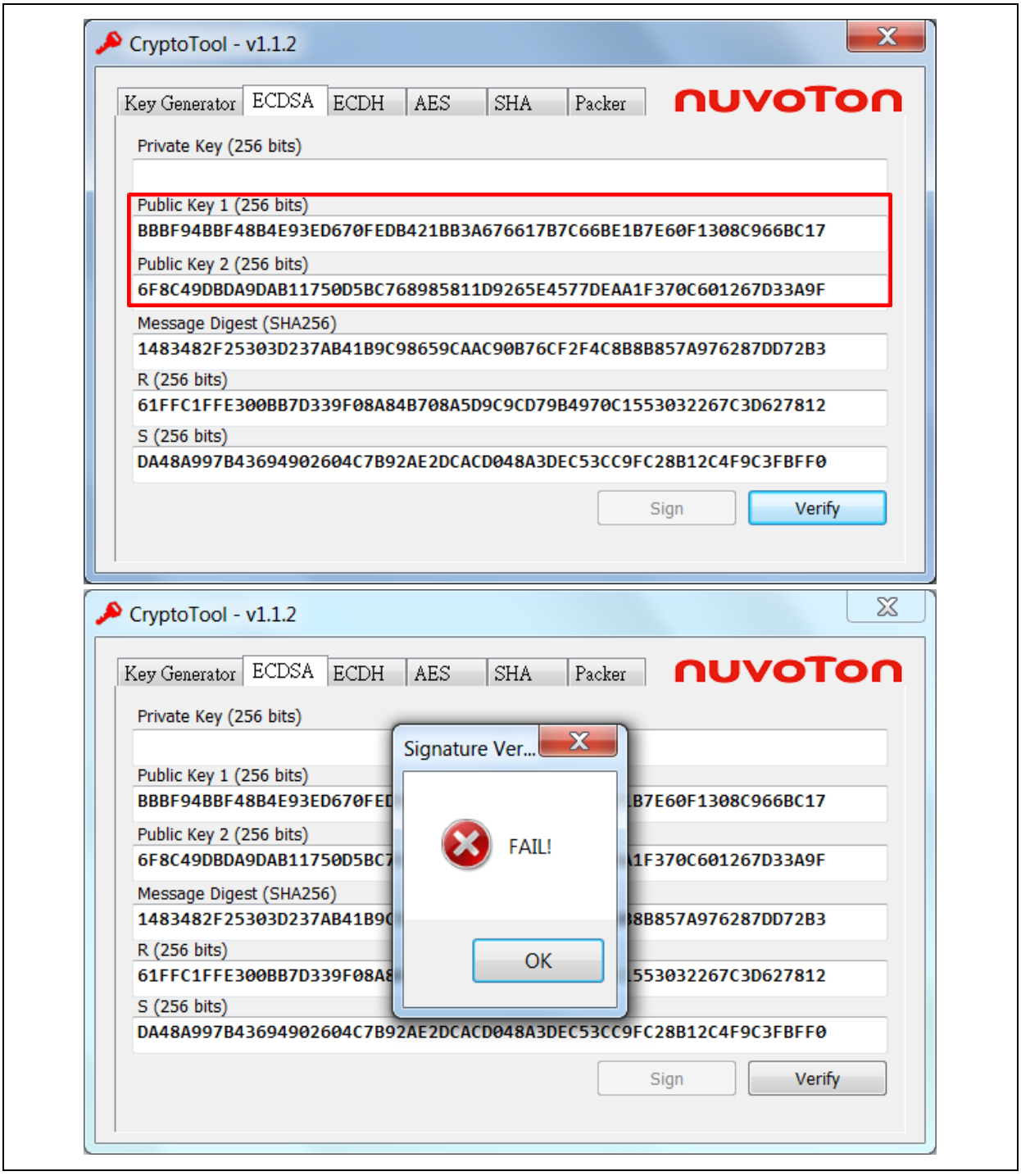


Figure 2-7 Signature Verification is Failed when Public Keys Error

2.3 ECDH

Choose "**ECDH**" in the CryptoTool. In this tab, tool can calculate a shared key according to private key and public keys which derived from another private key.

For example, if *A* and *B* want to transfer data to each other safely, first of all, they must create a shared key only they know.

A needs to specify *A*'s private key in "**Private Key (256 bits)**", specify *B*'s public keys in "**Public Key 1 (256 bits)**" and "**Public Key 2 (256 bits)**" respectively and click "**Calculate Share Key**" button to calculate a shared key in "**Share Key (256 bits)**" field.

In *B* side, it needs to set "**Private Key (256 bits)**" as *B*'s private key and set "**Public Key 1 (256 bits)**" and "**Public Key 2 (256 bits)**" as *A*'s public key pair. Then, click "**Calculate Share Key**" button to calculate a shared key in "**Share Key (256 bits)**" field.

The shared key calculated from *A* and *B* should be the same.

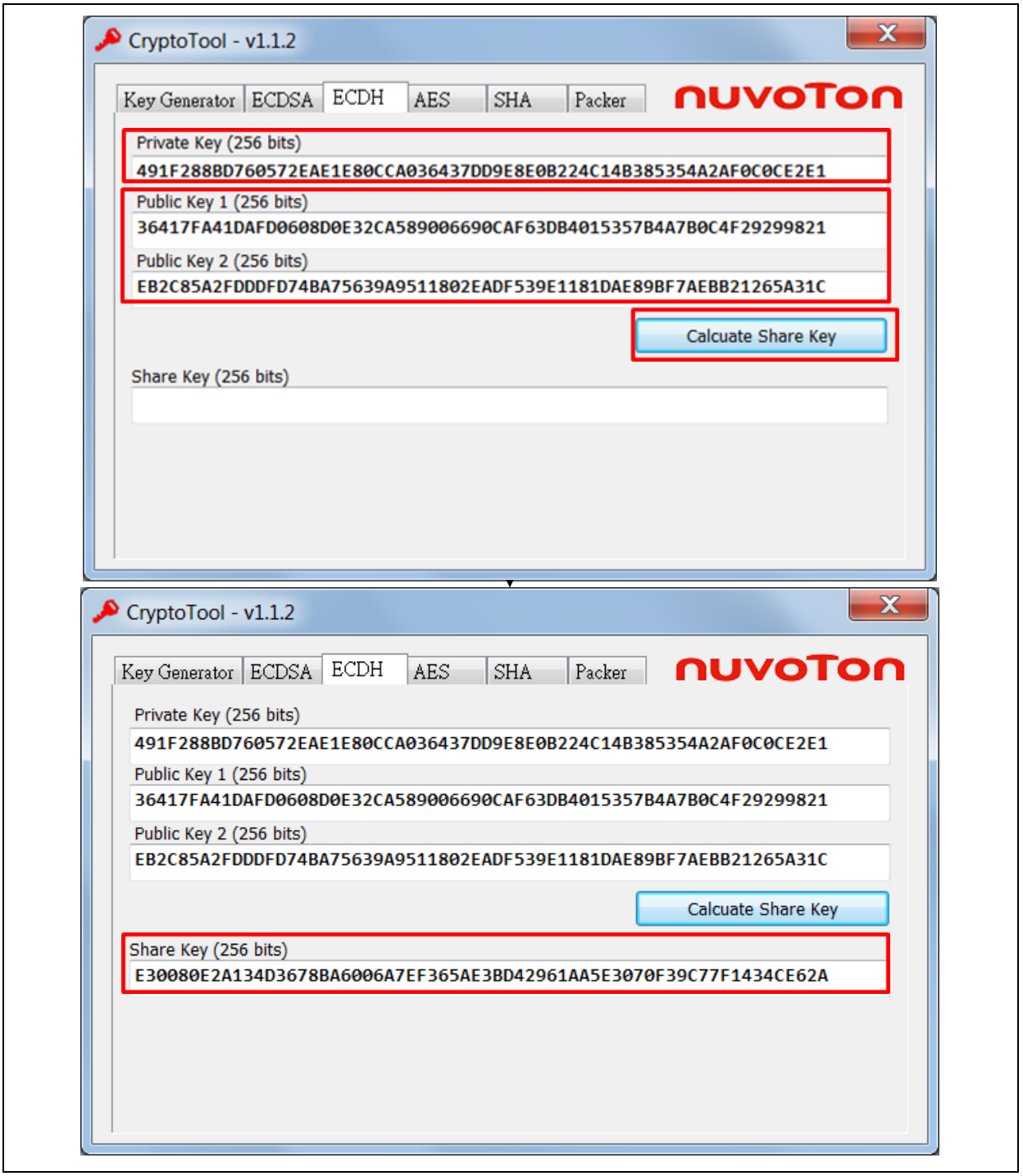


Figure 2-8 Calculate Shared Key by *A*'s Private Key and *B*'s Public Keys

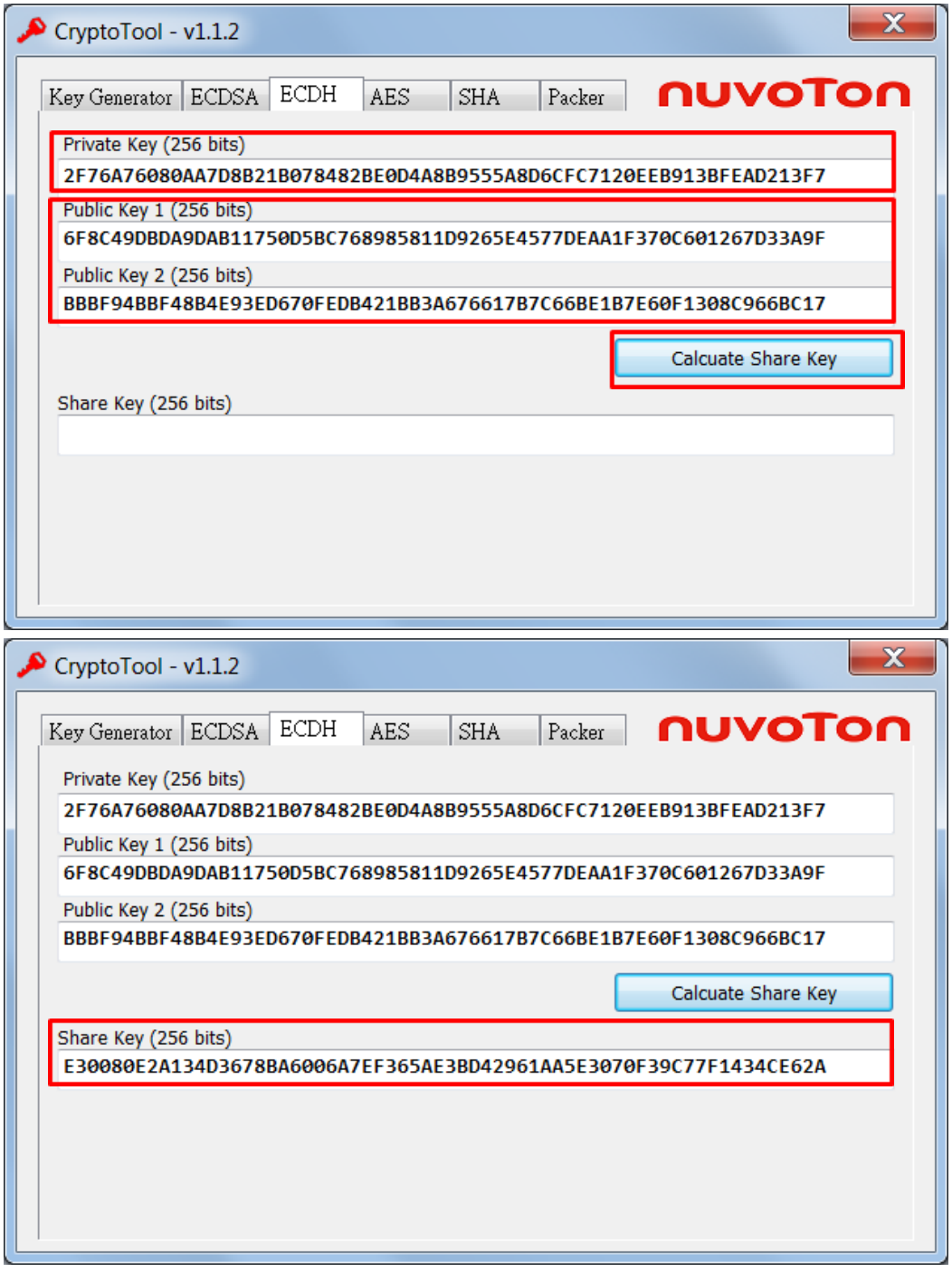


Figure 2-9 Calculate Shared Key by *B*'s Private Key and *A*'s Public Keys

2.4 AES

Choose "**AES**" in the CryptoTool. In this tab, tool can encrypt and decrypt a file by 256-bit AES CFB mode according to the key defined by user.

To encrypt input file by AES, user needs to enter AES key into "**AES Key (256 bits)**" field, enter 128 bits initialization vector (IV) into "**Initialization Vector (128 bits)**" field, specify input file name in "**Input File**" and output file name in "**Encrypt File**". Then click "**Encrypt**" button to encrypt input file by AES CFB mode.

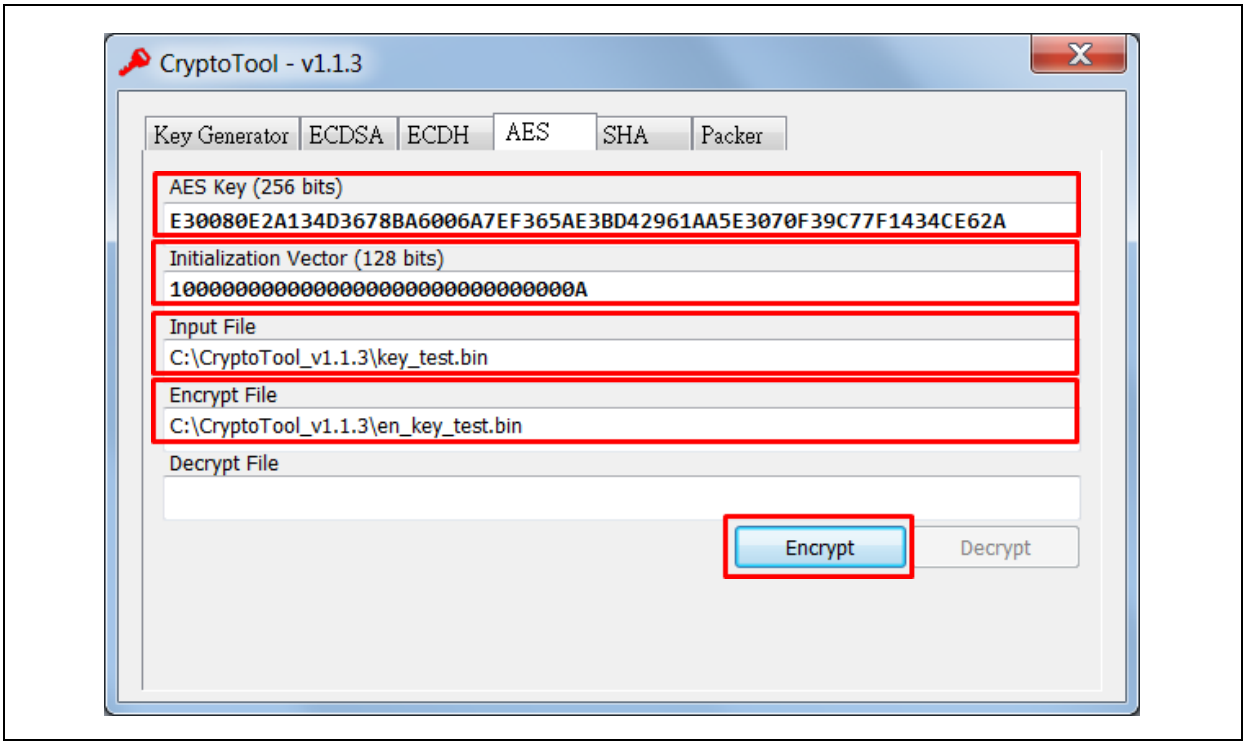


Figure 2-10 Encrypt File by AES

To decrypt an AES encrypted file, user needs to enter relative AES key into "**AES Key (256 bits)**" field, enter IV into "**Initialization Vector (128 bits)**" field, specify encrypted file name in "**Encrypt File**" and output file name in "**Decrypt File**". Then just click "**Decrypt**" button to decrypt the file.

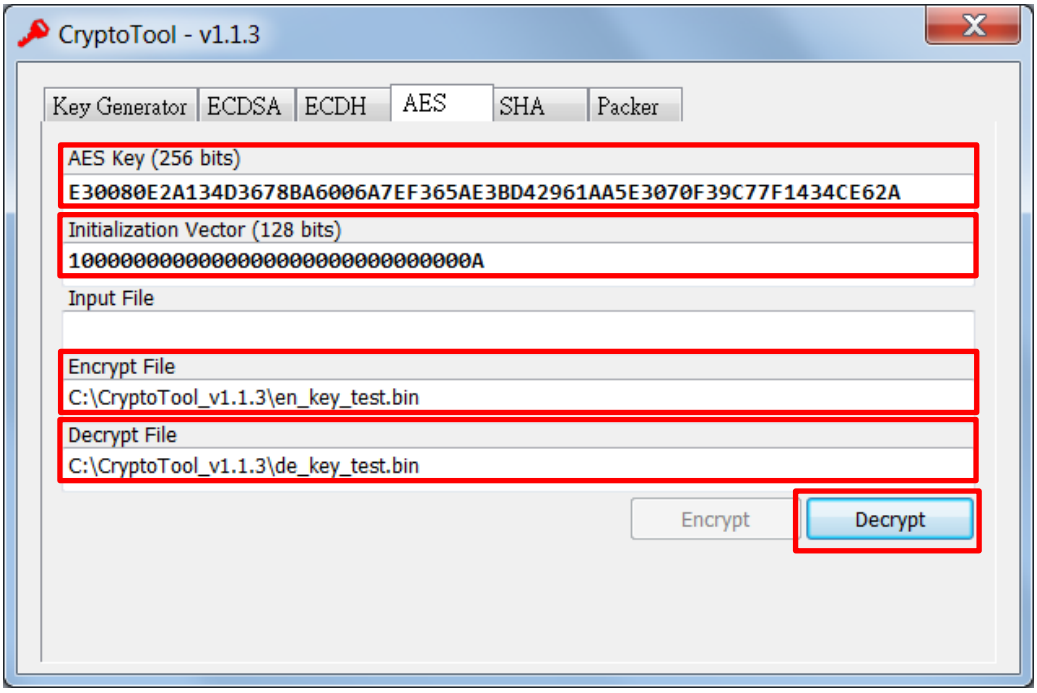


Figure 2-11 Decrypt File by AES

2.5 SHA

Choose "**SHA**" in the CryptoTool. In this tab, tool can calculate a string of hash value according to the input file. Please specify the directory that store the file in "**Input File**", such as a bin file, and click "**Generate Hash**" button, then the hash value will be calculated by SHA calculation and show in "**Hash**". The hash value could be used to verify the file to check whether it is corrupted or modified.

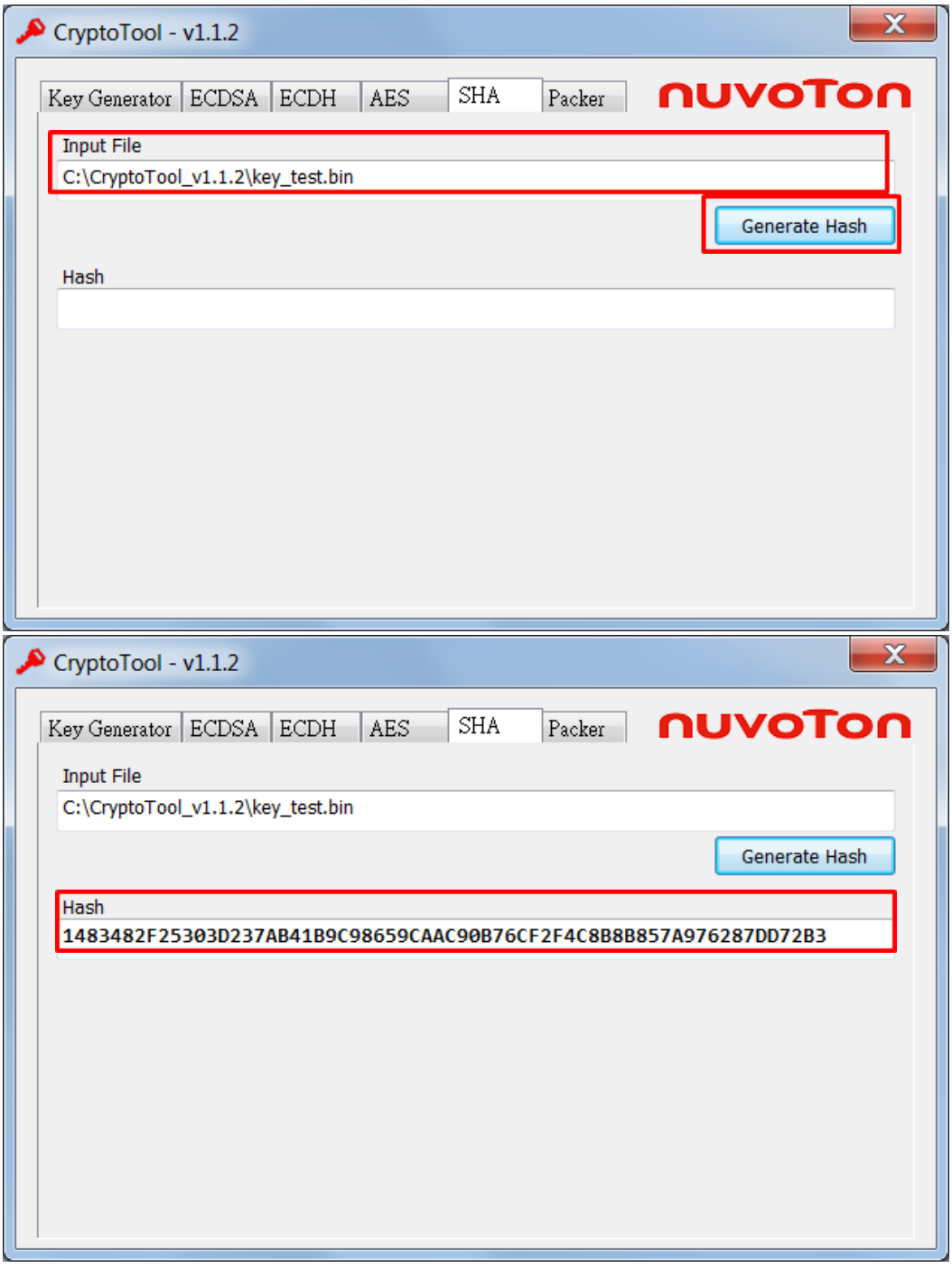


Figure 2-12 Generate Hash

For example, *B* receives a bin file from *A*, but *B* is not sure if the bin file is modified when transfer. At this time, *B* gets bin file hash value from *A* through another secure way and can compare the hash value generated by the received bin file with the hash value provided by *A*. Because different file has different hash value and same file has same hash value, *B* can verify the bin file whether modified or not by comparing hash value.

2.6 Packer

Choose “**Packer**” in the CryptoTool. In this tab, tool can help customer generate a firmware package file used for OTA flow which proposed in M2351. Firmware package include firmware information and encrypted firmware. Firmware information contains public keys, metadata such as the start address or size of encrypted firmware, hash value which is encrypted firmware calculated by AES calculation and signature.

When OTA client receives the firmware package, it needs to use ECDH to calculate a shared key by its own private key and the public keys learned from firmware information, and then it can get the original firmware by using the shared key to decrypt the encrypted firmware by AES. The following will demonstrate how to pack firmware with *A* playing as a customer and *C* playing as OTA client in the OTA application.

If *A* wants to generate a firmware package for *C*, *A* needs to specify the shared key that is created by its own private key and *C*’s public keys by ECDH calculation in “**ECDH Share Key (256 bits)**”, specify the initialization vector that is required by AES CFB mode and known by both sides of *A* and *C* in “**Initialization Vector (128 bits)**”, specify the metadata in “**Firmware Information**”, specify the directory stored the original firmware in “**Firmware 1**” and even “**Firmware 2**” if there are two original firmwares and click “**Pack Firmware**” button. And then, the ouput packed file will be generated and stored in the directory that you have specified in “**Output Pack File**”. It should be noted that, only up to two firmwares are allowed in this “Packer function”.

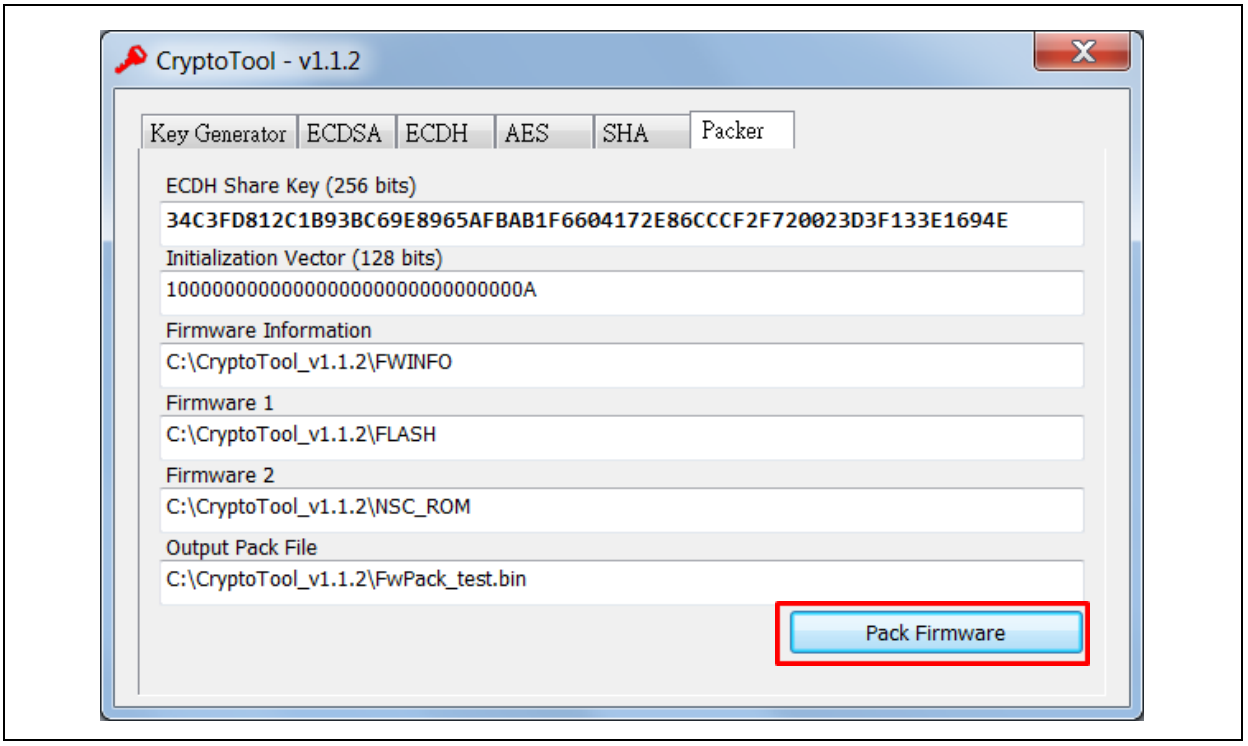


Figure 2-13 Pack Firmware

3 REVISION HISTORY

Date	Revision	Description
2018.08.15	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.