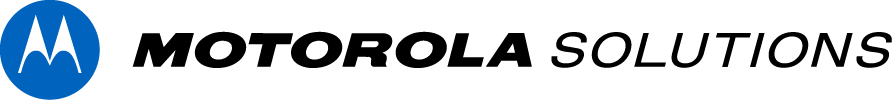
ATTACHMENT 2: Taiwan Indigenous Algorithm Development Training

System Test Platform to be a System

As NEW DESIGN CHANGE REQUEST,

TO DISCUSS!!

This Attachment is for new design change discussion, out of Scope



|  |  |
| --- | --- |
| Radio Solutions  Secure Products Group |  |

Taiwan Indigenous Algorithm

Development Training

6-Jan-25

Version: 0.9

NEW DESIGN CHANGE REQUEST,

TO DISCUSS!!

This Attachment is for new design change discussion, out of Scope

**This document is subject to formal control procedure and will be stored electronically. The electronic version will be considered as the ‘Controlled’ copy. All printed copies will be considered as ‘Uncontrolled’.**

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Authors** | **Description** |
| 27-Dec-10 | 0.1 | C. Perrin /  B. Pruss | Initial Creation |
| 05-Jan-11 | 0.2 | B. Pruss | Adding details on algorithm modification and test |
| 06-Jan-11 | 0.3 | C. Perrin | Added verification test |
| 07-Jan-11 | 0.4 | C. Perrin | Updated to include pictures for programming and added VPM to document |
| 14-Jan-11 | 0.5 | C. Perrin | Updated with descriptions for training sections, added key erase mechanism, updated development with illustrations |
| 14-Jan-11 | 0.6 | C. Perrin | Added test system verification snapshots |
| 25-Jan-11 | 0.7 | C. Perrin | Section 1, 3, 4, 5 and 7 updated to address customer email comments. |
| 11-Feb-11 | 0.8 | C. Perrin | Added images from uVision and updated to match inputs |
| 06-Jan-25 | 0.9 | Steven Chiang | Updated with latest equipment.  Updated with KVL5000 screen.  Renamed as “Taiwan Indigenous Algorithm Development Training”. |

References:

1. Taiwan Indigenous Algorithm Encryption Modification Plan
2. Taiwan System Test Plan

Table of Contents

[Revision History 25](#_Toc186213483)

[Table of Contents 26](#_Toc186213484)

[1 Introduction 28](#_Toc186213485)

[1.1 Abbreviations 29](#_Toc186213486)

[1.2 Definition 29](#_Toc186213487)

[2 Motorola Key Management and Encryption Modes 30](#_Toc186213488)

[2.1 Key Management and supporting encryption modes 30](#_Toc186213489)

[2.1.1 Over-The-Air-Rekeying (OTAR) 30](#_Toc186213490)

[2.1.2 Key management within the system 30](#_Toc186213491)

[2.1.3 Key Management Encryption Modes 31](#_Toc186213492)

[2.2 Traffic Encryption and supporting Encryption Modes 31](#_Toc186213493)

[2.3 Viewing Keys within the KMF 31](#_Toc186213494)

[2.4 KMF to KVL to end subscriber/infrastructure interaction 31](#_Toc186213495)

[2.4.1 Setting up the KMF 31](#_Toc186213496)

[2.4.2 Synchronizing the KMF to a specific KVL 31](#_Toc186213497)

[2.4.3 Synchronization of the KMF and subscribers 31](#_Toc186213498)

[2.4.4 KMF Operator Control 32](#_Toc186213499)

[2.5 Key Erase on the System 32](#_Toc186213500)

[2.5.1 KMF Key Erase 32](#_Toc186213501)

[2.5.2 MACE Key Erase 33](#_Toc186213502)

[2.5.3 VPM Key Erase 33](#_Toc186213503)

[2.5.4 Key Erase mechanism 33](#_Toc186213504)

[3 Algorithm Development Kit Usage 34](#_Toc186213505)

[3.1 Start the Algorithm Development Kit (ADK). 34](#_Toc186213506)

[3.2 Compiler and linker Settings 34](#_Toc186213507)

[3.3 Code Modifications 35](#_Toc186213508)

[3.4 Configurable Parameters Modification 35](#_Toc186213509)

[3.4.1 S-Box 35](#_Toc186213510)

[3.4.2 Mix Columns 35](#_Toc186213511)

[3.5 Development Process 36](#_Toc186213512)

[3.6 Layout for the Algorithm development Kit (ADK) 39](#_Toc186213513)

[3.7 Testing the modified algorithm 40](#_Toc186213514)

[4 Motorola Home Algorithm Support 42](#_Toc186213515)

[5 Programming the Configurable Parameters into the Target Devices 43](#_Toc186213516)

[5.1 Load Configurable Parameters into KVL 43](#_Toc186213517)

[5.2 Load Parameters into Target 44](#_Toc186213518)

[6 System Test Execution 48](#_Toc186213519)

[7 System Test Encryption Verification 49](#_Toc186213520)

[7.1 Verify KMF / CRYPTR-2 50](#_Toc186213521)

[7.2 Verify the Subscriber or VPM 51](#_Toc186213522)

[7.3 Entering and comparing information for the verification 52](#_Toc186213523)

**List of Figures**

[Figure 1 Key Management within System 30](#_Toc187164840)

[Figure 2 Protocol for Store and Forward 32](#_Toc187164841)

[Figure 3 Crypto boundary with the Tamper Security 34](#_Toc187164842)

[Figure 4 ADK set up (Sample) 37](#_Toc187164843)

[Figure 5 Configurable Parameters Entry (Sample) 37](#_Toc187164844)

[Figure 6 Test on Simulator (Sample) 38](#_Toc187164845)

[Figure 7 Test Simulator test parameters (Sample) 39](#_Toc187164846)

[Figure 8 Development Environment (Sample) 39](#_Toc187164847)

[Figure 9 Connection for Parameter Download 43](#_Toc187164848)

[Figure 10 Simulator to KVL connection (Sample) 43](#_Toc187164849)

[Figure 11 KVL Keys menu for key creation 44](#_Toc187164850)

[Figure 12 KVL menu option 44](#_Toc187164851)

[Figure 13 KVL with Key creation 45](#_Toc187164852)

[Figure 14 New Key created 45](#_Toc187164853)

[Figure 15 KVL with new key completed 46](#_Toc187164854)

[Figure 16 Base menu for KVL 46](#_Toc187164855)

[Figure 17 Target menu for keys 47](#_Toc187164856)

[Figure 18: KVL with load menu 47](#_Toc187164857)

[Figure 19: key display example 48](#_Toc187164858)

[Figure 20: Entry into verification simulator to perform verification (Sample) 55](#_Toc187164859)

# Introduction

This document is used to provide the details to support the development of an indigenous algorithm using the toolkit provided by Motorola.

The sections have been selected to ensure that the end-user understands the modes of encryption used in the Motorola public safety communications system, and can then apply the tools necessary to create an indigenous algorithm from a base template.

The modified AES algorithm will have performance that meets the P25 requirements for voice encryption.

Note: the settings for Standard AES and the modified AES with this specific algorithm implementation will have the same performance. Although the original algorithm implementation took advantage of known industry optimizations, the performance from this unit will meet the needs for P25.

The sections start with the encryption modes and flow through the development process until system test and verification is complete:

Section 1: Introduction

This Introduction is structured to provide information that builds upon the previous section such that the customer builds their knowledge of Motorola encryption processes, understands the development environment, can differentiate between what Motorola will provide and what the customer will create and finally is able to provision Motorola equipment with the custom algorithm and is able to verify the algorithm between the ADK and the system.

Section 2: Motorola Key Management and Encryption Modes

The Motorola key management and encryption mode training will focus on how keys are managed within a Motorola system and the modes of encryption used to encrypt the keys. The customer will then understand the basics of what types of keys exist within the system, how they are used (their role) and how they are distributed. The customer will also know the different modes of encryption applied to each key.

Section 3: Algorithm Development Kit (ADK) and configurable parameters and testing on the simulator.

The Algorithm Development Kit (ADK) training will focus on how the software project is structured to define the base algorithm support and how the configurable parameters fit in the algorithm perspective. The training will then outline how the customer changes the configurable parameters, how to build the software for the simulation and, finally, how to verify the operation within the simulator. This allows the customer to understand what to change and how to test their configurable algorithm parameters before programming them into the target devices.

Section 4: Motorola Home Algorithm Support

The training on the Motorola Home Algorithm Support will provide the customer with an understanding of the separation of the base algorithm for AES and how the software for the configurable parameters fits with the base algorithm. The customer will then understand the split of the functionality between the base algorithm that is provided and the parameters that they provide for the final algorithm.

Section 5: Programming the Configurable Parameters into the target devices

The section on programming the configurable parameters into the target devices outlines the method and procedure to getting the final parameters from the ADK into the target devices that contain the Motorola base algorithm.

Section 6: System Test Execution

The section on system test execution outlines the process and procedure that the customer will follow to ensure that their final algorithm functions correctly within a Motorola system. This ensures that the standard functionality provided by Motorola products functions without issue with the customer algorithm.

Section 7: System Test encryption verification

The system test encryption verification outlines the method used to verify the final encryption algorithm in the products within the system. The customer will learn how to run the system test, collect the necessary data, and validate the algorithm using the simulator and the collected data from the products.

## Abbreviations

AES Standard AES Algorithm

ADK Algorithm Development Kit

BTS RF Base Station

CAP P25 Compliance Assessment Program

CFX Taiwan Home-Country Algorithm, currently in use by Navy

CSIST National Chung-Shan Institute of Science and Technology

KVL Key Variable Loader

KMF Key Management Facility

KMM Key Management Messages

MACE Subscriber Motorola Advanced Crypto Engine

MSTL Motorola Solutions Taiwan Limited

NCSIST National Chung-Shan Institute of Science and Technology 國家中山科學研究院

NIST National Institute of Standards and Technology  
OTAR Over-The-Air-Rekeying

SOW NCSIST SC12060P-CS移動臺等10項 工作說明書

SU Subscriber Radios

V&V Vivian & Vincent International Trading Co., Ltd.

## Definition

Encryption Modification as 密式置換 in SOW.

Encryption Verification System as 密式驗證平臺 in SOW.

KMM Test Platform as the machine for verified KMM messages.

System Test Platform as 系統測試平臺 in SOW.

NEW DESIGN CHANGE REQUEST,

TO DISCUSS!!

This Attachment is for new design change discussion, out of Scope

# Motorola Key Management and Encryption Modes

This section outlines the encryption modes associated with the key management and voice encryption mechanisms. These modes are applicable for both standard AES and for the indigenous algorithm CFX.

## Key Management and supporting encryption modes

### Over-The-Air-Rekeying (OTAR)

OTAR is a suite of operations that enables key distribution and management to be conducted securely over-the-air. The Keyset Changeover command activates the inactive keyset. The CKR Update command sends new encryption keys to a specified CKR group. The Full Update command sends all key information to a selected unit, regardless of currency status. The Optimized Update command updates only the parameters that are marked as not current. This option is only available for Radios, Radio Groups, Secure Phones, and Secure Phone Groups. The Zeroize command removes all keys from a selected unit. KMF eliminates the burden of manually rekeying radios on a regular basis and solves the logistical problem of maintaining secure wireless communications.

### Key management within the system

To manage keys within the communications network, Motorola’s Astro system uses a Key Management Facility (KMF) and a Key Variable Loader (KVL) to provision and update encryption end points (subscribers and infrastructure).

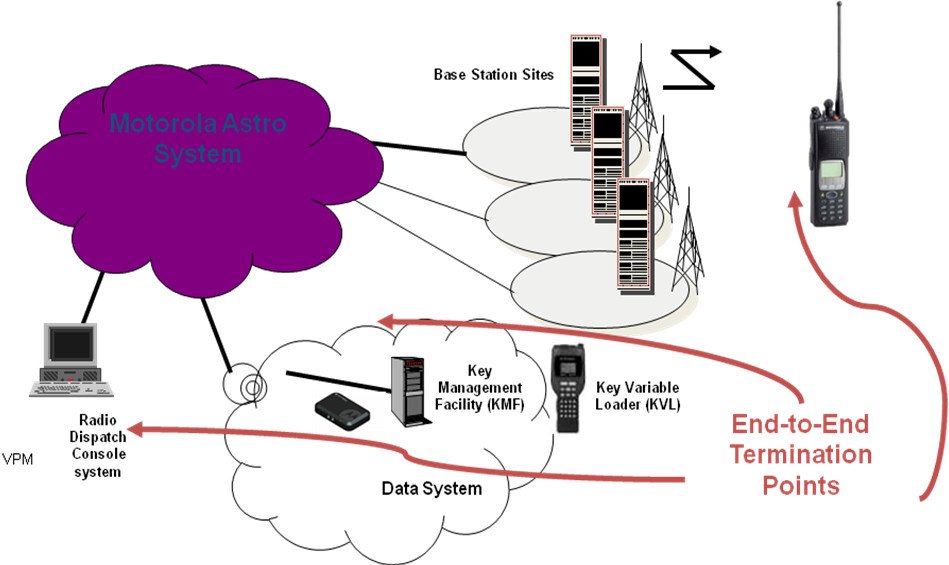


Figure Key Management within System

The KVL loads an initial Key Encryption Key (KEK) into the target devices (subscribers and infrastructure) that matches the KEK stored in the KMF. Then the KMF sends Key Management Messages (KMMs) to the target devices over the air or the network (OTAR). The KMMs can carry Traffic Encryption Keys (TEKs) used to encrypt voice. The TEKs are protected by being encrypted using the KEK.

### Key Management Encryption Modes

The traffic keys are encrypted using the Electronic Code Book (ECB) mode. Multiple TEKs can be combined in one message to form a single KMM. This KMM is then encrypted using a TEK in OFB mode. The ECB mode is known as the inner layer mode and the OFB mode is known as the outer layer mode. A Message Authentication Code may be added that uses Cipher Block Chaining (CBC-MAC).

## Traffic Encryption and supporting Encryption Modes

The traffic encryption mode handles the encryption of the voice information and uses the TEKs loaded from the key management system. The traffic algorithm operates in Output Feedback Mode and is periodically synchronized with an Initialization vector.

## Viewing Keys within the KMF

The KMF allows the user to view the traffic keys created within the KMF in the context of the key ID and its associated algorithm. The user cannot see the key data once entered into the KMF.

This view also allows the user to view what set of radios are using the keys in their current form.

Once a key has been created, a new key with the same ID cannot be created. The current key must be deleted if the user wishes to enter another key with the same key ID. Once the new version of the key is entered the old key will no longer be present or available and cannot be viewed.

To see the keys, the user must log in. The KMF users the windows log in mode to allow people to access the KMF Client.

## KMF to KVL to end subscriber/infrastructure interaction

### Setting up the KMF

To operate the KMF, the KMF will store all the keys locally encrypted. It completes this task by using a master key loaded from a KVL. The KMF is placed in Master Key Load Mode and then the KVL is connected and the Master key is loaded into the KMF. Now the KMF is ready for operation.

### Synchronizing the KMF to a specific KVL

The KMF and KVL are synchronized by using a Key Encryption Key that is matched between the two units. The key is typed in manually in the KMF and in the KVL. This is used for encryption of messages sent between the KMF and the KVL during a Red Store and Forward Keyload to an end point. This prevents any KVL from being attached to the KMF and downloading messages as the KEKs must match between the KMF and KVL.

### Synchronization of the KMF and subscribers

Once the Subscribers and the KMF share the same KEK and have the same algorithm configuration parameters, subsequent keyloading can be managed using black keyloading which encrypts the messages with the KEK and TEK that’s within the subscriber. This prevents a non-authorized radio from receiving and accepting new keys from the KMF via the KVL as the invalid subscriber would be unable to decrypt the key message.

The process below shows the interactions mentioned above.



Figure Protocol for Store and Forward

The process shows the configuration of the KMF, then highlights the manual entry of the KEKs and finally shows the loading of the KEK and TEK into the subscriber – note the warm start process ensures that the link between the KVL and KMF is encrypted until black store and forward (which uses the subscriber or VPM KEK) is in place.

### KMF Operator Control

The KMF supports providing control over what a user can do by having an admin create users with different profiles. One profile can be created to allow keys to be generated in the KMF but not allow update operations and another profile can allow only updates to radios. The control is based on the login of the user.

## Key Erase on the System

### KMF Key Erase

The KMF has a tamper and emergency erase mechanism such that the critical security parameters (keys) are essentially erased when one of the following two events occur:

1. Illegal access to the unit to try to access the keys – causes tamper and key loss.
2. Operator presses emergency key erase button on the KMF CRYPTR-2 – causes key loss.

Recovery consists of having to start again with programming the algorithm parameters and then keys.

### MACE Key Erase

The MACE has a similar mechanism to the KMF in that it can handle deliberate key erase through the user button press on the radio or through tamper activation because an attacker is attempting to access the keys.

This will also result in the algorithm parameters being required to be reloaded and then keys being provisioned as before.

### VPM Key Erase

The VPM also supports the tamper key erase functionality where the encryption processors are maintained under a shield and opening the shield will cause the keys to be erased. The VPM also has an emergency erase button that will erase keys if depressed.

### Key Erase mechanism

The Motorola encryption products use the same design for the key erase mechanism. They all have the following characteristics:

* A power source that feeds the encryption module even when the unit is turned off
* A tamper circuit that is connected to the security processor
* Critical security parameters that are protected through a Key Protection Key (KPK) internal to the security processor
* Connection from the tamper circuit to the tamper on the security processor
* Connection from the I/O and/or the power supply and a physical connection to the shield to the tamper circuit

The key erase mechanism works by first generating a protection key that protects the critical security parameters. Even when power is off the KPK is maintained through a battery backed register. This register will be erased if the tamper circuit is tripped. The tamper circuit can determine if some attempts to remove the module (I/O), cuts the power (power supply) or removes a shield (physical connection). If any of these events occur then the tamper circuit will trip the tamper on the security processor and subsequently the battery backed register is wiped. Therefore, all security parameters are lost and the module is rendered useless until reloaded with the appropriate keys.

The configurable algorithm parameters are stored as one of the critical security parameters and therefore are also lost with a tamper activation.

The diagram below shows an outline of the design:



Figure Crypto boundary with the Tamper Security

# Algorithm Development Kit Usage

When developing the indigenous algorithm, the end-user will use an Algorithm Development Kit (ADK) with a Motorola Project. This section outlines the use of the Algorithm Development Kit in terms of the how it is structured.

The key philosophy behind the ADK is that the end-user is able to focus on defining the values for the configurable parameters and running the simulations without overly focusing on the software development. To that end, Motorola has created an algorithm ADK that contains the source code modifications required for the indigenous algorithm with the end-user only needing to set the configurable parameter values in a single header file.

## Start the Algorithm Development Kit (ADK).

On the Laptop use windows to go to ADK program. The Motorola Project will already be installed on the Laptop. Open the Motorola Project in the File Open section.

## Compiler and linker Settings

The compiler directives are pre-configured in the Motorola project imported into the ADK. The end-user does not need to modify the compiler directives for building the software. The cross platform compiler runs on windows and compiles for the processor.

The order of compilation and linking is also pre-defined by the Motorola project and therefore will require no modification by the end-user.

## Code Modifications

The Motorola project already installs the source code for the algorithm. This includes the software that will allow the configurable parameters to be utilized. The end user need only focus on the changes associated with the configurable parameters. These changes will be contained within a single header file for ease of use. The file will be called Alg\_Param\_Config.h and the end user will modify the ALG\_PARAM\_CONSTANTS definition according to their pre-defined values.

## Configurable Parameters Modification

The end-user will determine the values for the S-Box and the MixColumns as per the following algorithms:

### S-Box

The S-box modifications will be defined using the following algorithm:

Specify a finite field polynomial to be used in the computation of the multiplicative inverse of a byte. The finite field will be represented by irreducible polynomial (see note 1 on Section 3.4.2).

Specify 1 byte to apply the affine transformation matrix and 1 byte to apply the affine transformation vector.

### Mix Columns

The mix colums will be represented by the following rules and specification:

1. Specify the Galois Field Polynomial (note: must be irreducible)
2. Specify the Forward FirstColumn Polynomial.
3. Specify the Reverse FirstColumn Polynomial.

These values will be managed through the following byte specification:

* + GF Polynomial (see note 1)
  + Forward FirstColumn Polynomial (4 bytes)
  + Reverse FirstColumn Polynomial (4 bytes)

Note 1: The all 9 bits of the Irreducible Polynomial can allow to be specified. However, the code will need to reject all values that do not have the 9th bit set to 1. That is a mathematical necessity - the algorithm can not generate usable output if that is not true.

The values for both configurable parameters are represented by the definition within the header as follows:

define ALG\_PARAM\_CONSTANTS \

{ \

/\* SubBytes GF Polynomial \*/ \

0x01, 0x1B, /\*<== NOTE: Most-significant bit of GF(2^8) Polynomial is \*/\

/\* required to be 1. Other values will be rejected. \*/ \

/\* SubBytes Forward Affine Transformation Matrix (Affine(x)) \*/ \

/\* (Note: Format should be an 8-bit vector defining a circulant matrix\*/\

/\* over GF(2), MSB first) \*/ \

0x1F, \

\

/\* SubBytes Forward Affine Transformation Vector (Const(x)), MSB first \*/\

0x63, \

\

/\* SubBytes Inverse Affine Transformation Matrix (Affine(x)) \*/ \

/\* (Note: Format should be an 8-bit vector defining a circulant matrix\*/\

/\* over GF(2), MSB first) \*/ \

0x4A, \

/\* SubBytes Inverse Affine Transformation Vector (Const(x)), MSB first \*/\

0x05, \

\

/\* MixColumns GF Polynomial \*/ \

0x01, 0x1B, /\* <== NOTE: Most-significant bit of GF(2^8) Polynomial is\*/\

/\* required to be 1. Other values will be rejected. \*/ \

\

/\* FirstColumns Forward Polynomial Coefficients \*/ \

/\* (Note: Format should be a 4x8-bit vector defining a circulant 4x4 \*/ \

/\* matrix over GF(2^8)) \*/ \

0x03, 0x01, 0x01, 0x02, \

\

/\* FirstColumns Reverse Polynomial Coefficients \*/ \

/\* (Note: Format should be a 4x8-bit vector defining a circulant 4x4 \*/ \

/\* matrix over GF(2^8)) \*/ \

0x0B, 0x0D, 0x09, 0x0E, \

\

/\* Reserved \*/ \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 \

}

## Development Process

The end-user develops their encryption algorithm by completing the following steps:

1. Start the development environment with the Motorola project (this is already loaded on the development laptop provided to the end-user).
2. Identify the header file, Alg\_Param\_Config.h, in the source code listing section, shown in Figure 4 ADK set up (Sample).

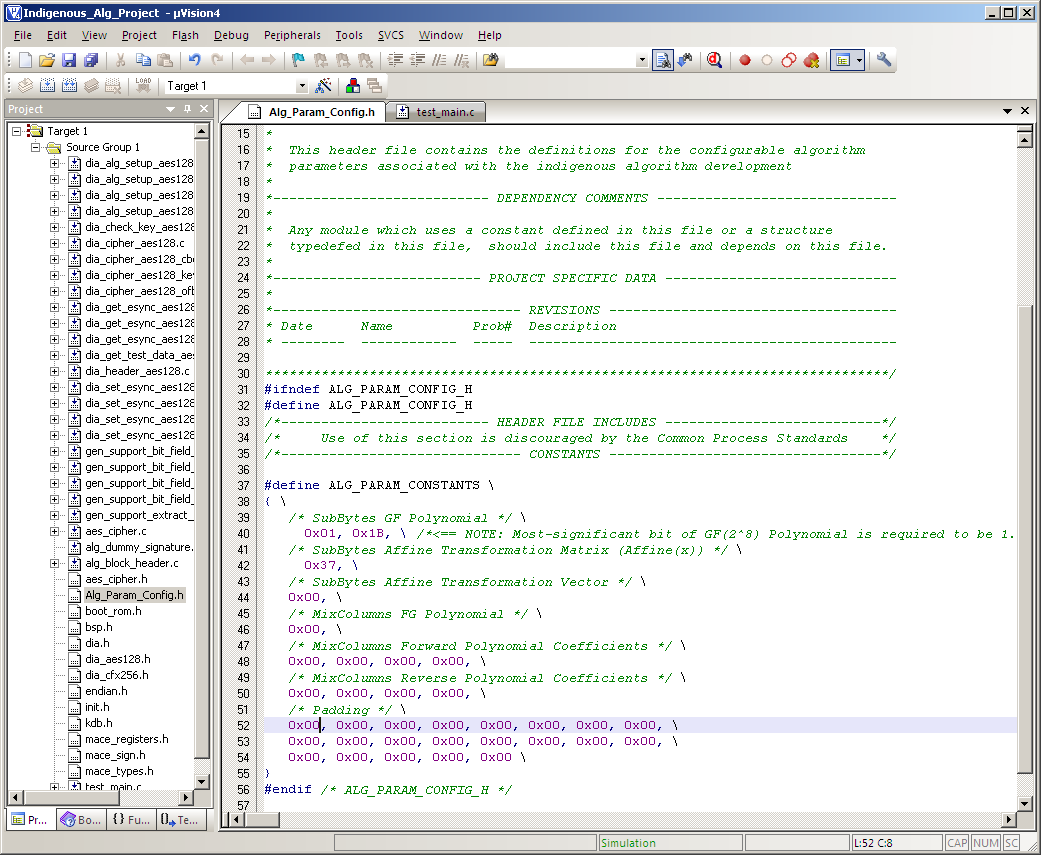


Figure ADK set up (Sample)

1. Select the header file for editing and it will appear in the source code section shown in Figure 4 ADK set up (Sample).
2. Modify the configurable parameter values for the mix columns and S-Box as identified in Section 3.4.

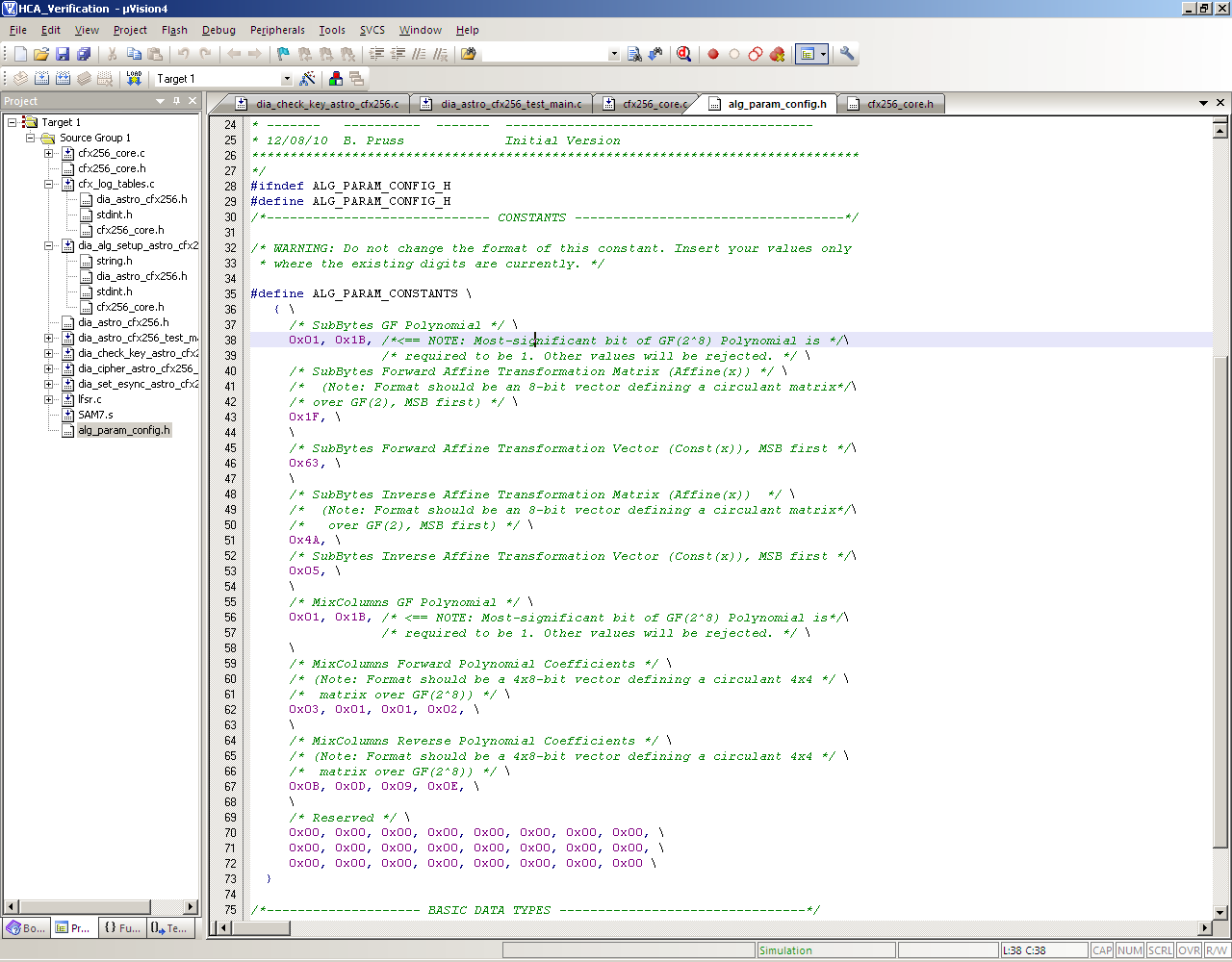


Figure Configurable Parameters Entry (Sample)

1. Select the test\_main.c file in the source listing and edit the test input data (plaintext to be encrypted) and traffic key to your chosen values.

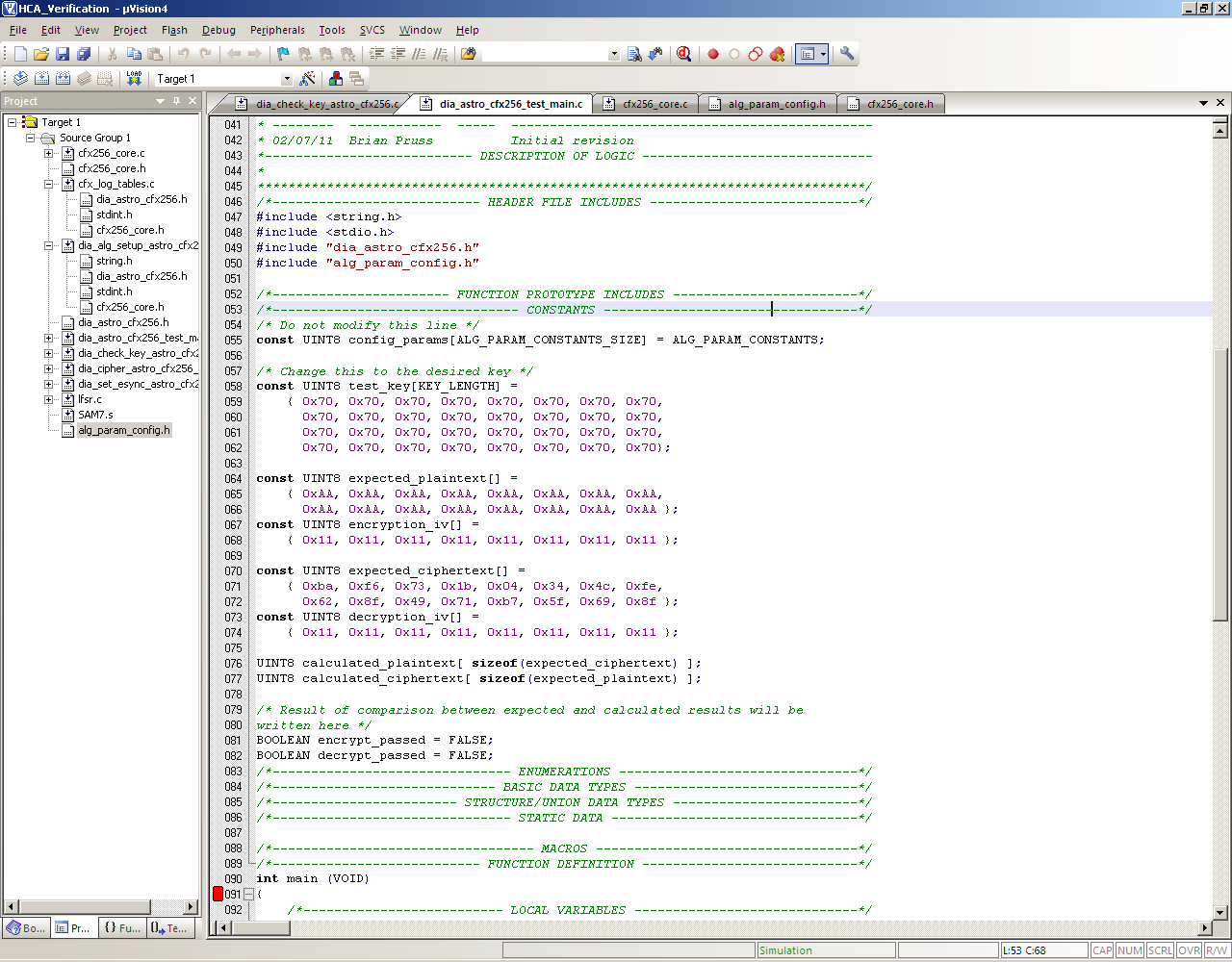


Figure Test on Simulator (Sample)

Click on the compile and link button as shown in Figure 4 ADK set up (Sample). This will now compile and link the software in debug mode. If successful, the simulator will be automatically invoked with the test program.

1. Use the simulator functionality (step line-by-line, or break points – see the ADK help menu for more information on standard ADK options), step through the execution of the test program. The test program will encrypt the end-user defined plaintext using the configurable parameters, the Initialization Vector and the Traffic Key. The end-user can monitor the variables that represent the plaintext and the cipher text through each round, refers Figure 6 Test on Simulator (Sample).

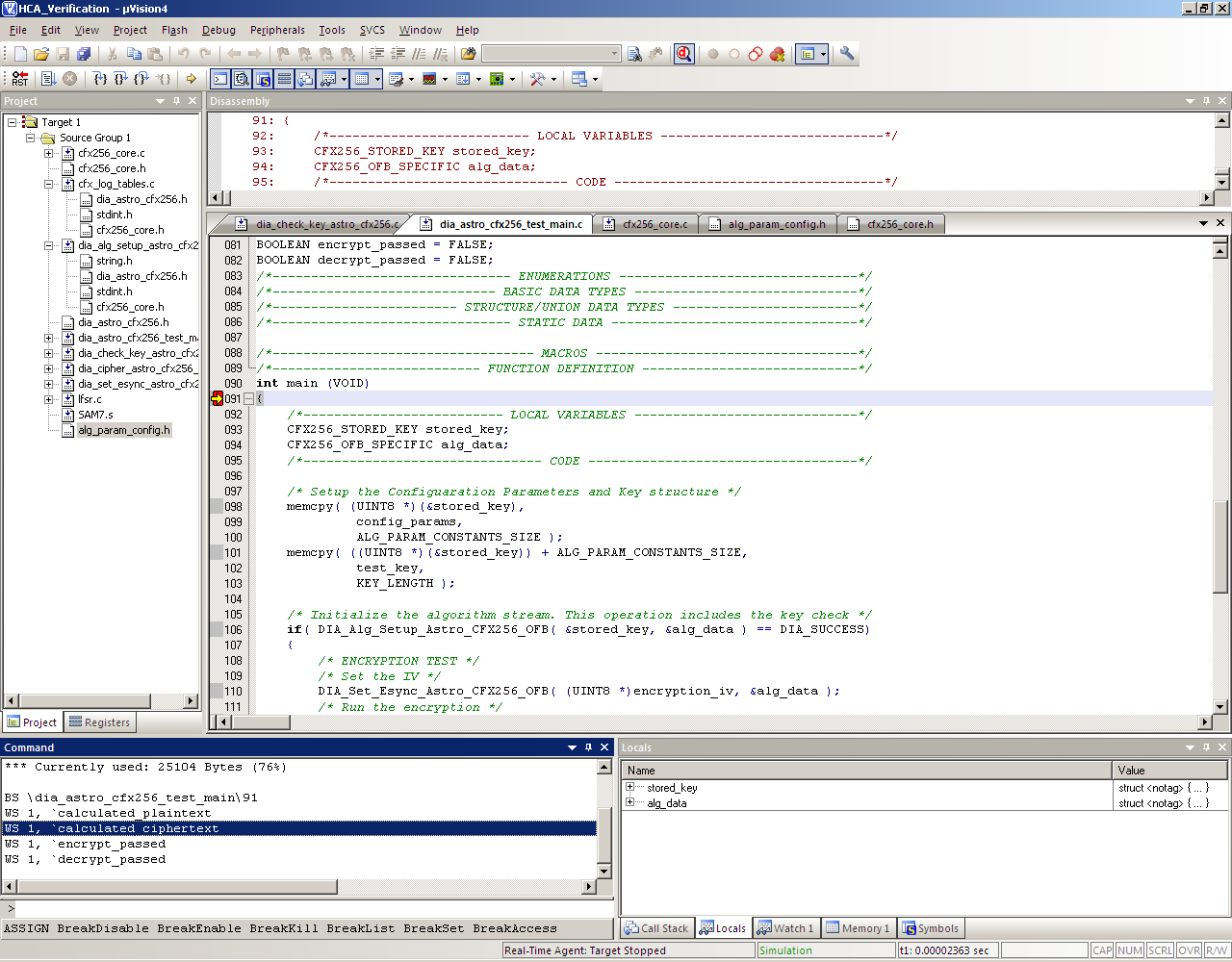


Figure Test Simulator test parameters (Sample)

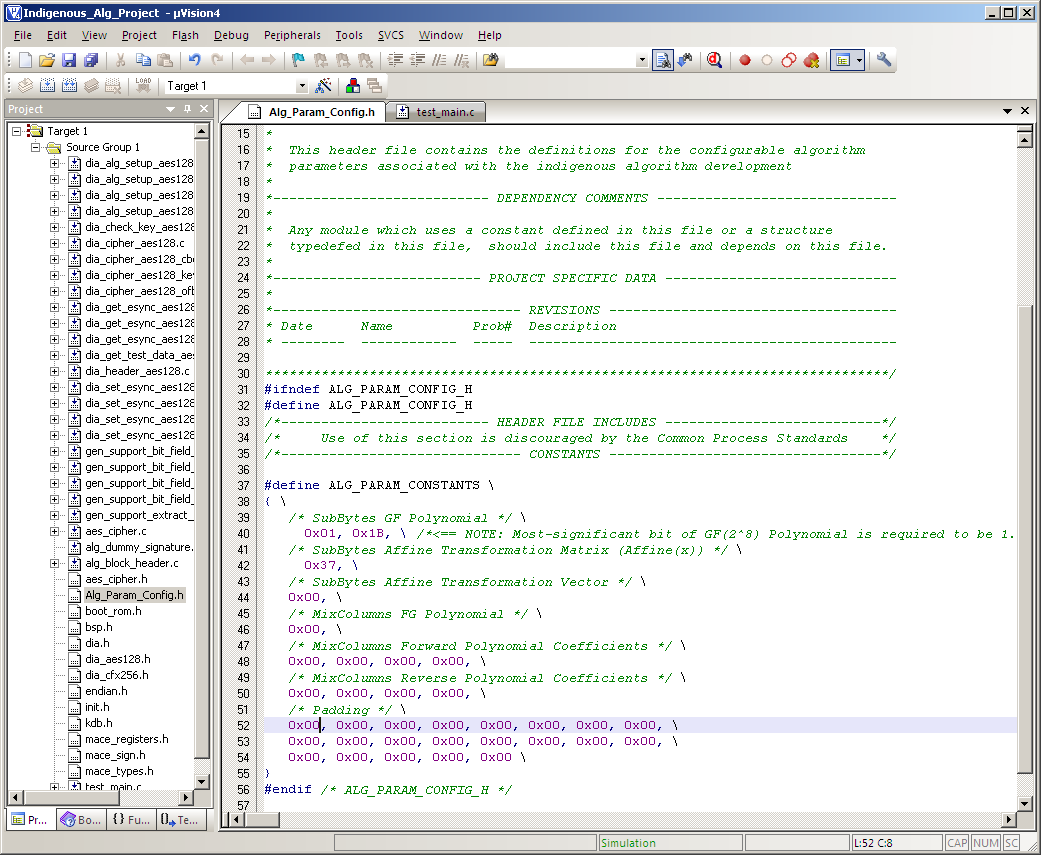
1. Once the end-user is satisfied with the results the parameters need converting for programming into the target device. This will be covered in a later section.

Note: If the user wishes to verify the algorithm according to standard AES (i.e. set the algorithm to standard AES) then they can load the s-box and mix-column values as per the standard AES algorithm.

## Layout for the Algorithm development Kit (ADK)

Compile and debug

The ADK will have a layout as below:



Source code files

Source code editor

Figure Development Environment (Sample)

## Testing the modified algorithm

The following steps are provided to use the ADK for testing the modified algorithm. Examples are provided where appropriate to use the ADK to implement standard AES.

1. Modify the parameters in Alg\_Param\_Config.h to the desired value. An example using the standard Rjindael values is presented below.

#define ALG\_PARAM\_CONSTANTS \

{ \

/\* SubBytes GF Polynomial \*/ \

0x01, 0x1B, /\*<== NOTE: Most-significant bit of GF(2^8) Polynomial is \*/\

/\* required to be 1. Other values will be rejected. \*/ \

/\* SubBytes Forward Affine Transformation Matrix (Affine(x)) \*/ \

/\* (Note: Format should be an 8-bit vector defining a circulant matrix\*/\

/\* over GF(2), MSB first) \*/ \

0x1F, \

\

/\* SubBytes Forward Affine Transformation Vector (Const(x)), MSB first \*/\

0x63, \

\

/\* SubBytes Inverse Affine Transformation Matrix (Affine(x)) \*/ \

/\* (Note: Format should be an 8-bit vector defining a circulant matrix\*/\

/\* over GF(2), MSB first) \*/ \

0x4A, \

/\* SubBytes Inverse Affine Transformation Vector (Const(x)), MSB first \*/\

0x05, \

\

/\* MixColumns GF Polynomial \*/ \

0x01, 0x1B, /\* <== NOTE: Most-significant bit of GF(2^8) Polynomial is\*/\

/\* required to be 1. Other values will be rejected. \*/ \

\

/\* MixColumns Forward Polynomial Coefficients \*/ \

/\* (Note: Format should be a 4x8-bit vector defining a circulant 4x4 \*/ \

/\* matrix over GF(2^8)) \*/ \

0x03, 0x01, 0x01, 0x02, \

\

/\* MixColumns Reverse Polynomial Coefficients \*/ \

/\* (Note: Format should be a 4x8-bit vector defining a circulant 4x4 \*/ \

/\* matrix over GF(2^8)) \*/ \

0x0B, 0x0D, 0x09, 0x0E, \

\

/\* Reserved \*/ \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \

0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00 \

}

1. Modify the Key, Plaintext, and Ciphertext values in the test\_main.c file to the desired values. For standard AES, the following values may be used:

const UINT8 test\_key[32] = {

0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70,

0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70,

0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70,

0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70, 0x70};

const UINT8 expected\_plaintext[16] = {

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA,

0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA, 0xAA };

const UINT8 expected\_ciphertext[16] = {

0x93, 0xac, 0x47, 0xe1, 0x20, 0xb5, 0xce, 0x9e,

0x3a, 0x3f, 0x96, 0xa1, 0xa4, 0x9b, 0xbe, 0x5d};

1. Using the instructions from section 3.5, step to the beginning of the main() function. Main() will have the following program flow:
   1. The Check\_Key function will be called on the test keys and configuration parameters. If it passes, execution will continue, otherwise the function will return and execution will cease.
   2. The Alg\_Setup function will be called. This will initialize the algorithm data structures.
   3. The Encrypt function will be called using the expected\_plaintext array as input. Output will be written into the calculated\_ciphertext array in RAM.
   4. Calculated\_ciphertext will be compared to expected\_ciphertext, and the result will be stored into a variable.
   5. Next, the decrypt operation will be tested. The Decrypt function will be called using the expected\_ciphertext array as input. Output will be written into the calculated\_plaintext array in RAM.
   6. Calculated\_plaintext will be compared to expected\_plaintext, and the result will be stored into a variable.

# Motorola Home Algorithm Support

This section details the separation of the Base algorithm from the configurable parameters.

The Motorola Home Country Algorithm kit allows the user to create a customized encryption algorithm based upon the same framework as AES. This is done by allowing the user to replace certain constants used in the calculation.

The algorithm code may be broken up into the following functional areas:

* API Code
  + Handles calls and data I/O from application code.
* Algorithm Mode Implementations
  + Uses Top-level Encrypt and Decrypt Functions along with additional code to implement various Algorithm Modes needed for Data Processing:
    - Output-Feedback (OFB) Mode
    - Cipher-Block-Chaining (CBC) Mode
* Top-level Encrypt and Decrypt functions
  + Calls Subroutine functions in sequence to implement the core Electronic Codebook (ECB) mode of the algorithm, used by the Algorithm Mode implementation.
  + Subroutine call sequence is unmodified from Rjindael / AES.
* Algorithm Subroutine functions (Based on Rjindael / AES algorithm design
  + KeyExpansion
    - Unmodified from Rjindael / AES
  + SubBytes
    - Modified to use replacement S-Box as derived from the configurable Algorithm Parameters.
    - Uses the SubBytes GF Polynomial, SubBytes Affine Transformation Matrix, and SubBytes Affine Transformation Vector as defined in Alg\_Param\_Config.h.
  + ShiftRows
    - Unmodified from Rjindael / AES
  + MixColumns
    - Modified to use MixColumns GF Polynomial, and MixColumns Forward and Reverse Polynomial Coefficients.
  + AddRoundKey
    - Unmodified from Rjindael / AES

# Programming the Configurable Parameters into the Target Devices

Once the end-user is satisfied with the settings for the configurable parameters then next step is to program them into the target devices. The target devices must have been programmed with the supporting software prior to the programming of the configurable parameters.

## Load Configurable Parameters into KVL

1. Connect the KVL to the **Accessory software, hardware and tools related to Encryption Modification** **密式置換相關附屬軟硬體及工具,** as illustrated in Figure 9: Connection for Parameter Download.
   1. Ensure that the KVL is powered on and connected to the COM serial port on the System Test Platform through an RS-232 Null Modem cable.

System Test Platform

As NEW DESIGN CHANGE REQUEST

TO DISCUSS!!

RS-232 Null-Modem Cable

KVL

COM1

ADK Computer



Figure Connection for Parameter Download

1. From the window pulldown menus, select Tools→ Download\_Configurable\_Parameters.

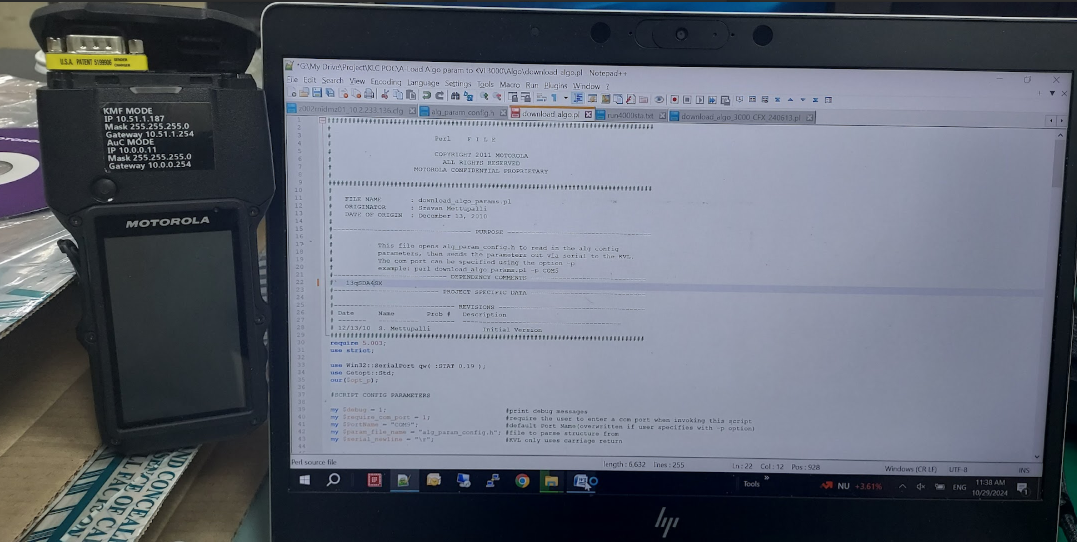


Figure Simulator to KVL connection (Sample)

1. A DOS shell window will appear and will display the status of the operation.
2. When the operation is complete, the window will display “Operation Complete. Press any key to continue.” Press any key to clear the window and returns.

## Load Parameters into Target

After the parameters have been loaded into the KVL, they must be transferred to each target device. This will be handled automatically when a key for the algorithm is loaded into the target device using that KVL. The simplest method is to create a traffic key to load into the subscriber. This will load the configurable parameters along with the key. The detail step for algorithm loading into target device can be referred to document “Taiwan System Test Plan”.

To perform this action complete the following:

1. Tab over to the Keys menu on the KVL

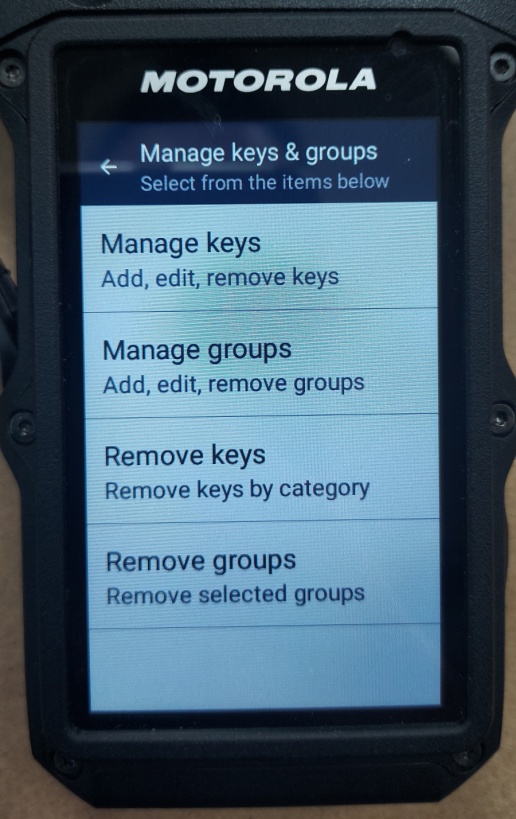


Figure KVL Keys menu for key creation

1. Select the Keys menu and the display will change to the option to edit a current key or create a new key.

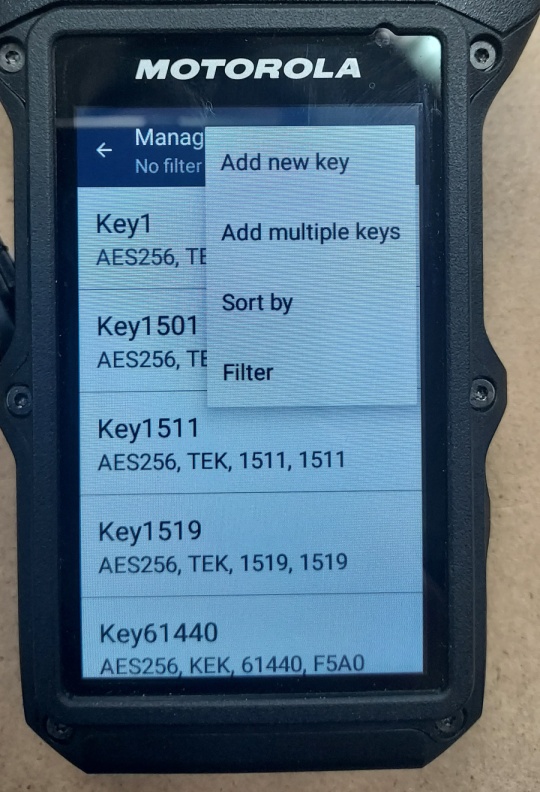


Figure KVL menu option

1. Select the “new” option and the screen will change to creating a new key.

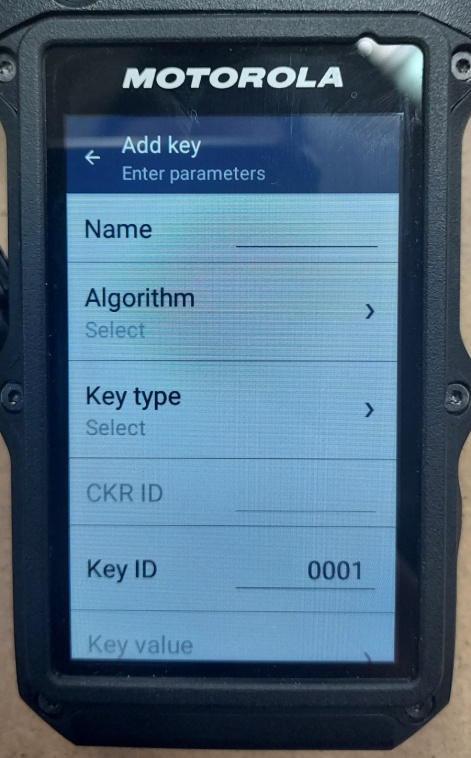


Figure KVL with Key creation

The KVL will now display the menu to enter the details. Enter a common key reference identifier (CKR). This is used to indicate a unique key reference.

1. Select the algorithm (this should be the Taiwan algorithm name). Select Accept.

The KVL will now accept input for the KVL key data

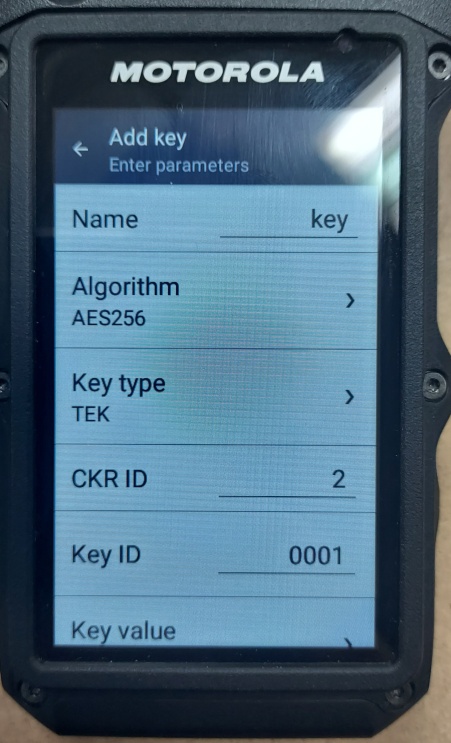


Figure New Key created

1. Once complete the KVL will display the “slot filled” – press “enter on the KVL. This will save the new key and show it on the screen

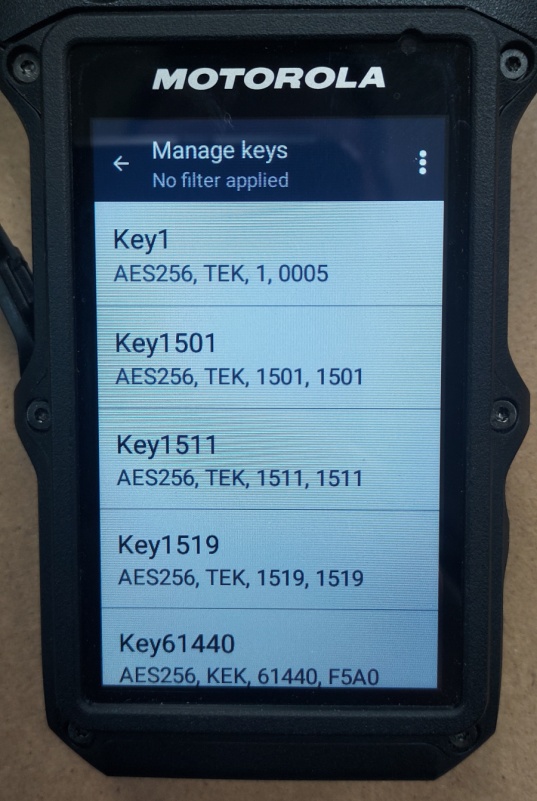


Figure KVL with new key completed

1. Connect the KVL to the mobile or the CryptR or the VPM

Press esc to get back to the main menu

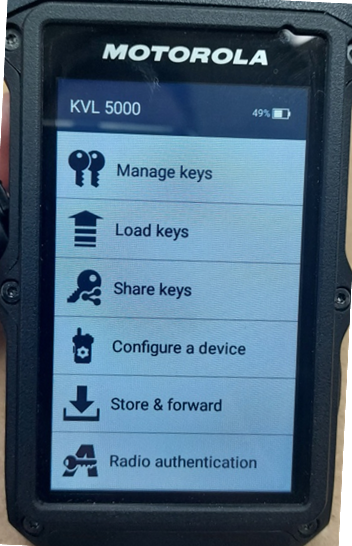


Figure Base menu for KVL

1. Then tab over using the “>” to the target menu and select Target

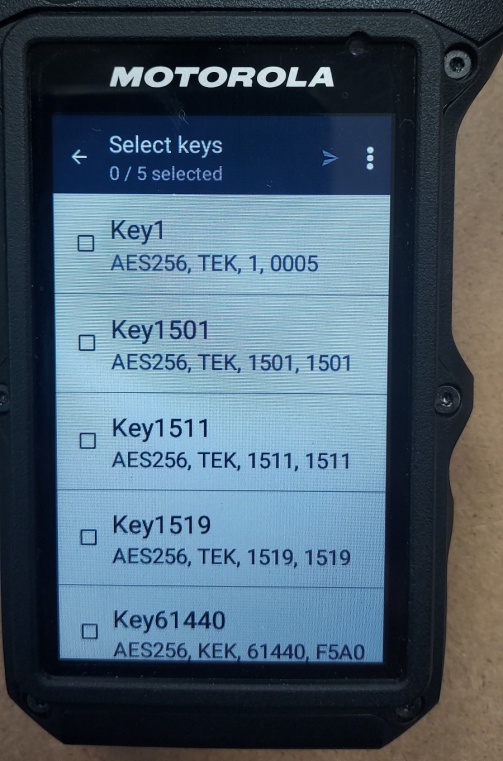


Figure Target menu for keys

1. Then select load and the menu will change to provide an option to load a key or group.

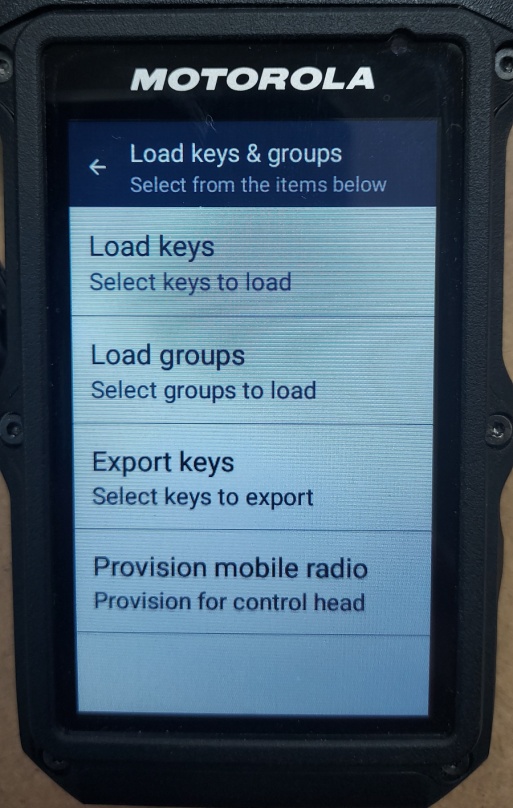


Figure : KVL with load menu

1. Select key and the menu will provide the option to select which key to load. Press “>” until the desired key is showing.

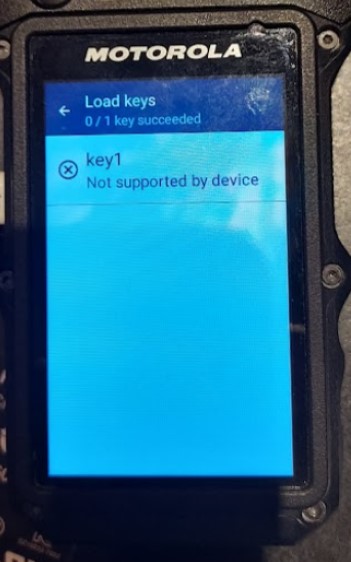


Figure : key display example

And the key will be entered into the target device. This will also load in the configuration parameters into the target device.

# System Test Execution

This section is essentially to verify the standard system level functions in which the corresponding detail procedures are to be referred in document “Taiwan System Test Plan”.

# System Test Encryption Verification

The aim of the encryption verification is to validate that the algorithm operation on the simulator now operates the same when it is configured in the system. To that end the procedure is as follows:

1. Verify the KMF CRYPTR-2 as detailed below.
2. Once complete the KMF CRYPTR-2 now acts as a trust point, and is used to send an encrypted KMM to the subscribers and the VPM and the encrypted responses are verified by the verification platform.

Note:

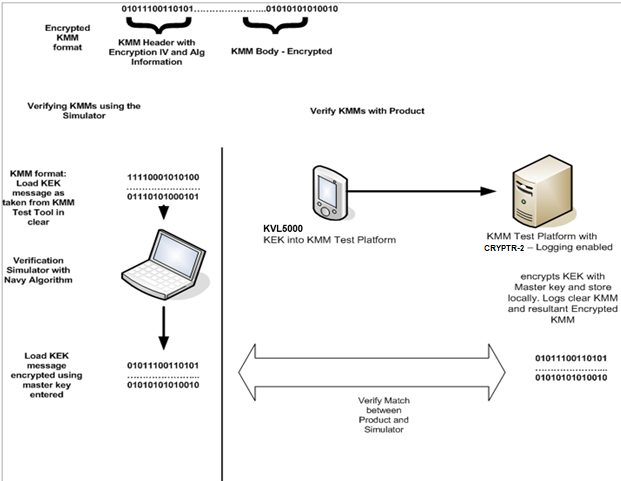
* 1. KMM refers to Key Management Message, it is different term from “KMM Test Platform”. KMM Test Platform is within Encryption Verification System (密式驗證平臺).
  2. KMF refers to Key Management Facility.

The principle of the encryption verification is based upon the fact that the KMM Test Platform is able to log the plaintext and ciphertext of the key management messages that it processes. A plaintext KMM will contain the data associated with the keys that are being transmitted but the encrypted KMM also contains information about the encryption process such as the initialization vector, the algorithm ID, and the key ID of the key used to encrypt the messages. This information can be used to reproduce either the encrypted version of a plaintext KMM or the decrypted version of an encrypted KMM.

Once the KMF CRYPTR-2 is verified, the subscribers and VPM can be verified by taking the encrypted KMM responses and, using the IV, Key ID (and hence Key data because the user entered the KEK material) and the Alg ID (Taiwan), the user can reproduce the plaintext response on the verification simulator and compare it to the logs on the KMM Test Platform. If they match the subscribers and VPM must be using the algorithm correctly.

The final step is to check the operation of the radios with the newly delivered traffic keys. This can be achieved by running encrypted calls in direct mode between the two subscribers. If the radios unmute then they correctly received the keys, stored them and, with the algorithm, are now using them correctly.

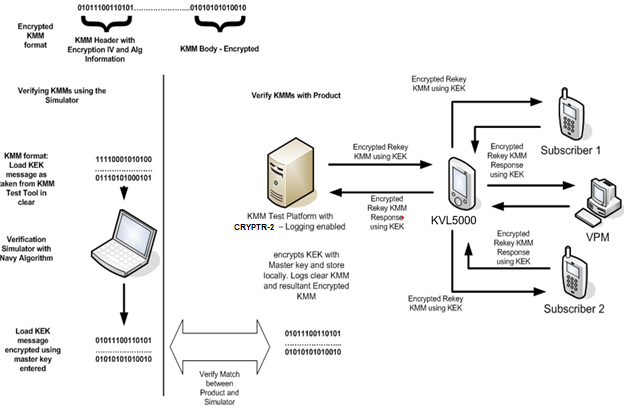
## Verify KMF / CRYPTR-2



The KMM format essentially consists of a header and the KMM body (data). The encrypted version contains a plaintext section that outlines the information required for a receiving device to decrypt that KMM (Key ID, IV, Algorithm ID).

For the KMF CRYPTR-2, the user inputs the master key into the KMM test platform. This key is used to encrypt all keys that are stored locally in the KMM Test platform. Once the master key is entered the user can load a KEK that will be used for KMMs designated for the subscribers and VPM at a later stage. The logging mechanism will provide the plaintext and ciphertext versions of the storage of the KEK. Given that the user entered the master key data, they can duplicate the results and verify the match between the KMM Test platform (which contains the KMF CRYPTR-2) and the Verification Simulator.

## Verify the Subscriber or VPM



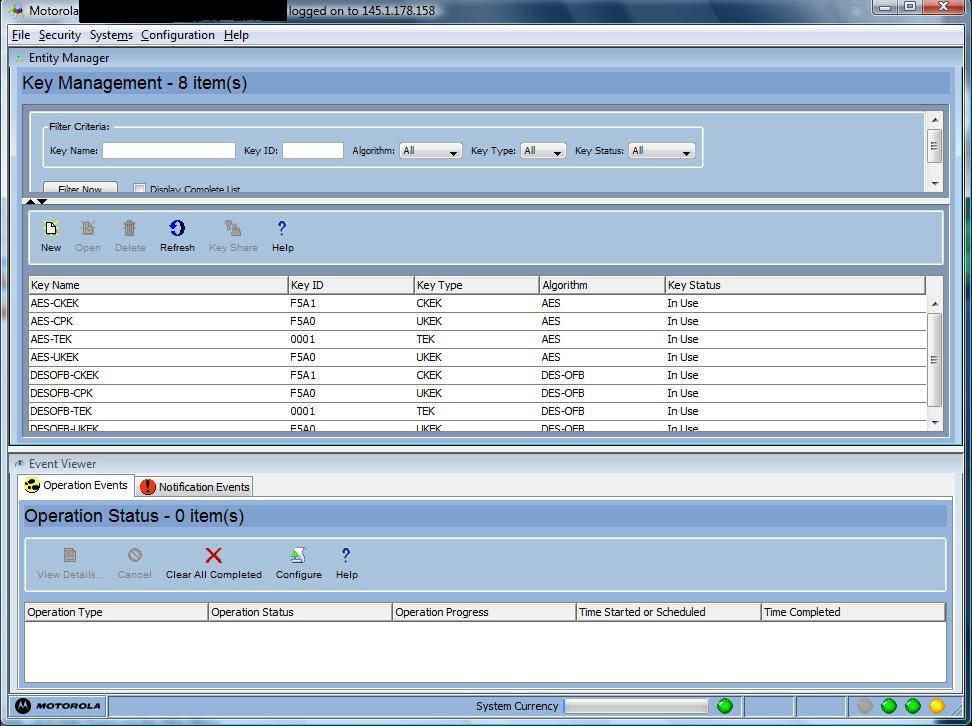
In the instance of the subscriber or VPM, the user will load the same KEK (that exists in the KMM Test Platform) into the subscribers and VPM. The user then configures and sends a Rekey KMM to the subscribers and VPM using the KVL. These messages are encrypted using the KEK.

The subscriber and VPM decrypt the keys and send an encrypted response via the KVL. The KMM test platform will store and log the plaintext and ciphertext versions of the rekey responses from the subscribers and VPM.

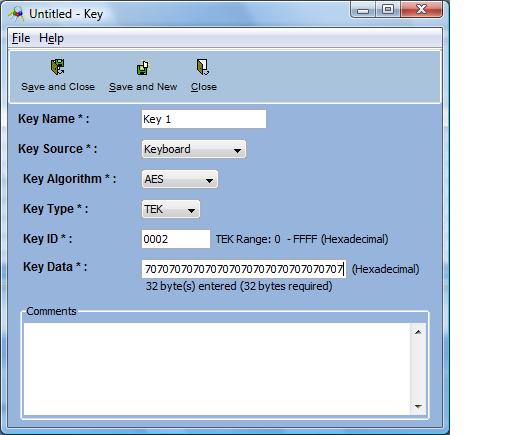
The responses can then be used for verification on the verification simulator by using the KEK and the information provided (and logged) to the KMM platform from the end devices (Initialization vector, key ID, Algorithm ID). The verification simulator can take as input the encrypted data from the response, set up the algorithm with the Initialization vector and the key and generate the plaintext. This is then compared with the plaintext logged by the KMM Test platform

## Entering and comparing information for the verification

The following screenshot shows the test tool for the KMM Test platform. The image shows the key list for a set of AES and DES keys. In the instance of the customer algorithm can be identified in “Key Name” column as shown in window below.

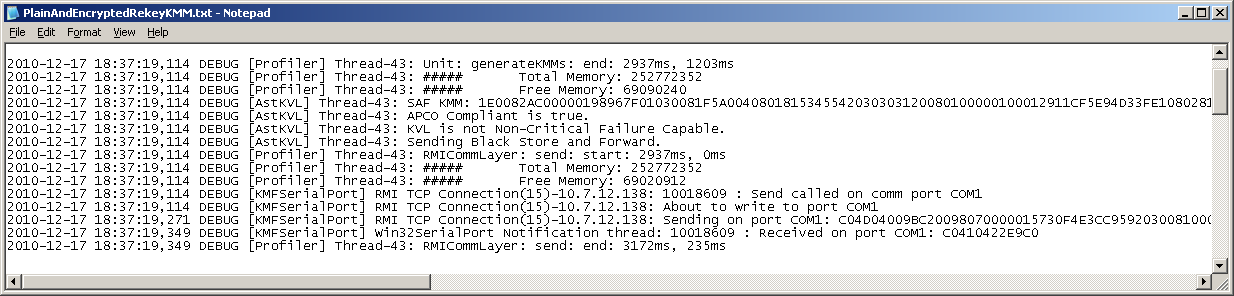


The plain text key is entered through the keyboard as shown on the screen below:



The screen shows the algorithm name, the entry of the key and the data.

The key data will be logged using KMM Test platform logging mechanism. When the key is used for a rekey of a radio a log will be created with the plain text and the cipher text in KMM Test platform. A snapshot of the log is shown below:



Then, the log will be forwarded to verification simulator as shown in the sample below:

**The plaintext KMM is indicated with the SAF KMM prefix which results in line 4 from above:**

2010-12-17 18:37:19,114 DEBUG [AstKVL] Thread-43: SAF KMM: 1E0082AC00000198967F01030081F5A0040801815345542030303120080100000100012911CF5E94D33FE10802815345542030303220080100000100012911CF5E94D33FE10803815345542030303320080100000100012911CF5E94D33FE180FF81080280F000F5A02911CF5E94D33FE180F001F5A12911CF5E94D33FE1810001AA533B7C

**The SAF KMM can be decoded according to the APCO format to be**

KMM Header

1E - MessageID (indicates a Rekey KMM)

0082 - Message Length

AC - Message Format

000001 - Destination RSI

98967F - Source RSI

0103 - Message Number

KMM Body

00 - Decryption Instruction Block

81 - Algorithm ID (DES-OFB) of key used for inner layer encryption

F5A0 - Key ID of key used for inner layer encryption

04 - Number of keysets included in this KMM

08 - Keyset Format of the 1st keyset

01 - Keyset ID of this keyset

81 - Algorithm ID of keys contained in this keyset

5345542030303120 - Keyset Name of this keyset

08 - Key length of keys in this keyset

01 - Number of keys contained in this keyset

00 - Key Format of this key

0001 - SLN of this key

0001 - Key ID of this key

2911CF5E94D33FE1 - Encrypted key variable of this key (DES-OFB keys are 8 octets long)

08 - Keyset Format of the second keyset

02 - Keyset ID of the second keyset

81

5345542030303220080100000100012911CF5E94D33FE10803815345542030303320080100000100012911CF5E94D33FE180FF81080280F000F5A02911CF5E94D33FE180F001F5A12911CF5E94D33FE1810001AA533B7C (remainder of KMM)

**The encrypted KMM is as follows:**

2010-12-17 18:37:19,271 DEBUG [KMFSerialPort] RMI TCP Connection(15)-10.7.12.138: Sending on port COM1: C04D04009BC20098070000015730F4E3CC959203008100010037F183AE3D392379C80583DEB19F9547176121F4CBEAB5A12AAB5A790ED48D017354C2C65EE4621E456DED270E039E9C8BD31ECE99DBDCD757DFAF94042EE71569A556C36CCE5618BDD03834B821974EB6CCAE4BA878F535449CE6E5E563D002DD07DDC55C3E66DEC1143C8A6A6EDAAC7E1BFAAFAC37D443EFD18D640F370E120E31977B001DE8971156C0

**This is decoded as follows:**

C04D04009BC2009807000001 - KVL APCO Interface Protocol Header

5730F4E3CC95920300 - Message Indicator

81 - Algorithm ID of the key used to encrypt the KMM

0001 - Key Id of the key used to encrypt the KMM

00 - Unused field

**The encrypted cipher (which equals the removal of KMM Header and body from the plain text)**

37F183AE3D392379C80583DEB19F9547176121F4CBEAB5A12AAB5A790ED48D017354C2C65EE4621E456DED270E039E9C8BD31ECE99DBDCD757DFAF94042EE71569A556C36CCE5618BDD03834B821974EB6CCAE4BA878F535449CE6E5E563D002DD07DDC55C3E66DEC1143C8A6A6EDAAC7E1BFAAFAC37D443EFD18D640F370E120E31977B001DE8971156C0

Given that the user has entered the KEK of the rekey KMMs the user can enter the known key and plain text into the verification simulator (test key), enter encrypted cipher text and run the test simulation for an encrypt.

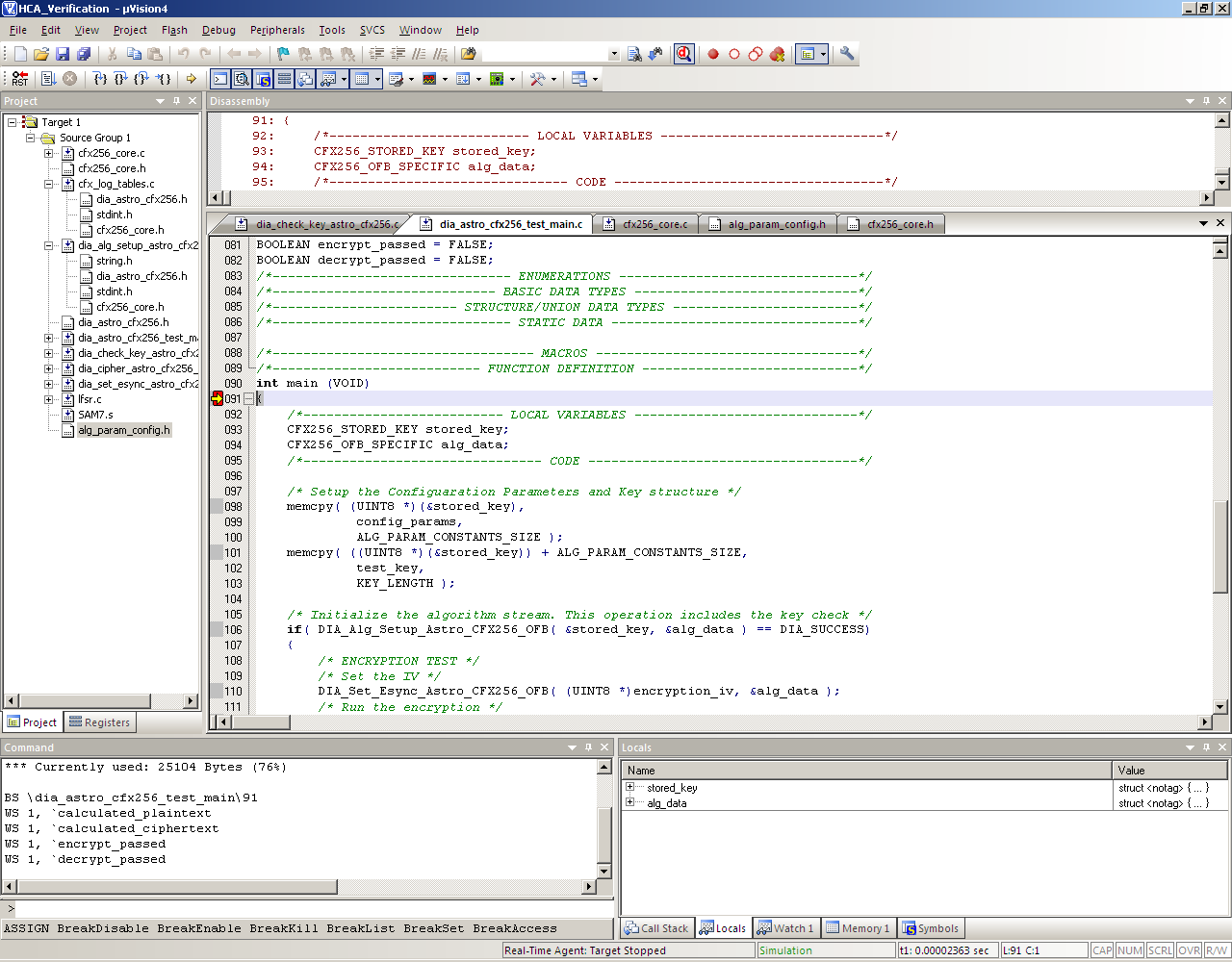


Figure : Entry into verification simulator to perform verification (Sample)

The verification simulator will show whether the encrypt result is a pass or fail and the user can monitor the variables as in the example shown below:

This shows the verification simulator monitoring the pass or fail for the encrypt along with the calculated data (which can be expanded to show all of the data during the simulation).