

Volume

1

A-WIT TECHNOLOGIES INC.

---

... a passion for execution ...

# Data Cryptography with the C Stamp

---

Version 1.0

A-WIT TECHNOLOGIES INC.

# Data Cryptography with the C Stamp Reference Guide Manual

---

© A-WIT Technologies Inc.  
Phone (800) 985-AWIT • Fax (800) 985-2948

---

# Table of Contents

Registering Your C Stamp or C Stamp	
Related Product	1
Notices	2
Getting Support	2
Installing the Microchip MPLAB and C	
Compiler Software	2
Installing the A-WIT C Stamp Quick	
Programmer	4
Installing the USB Software	4
Setting Up the C Stamp Software	
Templates	5
Documentation	5
C Stamp Program Example for RANDOM	
Data Cryptography	5
C Stamp Program Example for PRBS Data	
Cryptography	6
C Stamp Program Example for SKIPJACK	
Data Cryptography	8
C Stamp Program Example for XTEA Data	
Cryptography	9
C Stamp Program Example for AES Data	
Cryptography	11
Setting Up the C Stamp Program Examples	
for Data Cryptography	13
Simulating and Running Your Program	15
Developing Your Own Programs and	
Projects	15
Benchmarks of the Different Algorithms	15
Terms & Conditions	17

---

## Introduction

The purpose of this document is to describe five data encryption and decryption algorithms utilizing the C Stamp microcontroller. The usage of the algorithms is elaborated upon through this document. The five algorithms, in order of complexity from least to most complex, are RANDOM, PRBS (Pseudo-Random Binary Sequence generator, SKIPJACK, XTEA (Tiny Encryption Algorithm Version 2), and AES (Advanced Encryption Standard). This document also offers benchmarks on the memory usage and speed of the different algorithms.

## Registering Your C Stamp or C Stamp Related Product

At A-WIT Technologies we respect your privacy; however, we do ask you to register your C Stamp or C Stamp related product, so you can receive free of charge product updates. The registration procedure is simple. Just send an e-mail to [tech\\_support@awit.com](mailto:tech_support@awit.com) with the word “REGISTRATION x” in the subject line, where “x” is the product number that you purchased. If you purchased more than one product, send an e-mail for each different product.

## Notices

CSTAMP™ and CSTAMP™ Related Hardware Products, Software Products and Documentation are developed and distributed by A-WIT Technologies, Inc. All rights reserved by A-WIT Technologies, Inc. A-WIT SOFTWARE OR FIRMWARE AND LITERATURE IS PROVIDED “AS IS,” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL A-WIT BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY ARISING OUT OF OR IN CONNECTION WITH THE SOFTWARE OR FIRMWARE OR THE USE OF OTHER DEALINGS IN THE SOFTWARE OR FIRMWARE.

MPLAB C-18 and MPLAB C-18 Users Guide is reproduced and distributed by A-WIT Technologies, Inc. under license from Microchip Technology Inc. All rights reserved by Microchip Technology Inc. MICROCHIP SOFTWARE OR FIRMWARE AND LITERATURE IS PROVIDED “AS IS,” WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL MICROCHIP BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY ARISING OUT OF OR IN CONNECTION WITH THE SOFTWARE OR FIRMWARE OR THE USE OF OTHER DEALINGS IN THE SOFTWARE OR FIRMWARE.

## Getting Support

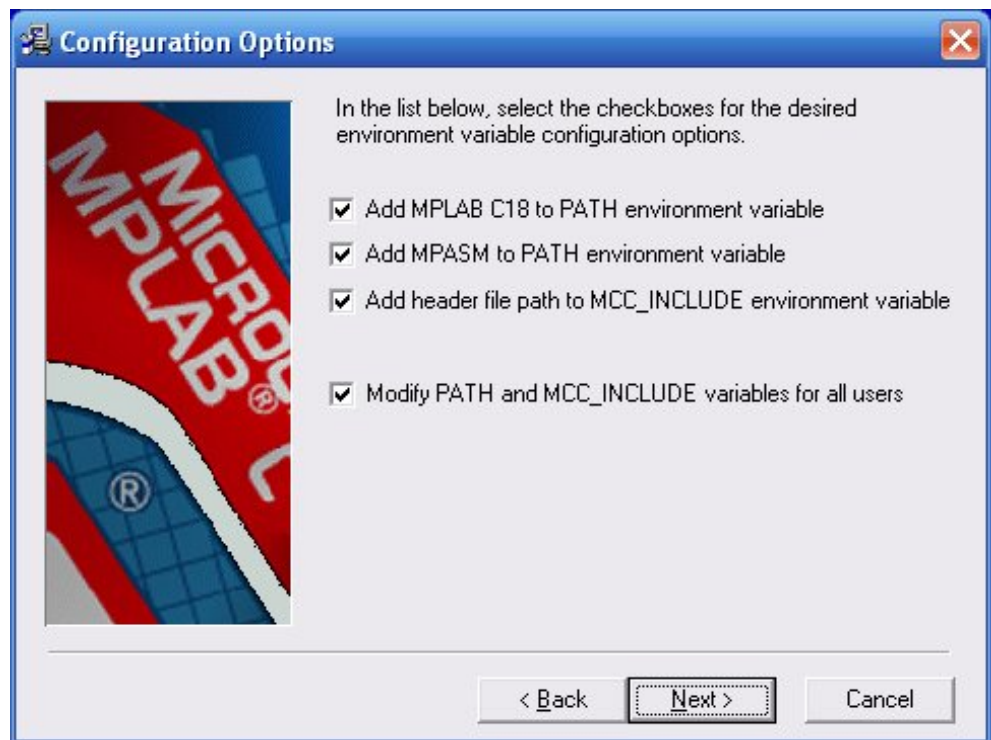
If possible, please check the C Stamp website [www.c-stamp.com](http://www.c-stamp.com) under SUPPORT for any updates to documentation, changes, or notices that may have become available since your Installation CD was produced. If you continue to have any issues for which a solution is not found in the aforementioned website, please e-mail [tech\\_support@a-wit.com](mailto:tech_support@a-wit.com) for help.

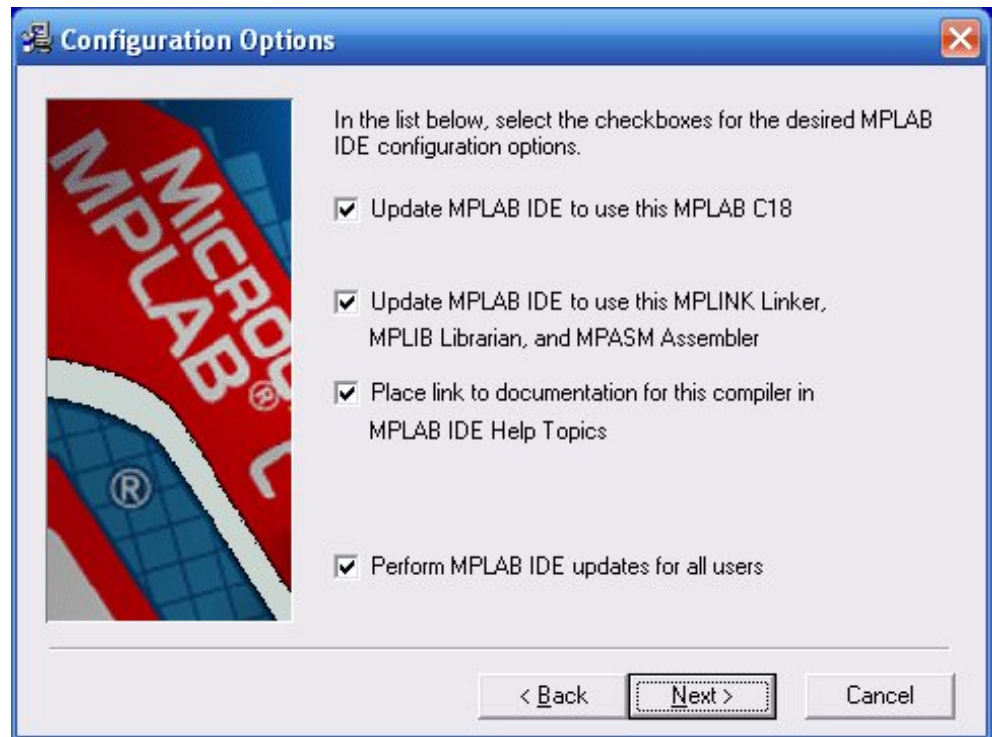
## Installing the Microchip MPLAB and C Compiler Software

The first step is to install the Microchip MPLAB software that you will use to develop your programs.

Insert your A-WIT provided Installation CD in your CD drive. Go to the MPLAB directory in the CD and double click on the “MPLAB vX.XX Install” file in that directory. Follow the installation steps, prompts, and directions provided by the installer software, accepting all the default options.

After the MPLAB installation is complete, switch to the C18 directory in the CD, and double click on the file in that directory. Follow the installation steps, prompts, and directions provided by the installer software, accepting all the default options. The only exceptions to accepting all the default options is that on the 5<sup>th</sup> and 6<sup>th</sup> windows of the installation process for the C18 Compiler, you have to select everything as shown in the figures below. This will ensure that MPLAB is configured to use the C18 Compiler.





## Installing the A-WIT C Stamp Quick Programmer

To install the A-WIT C Stamp Quick Programmer, switch to the CSTAMPQP directory in the CD using Windows Explorer, and double click on the file in that directory. Follow the installation steps, prompts, and directions provided by the installer software, accepting all the default options.

## Installing the USB Software

If you purchased a product with a USB download cable, make sure that the A-WIT provided CD is in the CD drive of your PC and insert the USB cable in the USB port of your PC. Windows auto detects the new USB device. If Windows prompts you to install drivers for the USB cable device, follow the installation steps, prompts, and directions provided by the installer software, accepting all the default options.

After the USB adapter has been installed, open a Windows Explorer window from the Accessories sub-menu in the Start menu, and right click on My Computer. Proceed to select Properties, and then select the Hardware tab. Click on the Device Manger button, and expand the Ports (COM & LPT) branch. Make a note of the COM port that has been assigned to the USB-to-Serial adapter. This is the port that should be selected in the C Stamp programmer software.

## Setting Up the C Stamp Software Templates

To set up the C Stamp Software Templates, switch to the CSTAMP\_Template directory in the CD using Windows Explorer, and double click on the file in that directory. Follow the installation steps, prompts, and directions provided by the installer software, accepting all the default options.

## Documentation

Copy the DOCS directory from the C Stamp Installation CD to your C:\A-WIT directory. This directory contains all the C Stamp related documentation in PDF format.

## C Stamp Program Example for RANDOM Data Cryptography

The RANDOM algorithm can operate on a variable length data block and uses a 16-bit key. The same function is used for encryption and decryption. The following program is a demo of how to use the C Stamp software infrastructure for cryptography. It shows how a system could encrypt and decrypt data.

The sample program is as follows:

```
#include "CS110000.h"

void main(void)
{
// TYPE YOUR CODE HERE AFTER THIS LINE
  BYTE messageRND1[] = "It was t";
  BYTE messageRND2[9];
  WORD Key = 0x5628;
  BYTE outdata[9];

  outdata[8] = 0;
  messageRND2[8] = 0;

//RND encryption and decryption
  printf("Message before encryption\r\n");
  printf("%s\r\n", messageRND1);
  RNDCrypt(messageRND1, 8, Key, outdata);
  printf("\r\nMessage after encryption\r\n");
  printf("%s\r\n", outdata);
  printf("\r\nMessage is encrypted\r\n");
  printf("Message is now going to be decrypted!\r\n");
```

```

RNDCrypt(outdata, 8, Key, messageRND2);
printf("r\nMessage after decryption\r\n");
printf("s\r\n", messageRND2);
printf("r\nMessage done being decrypted");
printf("by RND\r\n");

STOP();

}

```

After this program is simulated, the output is:

```

Message before encryption
It was t

```

```

Message after encryption
W=RiWD,

```

```

Message is encrypted
Message is now going to be decrypted!

```

```

Message after decryption
It was t

```

```

Message done being decrypted by RND

```

The following describes the function available to do the encryption and decryption:

```

void RNDCrypt(BYTE indata[], BYTE L, WORD inkey,
               BYTE outdata[]);

```

Encrypts or decrypts **indata** of length **L** with key **inkey**, and returns the results in **outdata**. **outdata** must be at least **L** bytes long.

## C Stamp Program Example for PRBS Data Cryptography

The PRBS algorithm can operate on a variable length data block and uses an 8-bit key. The same function is used for encryption and decryption. The following program is a demo of how to use the C Stamp software infrastructure for cryptography. It shows how a system could encrypt and decrypt data.

The sample program is as follows:

```

#include "CS110000.h"

```

```

void main(void)
{
// TYPE YOUR CODE HERE AFTER THIS LINE
  BYTE messagePRBS1[] = "It was t";
  BYTE messagePRBS2[9];
  BYTE Key = 0x28;
  BYTE outdata[9];

  outdata[8] = 0;
  messagePRBS2[8] = 0;

//PRBS encryption and decryption
  printf("Message before encryption\r\n");
  printf("%s\r\n", messagePRBS1);
  PRBSCrypt(messagePRBS1, 8, Key, outdata);
  printf("\r\nMessage after encryption\r\n");
  printf("%s\r\n", outdata);
  printf("\r\nMessage is encrypted\r\n");
  printf("Message is now going to be decrypted!\r\n");
  PRBSCrypt(outdata, 8, Key, messagePRBS2);
  printf("\r\nMessage after decryption\r\n");
  printf("%s\r\n", messagePRBS2);
  printf("\r\nMessage done being decrypted");
  printf("by PRBS\r\n");

  STOP();
}

```

After this program is simulated, the output is:

Message before encryption

It was t

Message after encryption

a`\*r P2=

Message is encrypted

Message is now going to be decrypted!

Message after decryption

It was t

Message done being decrypted by PRBS

The following describes the function available to do the encryption and decryption:

```
void PRBSCrypt(BYTE indata[], BYTE L, BYTE inkey,
                BYTE outdata[]);
```

Encrypts or decrypts **indata** of length **L** with key **inkey**, and returns the results in **outdata**. **outdata** must be at least **L** bytes long.

## C Stamp Program Example for SKIPJACK Data Cryptography

The SKIPJACK algorithm was developed in the early 1980s by NSA for governmental documents. It was classified SECRET until 1998. The SKIPJACK algorithm utilizes an 80-bit key on a 64-bit block of data. The following program is a demo of how to use the C Stamp software infrastructure for cryptography. It shows how a system could encrypt and decrypt data.

The sample program is as follows:

```
#define    __SJCRYPTO
#include  "CS110000.h"

void main(void)
{
// TYPE YOUR CODE HERE AFTER THIS LINE
  BYTE messageSJ1[] = "It was t";
  BYTE messageSJ2[9];
  BYTE Key[] = "Charles Di";
  BYTE outdata[9];

  outdata[8] = 0;
  messageSJ2[8] = 0;

//Skipjack encryption and decryption
  printf("Message before encryption\r\n");
  printf("%s\r\n", messageSJ1);
  SkipJackEncrypt(messageSJ1, Key, outdata);
  printf("\r\nMessage after encryption\r\n");
  printf("%s\r\n", outdata);
  printf("\r\nMessage is encrypted\r\n");
  printf("Message is now going to be decrypted!\r\n");
  SkipJackDecrypt(outdata, Key, messageSJ2);
  printf("\r\nMessage after decryption\r\n");
  printf("%s\r\n", messageSJ2);
```

```

printf("\\r\\nMessage done being decrypted");
printf("by SkipJack\\r\\n");

STOP();

}

```

After this program is simulated, the output is:

```

Message before encryption
It was t

```

```

Message after encryption
S Ni)nt

```

```

Message is encrypted
Message is now going to be decrypted!

```

```

Message after decryption
It was t

```

```

Message done being decrypted by SkipJack

```

The following describes the functions available to do the encryption and decryption:

```

void SkipJackEncrypt(BYTE indata[], BYTE inkey[],
                     BYTE outdata[]);

```

Encrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 8 bytes long, and the key must be 10 bytes long.

```

void SkipJackDecrypt(BYTE indata[], BYTE inkey[],
                     BYTE outdata[]);

```

Decrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 8 bytes long, and the key must be 10 bytes long.

## C Stamp Program Example for XTEA Data Cryptography

The XTEA algorithm is an adaptation of the original TEA algorithm and was created by David Wheeler and Roger Needham from the Cambridge Computer Lab. The XTEA is famous for its small size relative to its security strength. The XTEA algorithm utilizes a 128-bit key on a 64-bit block of data. The following program is a demo of

how to use the C Stamp software infrastructure for cryptography. It shows how a system could encrypt and decrypt data.

The sample program is as follows:

```
#include "CS110000.h"

void main(void)
{
// TYPE YOUR CODE HERE AFTER THIS LINE
  BYTE messageXTEA1[] = "It was t";
  BYTE messageXTEA2[9];
  BYTE Key[] = "Charles Dickens.";
  BYTE outdata[9];

  outdata[8] = 0;
  messageXTEA2[8] = 0;

//XTEA encryption and decryption
  printf("Encrypting now using XTEA\r\n");
  printf("r\nOriginal Message: %s\r\n", messageXTEA1);
  XTEAEncrypt(messageXTEA1, Key, outdata, 3);
  printf("r\nEncrypted Message: %s\r\n", outdata);
  printf("r\nDecrypting now using XTEA\r\n");
  XTEADecrypt(outdata, Key, messageXTEA2, 3);
  printf("r\nDecrypted Message: %s\r\n",
        messageXTEA2);

  STOP();
}

```

After this program is simulated, the output is:

```
Encrypting now using XTEA
Original Message: It was t
Encrypted Message: )3vS : l
Decrypting now using XTEA
Decrypted Message: It was t

```

The following describes the functions available to do the encryption and decryption:

```
void XTEAEncrypt(BYTE indata[], BYTE inkey[],  
                 BYTE outdata[], NIBBLE St);
```

Encrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 8 bytes long, and the key must be 16 bytes long. **St** is a parameter that sets the relative security strength for the XTEA algorithm. The possible values of **St** are 1, 2, and 3; which denote increasing security strength.

```
void XTEADecrypt(BYTE indata[], BYTE inkey[],  
                 BYTE outdata[], NIBBLE St);
```

Decrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 8 bytes long, and the key must be 16 bytes long. **St** is a parameter that sets the relative security strength for the XTEA algorithm. The possible values of **St** are 1, 2, and 3; which denote increasing security strength.

## C Stamp Program Example for AES Data Cryptography

The AES (Advanced Encryption Standard) algorithm, which is the most complex of the implementations in this document, was adopted by the National Institute of Standards and Technology (NIST) on October 2, 2000. The algorithm has been named "Rijndael" after Joan Daemen and Vincent Rijmen of Belgium. The algorithm is based on a 16-byte block of data and a 16-byte key size (in our case). The key can also be extended to 24 and 32 Bytes. The following program is a demo of how to use the C Stamp software infrastructure for cryptography. It shows how a system could encrypt and decrypt data.

The sample program is as follows:

```
#define    __AESCRYPTO    // need for both  
#define    __AESCRYPTOD  // need for decryption only  
#include   "CS110000.h"  
  
void main(void)  
{  
// TYPE YOUR CODE HERE AFTER THIS LINE  
  BYTE messageAES1[] = "It was the best ";  
  BYTE messageAES2[17];  
  BYTE Key[] = "Charles Dickens."  
  BYTE outdata[17];  
  
  outdata[16] = 0;
```

```

messageAES2[16] = 0;

//AES encryption and decryption
printf("Encrypting now using AES\r\n");
printf("\r\nOriginal Message: %s\r\n", messageAES1);
AESEncrypt(messageAES1, Key, outdata);
printf("\r\nEncrypted Message: %s\r\n", outdata);
printf("\r\nDecrypting now using AES\r\n");
AESDecrypt(outdata, Key, messageAES2);
printf("\r\nDecrypted Message: %s\r\n", messageAES2);

STOP();

}

```

After this program is simulated, the output is:

```

Encrypting now using AES

Original Message: It was the best

Encrypted Message: ?Xi D PL' $`d]vJ

Decrypting now using AES

Decrypted Message: It was the best

```

The following describes the functions available to do the encryption and decryption:

```

void AESncrypt(BYTE indata[], BYTE inkey[],
               BYTE outdata[]);

```

Encrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 16 bytes long, and the key must be 16 bytes long.

```

void AESDecrypt(BYTE indata[], BYTE inkey[],
                BYTE outdata[]);

```

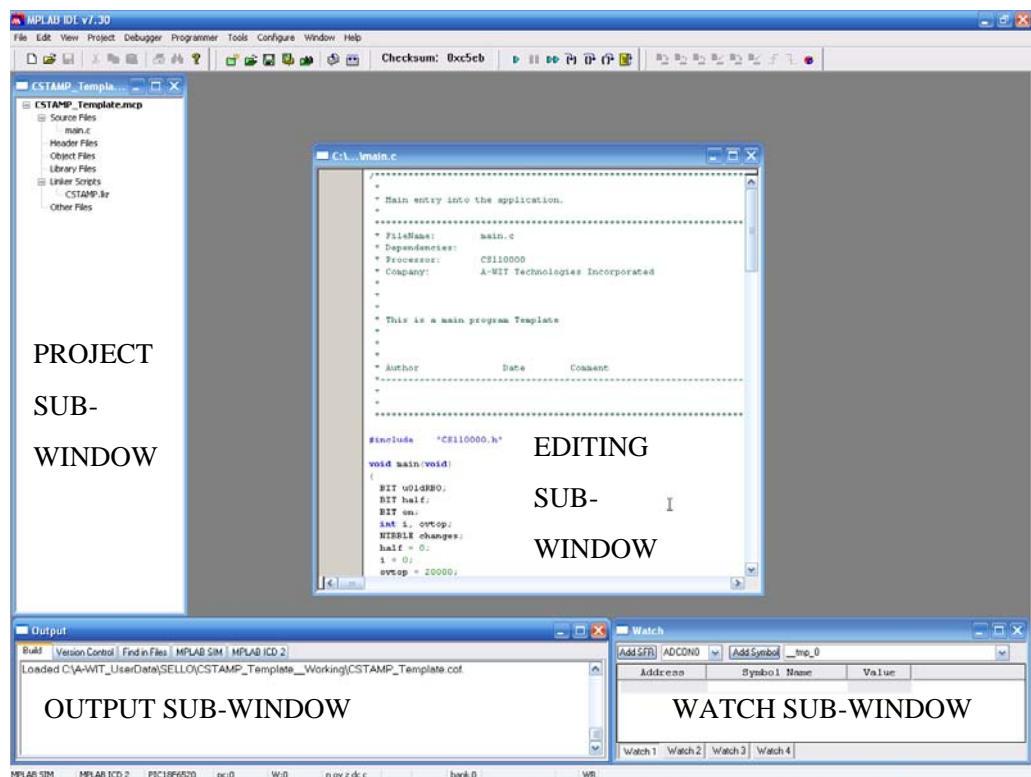
Decrypts **indata** with key **inkey**, and returns the results in **outdata**. **indata** and **outdata** must be 16 bytes long, and the key must be 16 bytes long.

## Setting Up the C Stamp Program Examples for Data Cryptography

Create a directory where you want to have all the files for your program; for example CRYPTO\_APP. We recommend making this directory under your C:\A-WIT directory, so you can have all your CSTAMP related files in one place.

Copy the all files in your C Stamp Software Templates directory C:\A-WIT\CSTAMP\_Template to the directory you just made.

Open the Microchip MPLAB IDE application. As shown the following figure, the IDE has several sub-windows. Depending on the resolution of your screen, your sub-windows may have a different layout. However; you can move and resize these into the position that you want to fit your screen, and your layout for that particular project will get saved upon answering yes to the prompt of saving the workspace when you exit the software development environment.



Go to the “File” menu to “Open Workspace...”. Then navigate to your program directory, and open `CSTAMP_Template.mcw`.

Right click on `CSTAMP_Template.mcp` in the “Project” sub-window, and “Save as...” the name of your program project after you have navigated to your program directory. For example, your program project could be named “CRYPTO\_APP”. Now when you open the Microchip MPLAB IDE (Integrated Development Environment), and go to your program directory to open the workspace for your program, you will see a `.mws` file with the name of your program preceding it. This is the file that you should open any time you want to work on your program.

Double click on the `main.c` source and type the following code fragment where it is indicated. You can omit the comments for brevity, as they are written here to offer clarifications of what the code does. Do pay attention; however, to the indentation of the code blocks between curly brackets for loops, if statements, etc. Although indenting the code is not a requirement for the compiler to parse your code (i.e. any blank spaces are ignored by the compiler), it does help tremendously to make your code much more readable, and consequently, it makes finding any errors easier. Keywords and function names in the code are bolded.

After you simulate your C Stamp program explained in the “Simulating and Running Your Program” section, the program will run.

Save your program from the “File” menu or by clicking on the appropriate icon in the tool bar. Then “Build All” from the “Project” menu or from the tool bar.

If the code was typed correctly, the program will build.

If you get an error message or an indication that your program did not build successfully in the “Output” sub-window of the IDE, you probably have one or more syntax error(s). Double click on the line of the “Output” sub-window that mentions the error, and the program line that most likely contains the error will be indicated in the sub-window where you were editing your program. Correct as necessary and “Build All” again until you get a successful build output.

## Simulating and Running Your Program

Double click on the program line that has the STOP command to insert a breakpoint at that line. Then click on the Play button, and the program will run and produce the intended output.

## Developing Your Own Programs and Projects

Now that you have successfully developed and run your program, it is easy to move on to more complex and elaborate projects of your own.

## Benchmarks of the Different Algorithms

The following Table compares the different cryptography algorithms in terms of relative security, speed per byte, and memory utilization.

<i>Algorithm</i>	<i>Relative Security (1 is lowest)</i>	<i>Speed/Byte (<math>\mu</math>S)</i>	<i>Program Memory (Bytes)</i>	<i>Data Memory (Bytes)</i>
<b>RANDOM</b>	2	10.8	570	2
<b>PRBS</b>	1	8.1	278	0
<b>SKIPJACK Encryption</b>	3	1102.2	4320	0
<b>SKIPJACK Decryption</b>	3	1104.3	4444	2
<b>XTEA St=1 Encryption</b>	4	161.9	2102	0
<b>XTEA St=1 Decryption</b>	4	163.5	2186	19
<b>XTEA St=2 Encryption</b>	5	282.3	2102	0
<b>XTEA St=2 Decryption</b>	5	284.3	2186	19
<b>XTEA St=3 Encryption</b>	6	523.1	2102	0

<i>Algorithm</i>	<i>Relative Security (1 is lowest)</i>	<i>Speed/Byte (<math>\mu</math>S)</i>	<i>Program Memory (Bytes)</i>	<i>Data Memory (Bytes)</i>
<b>XTEA St=3 Decryption</b>	6	525.9	2186	19
<b>AES Encryption</b>	7	519.1	1700	5
<b>AES Decryption</b>	7	1776.1	2353	14

## Terms & Conditions

### **Quality Assurance**

A-WIT has stringent quality control procedures in place to insure the best quality products.

### **90-Day Limited Warranty**

A-WIT Technologies, Inc warrants its products against defects in materials and workmanship for a period of 90 days. If you discover a defect, A-WIT Technologies, Inc. will, at its option, repair, replace, or refund the purchase price. After 90 days, products can still be sent in for repair or replacement, but there will be a \$10.00USD minimum inspection/labor/repair fee (not including return shipping and handling charges).

### **14-Day Money-Back Guarantee**

If, within 14 days of having received your product, you find that it does not suit your needs, you may return it for a refund. A-WIT will refund the purchase price of the product in the form of a check, excluding shipping/handling costs, once the product is received. This refund does not apply if the product has been altered or damaged. If you decide to return the products after the 14-day evaluation period, a 20% restocking fee will be charged against a credit.

### **Disclaimer**

Warranty does not apply if the product has been altered, modified, or damaged. A-WIT makes no other warranty of any kind, expressed or implied, including any warranty of merchantability, fitness of the product for any particular purpose even if that purpose is known to A-WIT, or any warranty relating to patents, trademarks, copyrights or other intellectual property. A-WIT shall not be liable for any injury, loss, damage, or loss of profits resulting from the handling or use of the product shipped.

### **How to Return a Product**

When returning, you must first e-mail [sales@a-wit.com](mailto:sales@a-wit.com) for a Return Merchandise Authorization number. No packages will be accepted without the RMA number clearly marked on the outside of the package. After inspecting and testing, we will return your product, or its replacement using the same shipping method used to ship the product to A-WIT within 30 days. In your package, please include a daytime telephone number and a brief explanation of the problem.

Please contact our Sales Department at [sales@a-wit.com](mailto:sales@a-wit.com) if you have any questions regarding our warranty policy or if you are requesting an RMA number.

