Encryption and Mac'ing

In the field of cryptography, there are three general maxims commonly represented with the acronym C.I.A. C.I.A. refers to the three ideals of confidentiality, integrity, and availability. Confidentiality refers to the prevention of third parties from intercepting and correctly interpreting a message. Integrity means that the contents of a message can be shown to have originated from a specific source without being tampered with. Availability means that messages are able to be shared freely among participants. In a perfect system, all information transferred would have these three attributes.

Our primary procedure for ensuring confidentiality is encryption. Through this process, messages are scrambled into 'ciphertext' before being transmitted such that third parties are unable to view the original message. Many systems exist to fill this need, such as AES or RSA. Simply using standard encryption may ensure that we are protected from evesdroppers, but it does nothing to ensure we are interacting with who we think we are. This is the case because the encrypted message (ciphertext) is simply a large number of seemingly random bytes. The seemingly random nature of ciphertext makes it easy for an attacker to generate. All they have to do is fire up a pseudo-random number generator and create their own ciphertext.

One example of why this is problematic can be seen by considering the following situation. Imagine you have written a piece of software controlling the launch of nuclear missiles. Your software is connected to the internet to allow the president to launch the missiles from anywhere in the world. When a new connection is received, your software reads in 128 bites from the connection, decrypts it, and checks the first 2 bytes for the president's unique passphrase 'Hi'. Due to the advanced encryption used, when the president does send the command to fire the missiles, nobody will be able to intercept and read their message.

This system does a great job at implementing confidentiality because intercepted messages from the president to the missile launcher cannot be read. However, this system does not do a good job at implementing integrity because it makes no attempts to verify who sent the message before acting on it. To better understand this vulnerability, imagine an attacker wants to fire the missiles. They know the ip address of the software and decide to start creating lots of connections sending random packets of 128 bits. Due to the seemingly random nature of decryption, it will on average take 2^16, or 65536 connections made before a decrypted message will start with the message 'Hi'. After setting up a few bots to make these requests automatically, the hacker wouldn't have to wait a full hour before the missiles would begin firing.

To solve this problem, we introduce a new cryptographic construct known as a message authentication code, or MAC. This category of algorithms uses a shared secret to 'sign' a message as having come from a particular source. This is implemented through one way functions that take in the message and a key, then deterministically produce an output. When sending a message, you will call this function with your message and your shared secret, which will produce the mac. Then you will send this mac over the wire to the receiver. When you receive a message you will split the message and the mac, then use your shared key to run the message through the mac function. If the macs match, you can be confident the message originated from your partner.

There are many types of attacks on macs. One very powerful example of this is the replay attack. In this attack, an eavesdropper first bides their time, collecting and listening to a few messages being sent and received. By observing the actions of you and your partner, the eavesdropper can glean some information about which messages had which effect. When the attacker wants to cause one of these effects again, they can simply send one of the mac'd messages to one of the parties originally involved. When the receiving party gets the message, they will calculate the mac and see that it matches. They will then assume the message to have originated from their partner and act on the given information.

Another difficulty in using macs comes when trying to simultaneously encrypt and mac. One naive approach is to mac the plaintext, then encrypt the entire message (plaintext and mac). This approach is poor because it forces us to first decrypt the message before we can tell if it is valid. By forcing us to decrypt, malicious actors can glean information about your key or how certain cipher blocks would be decrypted.

Another potential scheme is to run the mac on the plaintext, then encrypt just the plaintext and send the ciphertext and mac as the message. This has the same problem as the previous approach where endpoints can get you to decrypt arbitrary messages. This specific approach has led to well known attacks on popular standards such as SSH. In the SSH plaintext recovery attack, an attacker can manipulate the padding of a message to manipulate SSH into decrypting a byte into plaintext. This is exceptionally dangerous because all messages are simply a large series of single bytes.

The final approach involves encrypting the message, then appending a mac of the cipher text to the end of the message. Doing this form of mac ensures that you do not decrypt any messages unless they are properly verified first. Avoiding the malicious decryption avoids many kinds of unforeseen attacks on your key. The end result is that your key is safe and your system secure.