A Beginner's Guide to Encryption (and Encryptionizer®)

WHAT YOU WILL LEARN

- Basics of Encryption and Key Management.
- Problem we are trying to solve using the example of a hospital trying to protect the data on their Medical Devices.
- How NetLib® Encryptionizer® can solve the problems quickly, easily, transparently and without any programming.
- How Encryptionizer can help you come into compliance with such regulations as HIPAA or GDPR.

BASICS OF ENCRYPTION

(TIP: If you are already familiar with encryption and key management, you can skip to page 5 for some more specifics about Encryptionizer.)

Let's say we are in a hospital and instead of computers, there is a big whiteboard at the nurses' station with the room numbers, the patient's names and their medical information. There is also a person (call him the **Operator**) standing in front of the whiteboard whose job it is to write down the information that the doctor or nurse gives him.

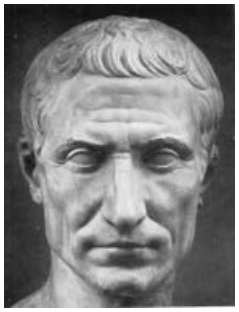
Let's look at our first patient, Harry in room 102. His doctor just diagnosed him with 875 (which is a hospital code for a specific diagnosis but not pertinent to this article). So he goes to the Operator and tells him to write on the board:

Room 102: diagnosed with 875

This way the doctors and nurses can walk by the board and get the information they need about a patient.

Even without HIPAA or GDPR, obviously this is sensitive information that you only want designated nurses and doctors to be able to read. But right now anyone walking past the board can read it: visitors, janitors, members of the public, etc.

As a result, the hospital decides to implement a type of "encryption" so that only the people with the proper authority can obtain the information on the board. They tell the Operator to start "encrypting" information before writing it on the board. Then when an authorized nurse or doctor asks for the information the Operator can "decrypt" it and read it back to them.



To make this happen, the hospital administrator has to provide the Operator with an encryption technique (or **Algorithm**) and an **Encryption Key (EK)**. In this case the Administrator decides on an Algorithm called a **Caesar Cipher** which is a centuries old form of encryption that anyone can use. This is where you "add" a number (known as the "Passphrase" to each letter. For example, if the letter is 'D' and you "add" 3 to it, you get the letter 'G'.

Now let's say the Administrator tells the Operator to use Passphrase = 3. The unencrypted message is referred to as **Plaintext**, and the encrypted message is referred to as **Ciphertext**.

PlainText	d	i	a	g	n	o	s	e	d		w	i	t	h		8	7	5
Ciphertext (+3)	g	l	d	j	q	r	v	h	g		z	l	w	k		1	0	8

If we needed to, we would cycle around to the start of the alphabet. For example if we had the letter **y** in the Plaintext we just cycle around to the beginning of the alphabet so it becomes **y**+3 = **b**. The same goes for numbers, for example when encrypting the number 8 it cycles around to become 8+3=1.

Now when a person comes by asking for information about a patient, the Operator has to first decide if they are authorized. If they are, the Operator has to read the information on the board, "decrypt" it using the reverse process with the same "Key" and then read it to the doctor or nurse.

> ***Sidebar***
>
> We are pretending that anyone who doesn't know the Key can't decrypt it. In reality, most people could decrypt a Caesar Cipher, given enough words to examine. Of course Julius Caesar did not have to worry about HIPAA or GDPR!

For computers the algorithm is much more complicated and the key is much longer than a single digit number. But the concept is the same:

- To Encrypt you need:
  - Algorithm (in this example the Caesar Cipher)
  - Passphrase (in this case the number 3)
  - Plaintext (in this case `diagnosed with 875`)
- To Decrypt you need:
  - The same Algorithm (Caesar Cipher)
  - The same Passphrase (number 3)
  - The Ciphertext (in this case `gldjqrvhg zlwk 108`)

KEY MANAGEMENT

Key Management is one of the most important parts of encryption and data security. How does the Operator know what Encryption Key to use? The Administrator must communicate the Encryption Key (EK, or in this case, EK=3) to the Operator in secret. The Administrator has decided on this solution:

- Administrator writes the EK (3) on a piece of paper (we'll pretend for the moment that email and telephone are not options).
- The Administrator hands the paper to an Intermediary to deliver to the Operator. (The Administrator does not leave his office.)
- The Intermediary drops the paper in a box next to the Operator's stand. (The Operator might be in a different part of the hospital at the moment or busy with something else.)
- The Operator takes the paper out of the box, reads and memorizes it, and then destroys the paper.

I am sure you can already see the potential problems. What is to prevent the Intermediary from reading the Encryption Key and then using it nefariously or telling other people what it is? What is to prevent anyone from picking up the paper from the box and reading it before the Operator gets to it?

Key Exchange Key (KEK)

This is where the idea of a **Key Exchange Key** (**KEK**) comes in as part of Key Management. The Administrator has to keep the Encryption Key secret at all times, and to pass the EK to the Operator without caring whether someone will see it or not. He does this by agreeing with the Operator, in advance, on a Key Exchange Key (KEK). (Let's forget for the moment how they agreed on a KEK. That is not really pertinent to the article.) Say they agreed on 5 as the KEK. To communicate the EK to the Operator, the Administrator uses the same process as above, but before writing the EK (3) on a piece of paper, he encrypts it with the agreed upon KEK (5) resulting in an encrypted EK value of 8. Now the only one who can decrypt the EK is the Operator since he is the only one who knows the agreed upon KEK. In fact, he does not even need to memorize and destroy the paper. He can leave it in the box to reference anytime he needs.

> ***Sidebar***
>
> The Caesar Cipher is a "Substitution Cipher", which is a variety of Algorithm called a **Symmetric Algorithm**. Meaning the Encryption Key is the same as the Decryption Key. Encryptionizer uses a far more secured Symmetric Algorithm

called **AES**. Instead of Passphrases being a single number (for example, 3), the Passphrase can be up to 32 characters long.

There are other complex encryption algorithms where the Encryption Key is different than the Decryption Key. These are called **Asymmetric Algorithms**. They have many uses but one of the main uses is as a KEK for protecting web site traffic. The mathematics are very complex and use something called paired Prime Numbers.

The reasons to choose a Symmetric Algorithm or an Asymmetric Algorithm have a lot to do with performance and use case. While Asymmetric algorithms have many uses, they are also much slower than Symmetric Algorithms due to the computations involved. (Working with large Prime Numbers takes a large amount of computing power.) So Symmetric Algorithms are used when processing huge amounts of data (e.g., megabytes, gigabytes and terabytes). Having the computer use an Asymmetric Algorithm on large amounts of data that are frequently accessed would make the system so slow as to be virtually unusable, even on modern day computers.

USING A COMPUTER INSTEAD OF A WHITEBOARD

Let's now use a computer instead of a whiteboard. In this situation, called a Medical Device. At this point we have not yet introduced Encryption. This is just the normal operation.

Instead of "diagnosed with 875" written on a whiteboard, it is stored in a file on the disk in the Medical Device. When the nurse or doctor wants to input the information this is what happens:

- Doctor starts the Application (in this document also referred to as **Program**) on the Medical Device. For example, they tap an Icon on the tablet (or double click a Shortcut on the Desktop) that runs a Program:
  `DEVICE.EXE`
- Doctor types "diagnosed with 875" and clicks Save.
- Application writes the information to the file where the information is stored (ignore the file extension for now):
  `MEDICAL-DATA.MDF`

To retrieve the information later, a doctor or nurse:

- Starts the Application on the Medical Device:
  `DEVICE.EXE`

- Enters the patient identification
- The Application reads the patient information from the file on disk:
  `MEDICAL-DATA.MDF`
- Application displays or prints the information.

See the potential problem here? Anyone using (or stealing) the Medical Device could read (or print, or email) the file, MEDICAL-DATA.MDF, and all the sensitive information in it; Periodic backups may be required for **Disaster Recovery**, also called **DR**, if, for example, the Medical Device breaks down and everything has to be transferred to a new Device. The data on these backups would also be vulnerable.

ENTER NETLIB® SECURITY'S ENCRYPTIONIZER®

This is where Encryptionizer comes in. Encryptionizer has been providing data encryption and key management for over twenty years. Encryptionizer functions as both the Intermediary and the Operator. The primary objectives of Encryptionizer are:

- Fast – Meaning the doctors and nurses do not see any slowdown in how fast the Medical Device performs.
- Ease of use – Meaning the Administrator can install and configure it easily.
- Transparent – Meaning that the hospital does not have to make any changes to the Medical Device application or how it is used by the doctors and nurses. For example, the doctors and nurses can use the Medical Device without having to know or enter any Encryption Keys.
- Secure – Of course.
- Help with compliance with HIPAA or GDPR.

Sidebar

For the sake of simplicity we will pretend that Encryptionizer also uses the Caesar Cipher. However, as mentioned previously, Encryptionizer uses the AES Algorithm. With our Caesar Cipher there are 9 possible Passphrases (we don't use zero since adding zero to the Passphrase does not change it). With AES, Passphrases are up to 32 characters or "bytes". This results in more than 1 followed by 77 zeros possible combinations! It would take all the computers currently in the world working together millions of years to try all possible combinations (called a Brute Force attack).

Since Brute Force attacks are impractical, the most common form of attack against AES is called a "Dictionary Attack". That is where the attacking computer has a "dictionary" of thousands or millions of the most commonly used words

> and phrases that people use when creating a Passphrase. It would not take a powerful computer very long to try all the possible combinations. This is why choosing a strong Passphrase is so important. (For example, not your cat's name!)

After the Administrator installs Encryptionizer on the Medical Device, he begins configuring it with various utilities provided along with Encryptionizer.

Configuration – First Step

Using one of the Encryptionizer configuration utilities, the Administrator is prompted for several pieces of information:

- The location of the data file(s) to be encrypted, in this case:
  `MEDICAL-DATA.MDF`
- The Encryption Key to use, in this case, Caesar Cipher with Passphrase 3.

The configuration utility will encrypt the file(s) one time with the designated Encryption Key. So in the file where it previously had this Plaintext:

`diagnosed with 875`

it now has this Ciphertext:

`gldjqrvhg zlwk 108`

Of course all the other data in the file is encrypted too.

So now, the data in the file is encrypted, but there is a slight problem: when any authorized person uses the Medical Device, all they will see is garbled information! Or, more likely the Medical Device will fail to function at all. So, now we need to instruct Encryptionizer to allow the Medical Device Application, and *only* the Medical Device Application to access the encrypted data.

Configuration – Second Step

Using another Encryptionizer configuration utility, the Administrator creates a special file (called a **Security Profile**) that contains this information:

- The Encryption Key that was used to encrypt the data file(s). In this case, the EK = 3.
- The Application that will be using the encrypted files. In this case:
  `DEVICE.EXE`
- Some additional details that are beyond the scope of this article.

The Security Profile is stored in the same directory with the same name as the Application but with a File Extension of `.SEC` ****** So, for example, if the Application is:

```
DEVICE.EXE
```

the Security Profile is stored in:

```
DEVICE.SEC
```

** Note: there are other more complex Key Management options available to Encryptionizer, but this is the simplest and most typical.

This Security Profile plays the same role as the box next to the Operator's station in which the Intermediary places the paper with the EK. The big difference – this Security Profile is itself also encrypted so that it cannot be read directly. (More about the Security Profile later.)

Using the Encryptionizer configuration utility to create the Security Profile is referred to as "**Securing** the Application". We have marked this application as **Trusted**, and have told Encryptionizer it is allowed to access the encrypted data. The next section explains how.

KEY MANAGEMENT SERVICE AND ENCRYPTION DRIVER

When you install Encryptionizer it also installs two special programs that run in the background at all times:

- **Key Management Service (KMS)** . The job of the KMS is to act as the Intermediary in the example we discussed earlier. It's job is to "watch" to see whenever the user starts a "Secured" or "Trusted" application. Much like a Virus Scanner "watches" to see whenever a program is started so it can examine it for viruses.
- **Encryption Driver (Driver)**. This takes the job of the Operator in the example we discussed earlier. Its job is to encrypt the information that the doctor or nurse enters before writing it to the file, and to decrypt the information when reading the file so the Application can display or print it.

On a very simplistic level this is how the process happens now that the Application is "Secured".

- Doctor or nurse starts the Medical Device Application:
  ```
  DEVICE.EXE
  ```
- The KMS "sees" that an Application has started and looks to see if this is a Secured Application. Which is to say, does the Security Profile file exist? In this case, does this file exist:
  ```
  DEVICE.SEC
  ```
- If it does, the KMS reads the Encryption Key (3) from the SEC file.

- The KMS program then communicates the Encryption Key 3 to the Driver.
- Whenever the Application reads from the file, the Driver decrypts it with the EK 3.
- Whenever the Application writes to the file, the Driver encrypts it with EK 3.

Notice that this all happens transparently to the Medical Device Application. The Application has no idea it is working with Encrypted files.

Security Profile

You might have observed one potential problem. If the EK (3) is stored in the Security Profile (SEC file), what is to stop anyone from simply opening and reading the Profile to see what the EK is? This is where the idea of the KEK or Master Key comes back in.

Let's continue with the example of using 5 as the KEK or Master Key. The different parts of Encryptionizer all have to agree on what the Master Key is. The Master Key (5) is programmed into Encryptionizer at manufacturing time. So now:

- When the Administrator uses the configuration utility to create the Security Profile, the configuration utility encrypts it with the Master Key (5) before saving it to the file.
- Then the Key Management Service reads the Security Profile (when the Secured Application starts) and decrypts it with the same Master Key (5) so it can see what the actual Encryption Key is (3).

> ***Sidebar***
>
> In reality, the behind-the-scenes process is much more complicated and there are other alternatives for Key Management, including **Encryptionizer Key Manager (EKM)** for centralized key management, where even the Administrator does not need to know the Encryption Key, so he cannot accidentally divulge it. Encryptionizer also uses various anti-hacking techniques. However, these are subjects for another White Paper.

SUMMARY

As you can see, we have accomplished our objectives stated earlier.

- We have encrypted the information stored in the files.
- We have not changed the Application.
- We have not changed the way doctors and nurses use the Application. For example, they don't have to enter an Encryption Key, nor do they even have to know it.
- Helped come into compliance with HIPAA and/or GDPR.