Message Authenticator

Sign and Validate Any Message

Table of Contents

Overview

Message Authenticator (*MsgAuth*) is a small, simple program that is useful for signing and validating any message. For the purposes of this manual radio communication language will be used, but *MsgAuth* may be used with any form of message, from paper to SMS, email to smoke signal.

MsgAuth works by utilizing a strong hash function during the generation of a CRC. The sender and recipient are both included with the message when the CRC is generated, and are part of the authenticated data.

Prerequisites

The application will run under Windows, Linux, or MacOS, and is suitable for use on a Raspberry Pi. A functional Python3 installation is required only when not using the Windows binary.

Windows users

Download the latest version from https://kf7mix.com/msgauth.html then locate the file, right-click and Extract All. A new folder should be created for *MsgAuth*. Enter the folder and double-click the application icon. You may need to install the Visual C++ Redist package if your system doesn't already have it.

Linux Users

You will require a functional Python3 configuration with Tkinter libraries installed. On a default Linux Mint installation (where Python3 is included), for example, the following command should suffice to setup a basic environment: "sudo apt install python3-tk"

Upgrading From a Previous Version

You may keep as many versions as you like, and run the one you need. The application has no hidden or system files, and no registry entries, and therefore cannot conflict with other versions stored in other folders.

To upgrade, simply download the latest version, unpack, and use it. Both operators should use the same version of the program to sign and validate.

Software Settings

The settings for *MsgAuth* are stored in a configuration file named msgauth.cfg, in the same folder as the program. Your config file should have at least three lines:

- Line 1 must read either "light" or "dark" with no quotes, to set the desired theme.
- Line 2 contains your callsign or other designator
- Line 3 must contain your first key
- All additional lines from 4 onward contain keys, one key per line

Technical Details

Key Creation, Exchange, and Maintenance

The key is a string, saved one per line starting on Line 3 of the configuration file. It is recommended that users utilize only capital letters (A-Z) and numbers (0-9), but any UTF-8 character is allowed. If multiple keys are used, you may wish to prepend additional characters to identify the key. The following are all valid example keys (for demonstration, not for actual use):

- 2A83MBLHA2GUA77I99BSTUASZ
- MTAMIODEL3FYH4W6RMVIVBFCES43W6S3U0GY
- 1: TQT4D8IHDAU85F06F39S16K3GR83
- 2: 1G1U9RQNDETM2BJUKIVHOGTP0H2YBHKF94X05LAR
- BOB=B9DBRP40DA4O4076HWGM1T9D2DYZLRFVMQ3CTXQ
- pτ-Ҧђ(ʔ¾Ʊ+эȜṄj5U.Aḉڪцinľz͙─}ŞˁɆɄ̄Ȼ1:ăǪcƋў̱ꝏχ

PLEASE NOTE: Each individual line above is a key. The space character is treated like any letter/number, as are characters like :, =, etc. If you choose to use UTF-8 extended characters, it is up to both parties to ensure they can support that encoding; ASCII characters are recommended for most users.

Keys should ***never*** be exchanged over the air. To do so would completely eliminate any value in the entire sign/validate process. Exchange keys through the most secure means available (secure email, in-person, etc.)

Keys do not last forever. The more a key is used, the higher chance bad actors have of cracking them. Consider strategies such as exchanging larger key sets and rotating through them, creating new keys monthly, etc.

Keys should not generally be composed of publicly available information (quotes, names, dates, phone numbers, etc.). Random characters are ideal.

The software includes a random key generator. You may use keys generated through this tool, or through any other method.

Large Messages

You may sign and validate a short message, or a long one. Large messages should be composed in a viable text editor, such as Notepad++ on Windows, or Xed on Linux. Once a message is completed and ready for transmission, copy and paste it into *MsgAuth*. After signing your message, copy your CRC and optional Datecode and append them to your original message in your text editor. Make sure there is a space after your original message and the information you're adding. Then, copy the full message + optional Datecode + CRC and transmit as needed.

How the CRC is Generated

The three-character (A-Z0-9) CRC is generated using the following steps:

1. Normalize the input:
    1. Stripping extra spaces from the Target and Source inputs (removed)
    2. Stripping control characters from the Message input (converted to spaces)
    3. Reducing any double spaces to single spaces
2. Combine the inputs:
    1. Add the Source, Target, and message together into a single string
    2. Convert the string to uppercase letters
    3. Add the optional Datecode if selected
3. Process the normalized and combined inputs, coupled with the key (the full string):
    1. Run the full string through a sha256 hash with UTF-8 encoding
    2. Run the output of the previous step through the same hash function and convert to a decimal number
    3. Use the decimal number to create a base 36 (A-Z0-9) string
    4. Capture the last three characters and return them as the CRC

Once generated, the three-character CRC is appended (preceded by a space) to the original message before transmission.

How Does the Datecode Work?

A Datecode is a short code (4 character plus an identifier) that represents a date and time stamp with useful resolution/accuracy. It provides vital information in very few characters. It is calculated as follows:

First Character	Month, represented as a single letter: A-L = 1-12
Second Character	Day, represented as a single letter or number: A-Z = 1-26, 0-4 = 27-31
Third Character	Hour, represented as a single letter: A-W = 0-23
Fourth Character	Minute, represented as a single letter, 2min resolution: A-Z, 0-3 (A=0, B=2, C=4, etc.)

In *MsgAuth*, the Datecode is provided along with the CRC, if the option is enabled. The Datecode is technically a part of your full message and must be included for your CRC to pass. Datecodes are especially important if you transmit the same information repeatedly (such as net checkins, or sitrep forms). This prevents bad actors from imitating messages, as the CRC and Datecode will differ even if the rest of the message remains the same.

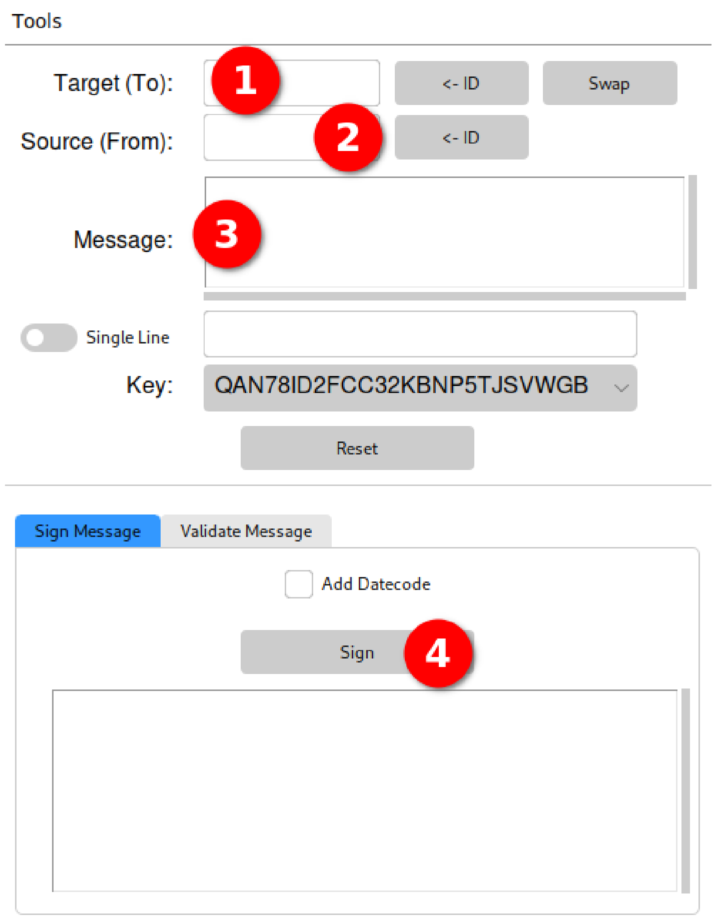
QuickStart

To sign a message, start the program, enter the to/from and message information, and click "Sign". Transmit your message just how you entered it in the message field, but add on a space plus the CRC at the end.

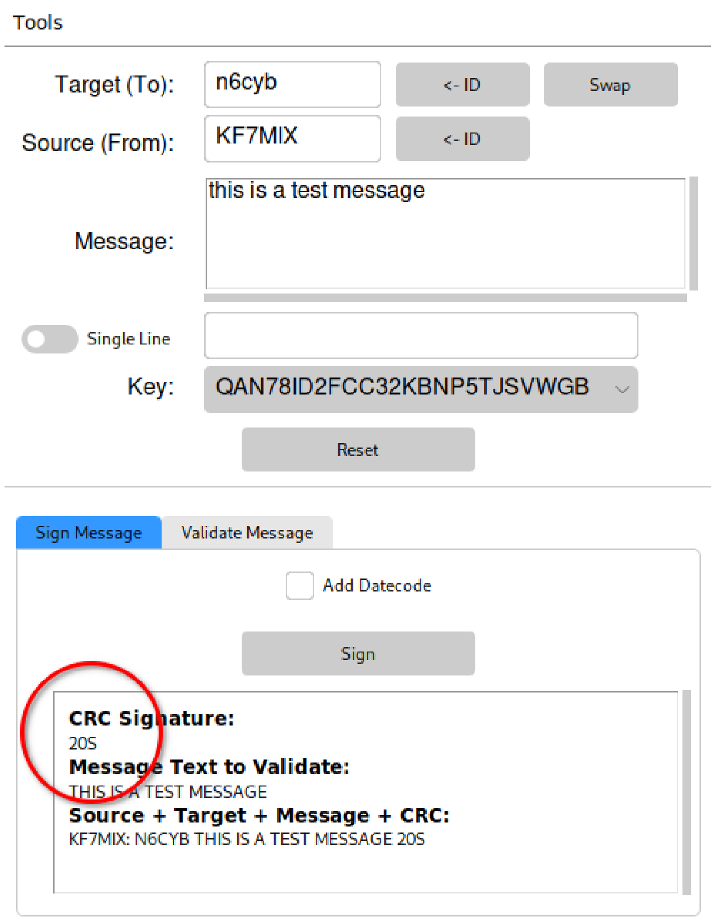
To validate a signed message, start the program, and enter the to/from info and message as it was received. Click the Validate tab. Remove the CRC from the end of the message and put it in the CRC box, then click "Validate".

Detailed Example

Let's say the operator KF7MIX wants to send a short test message to operator N6CYB using the key QAN78ID2FCC32KBNP5TJSVWGB. They open up the program and start entering the information: 1) the target station, in this case N6CYB; 2) the source station, KF7MIX; 3) the message "this is a test message"; 4) select the right key and click "Sign".



Key: QAN78ID2FCC32KBNP5TJSVWGB



Target (To): n6cyb

Source (From): KF7MIX

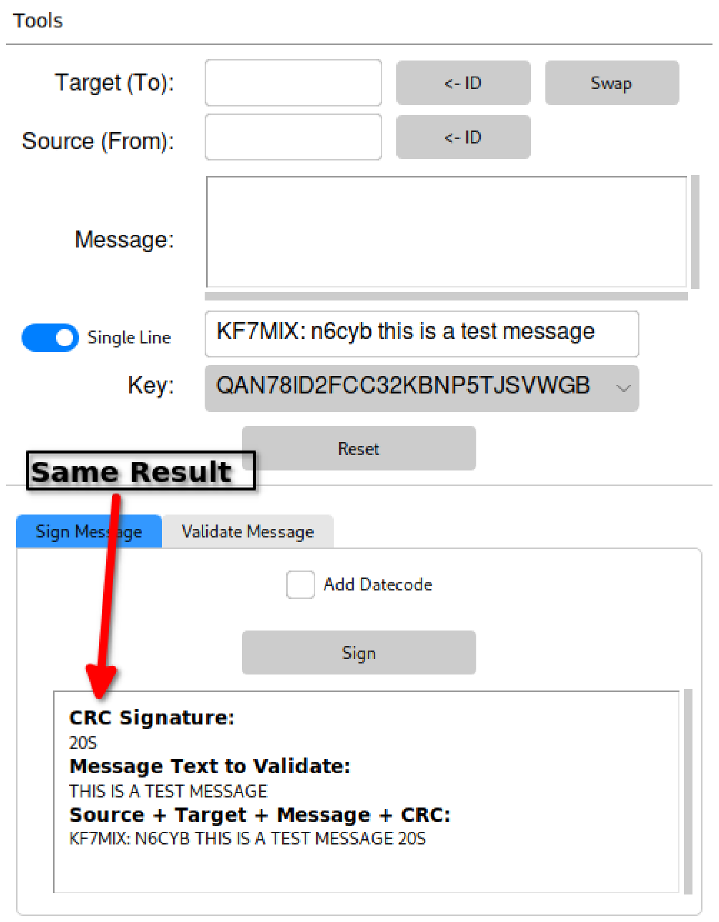
Message: this is a test message

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC Signature:
20S
Message Text to Validate:
THIS IS A TEST MESSAGE
Source + Target + Message + CRC:
KF7MIX: N6CYB THIS IS A TEST MESSAGE 20S

In the above example, the CRC generated was **20S**. Copy and save that, so that you can transmit it along with your message (add a space and the CRC at the end of your message).

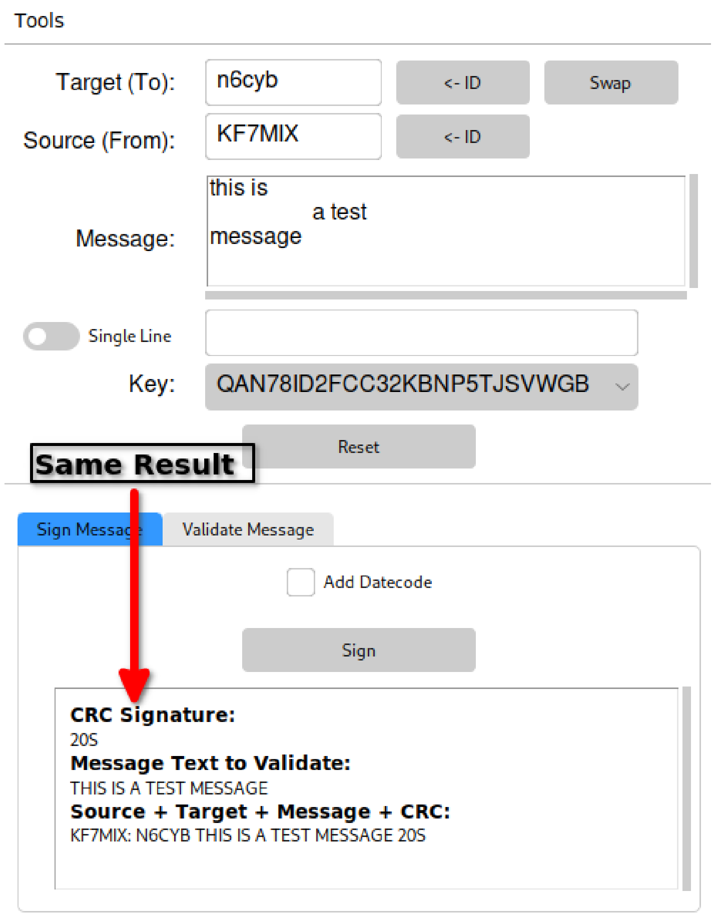
But, let's say KF7MIX wanted to try out the "Single Line" feature, or perhaps he wants to format his text a little using line breaks and tabs. First he clicks the "Single Line", then he clicks "Sign" again, producing the same CRC. Finally, he switches back to multi-line entry and modified his message with control characters like line breaks and tabs, then clicks "Sign" again, producing once again the same result.



KF7MIX: n6cyb this is a test message

QAN78ID2FCC32KBNP5TJSVWGB

CRC Signature:
20S
Message Text to Validate:
THIS IS A TEST MESSAGE
Source + Target + Message + CRC:
KF7MIX: N6CYB THIS IS A TEST MESSAGE 20S

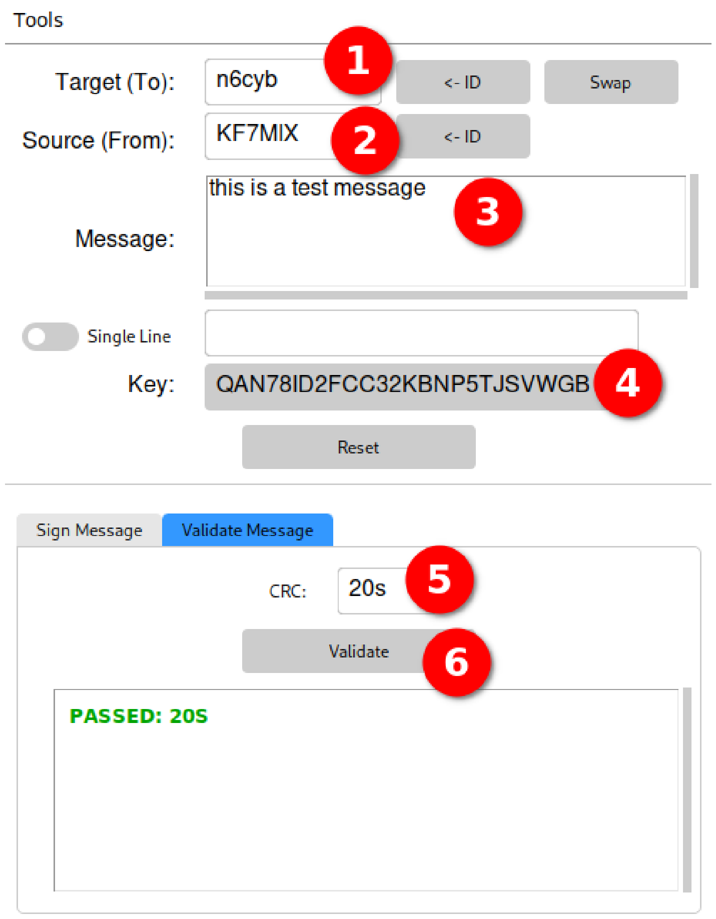


Target (To): n6cyb
Source (From): KF7MIX
Message:
this is
	a test
message

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC Signature:
20S
Message Text to Validate:
THIS IS A TEST MESSAGE
Source + Target + Message + CRC:
KF7MIX: N6CYB THIS IS A TEST MESSAGE 20S

Now, N6CYB will of course want to validate the message he received. He does this by entering the information that was transmitted, and using the Validate tab. He enters, 1) the target, which is himself; 2) the source, which was KF7MIX; 3) the message, as received but without the CRC; 4) he selects the same key that he and KF7MIX exchanged securely; 5) he enters the CRC as it was received; 6) he clicks "Validate".



Target (To): n6cyb

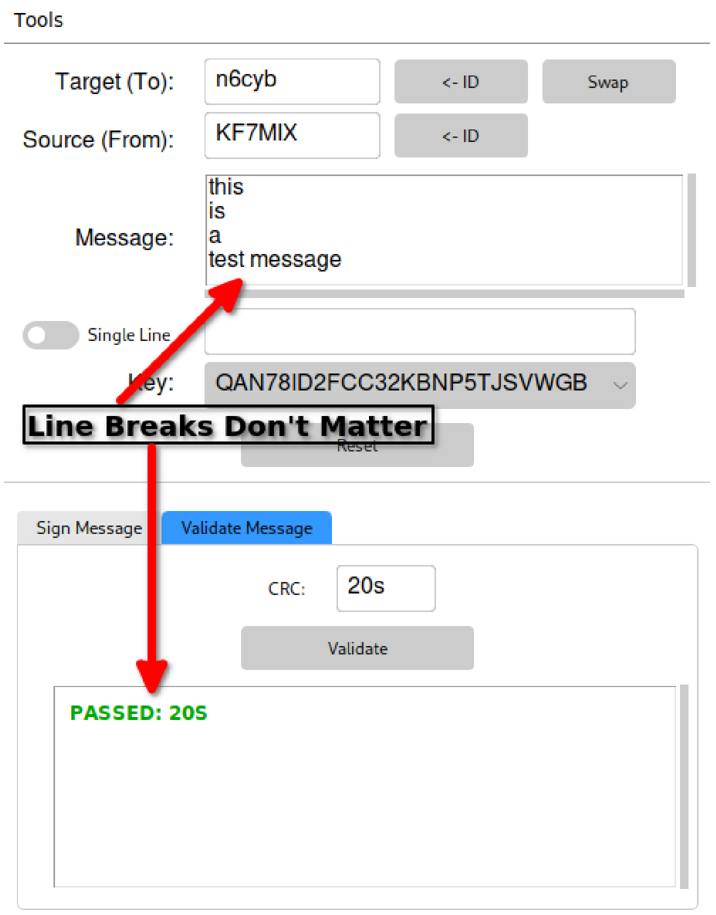
Source (From): KF7MIX

Message: this is a test message

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC: 20s

PASSED: 20S



Target (To): n6cyb

Source (From): KF7MIX

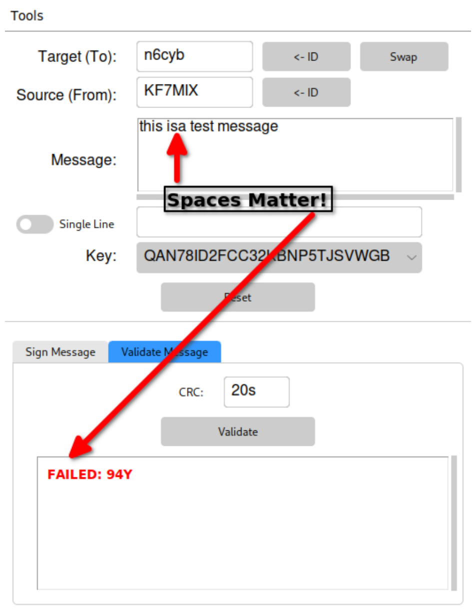
Message:
this
is
a
test message

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC: 20s

PASSED: 20S

At this point, N6CYB can say with reasonable surety that the message was received correctly and that it originates from KF7MIX. Of course, N6CYB likes to experiment (i.e. break things), so he starts to fiddle with the program a little to see what is allowed. First, he tries breaking up the message by pressing enter after a few of the words. He clicks "Validate" again, and finds that line breaks and control characters don't matter—the message validates just fine.



Tools

Target (To): n6cyb

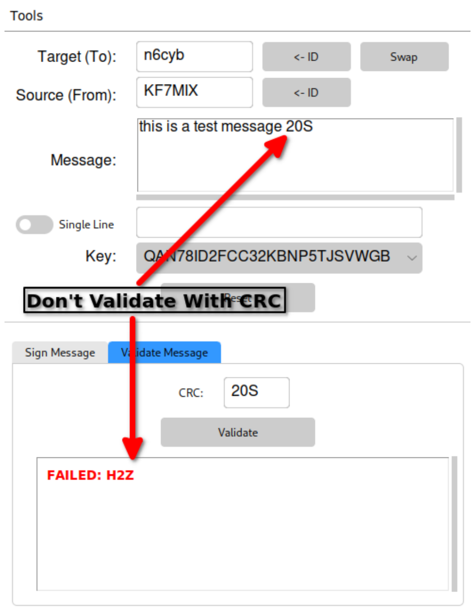
Source (From): KF7MIX

Message: this isa test message

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC: 20s

FAILED: 94Y



Tools

Target (To): n6cyb

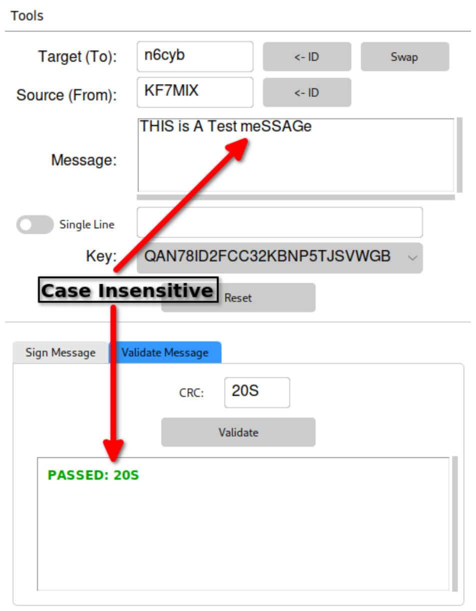
Source (From): KF7MIX

Message: this is a test message 20S

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC: 20S

FAILED: H2Z



Tools

Target (To): n6cyb

Source (From): KF7MIX

Message: THIS is A Test meSSAGe

Key: QAN78ID2FCC32KBNP5TJSVWGB

CRC: 20S

PASSED: 20S

Next, he tries removing a space between the words "is" and "a". This breaks the validation, and he learns that a space is a valid character that must be preserved. He also tries to copy the whole message, including the CRC, and validate that, which fails. Finally, he tries messing with the capitalization of the message, and learns that the program ignores capitalization when it signs and validates.

Getting Additional Help

Additional help, insights, and videos are available online:

https://kf7mix.com/msgauth.html