Encryption at rest:
Not the panacea to data protection

Prepared by:
Matthew Pettitt, Security Consultant

Table of contents

1. Introduction

Is encryption at rest really the answer?

Encryption is a big deal. News reports often talk about ransomware, which encrypts data then holds it to ransom until a payment is made. Payment is usually in Bitcoins, a currency that uses encryption algorithms to make it difficult to trace the exact ownership of wallets.

As for devices that are stolen, there has been much debate about whether the data on them is encrypted. When encrypted devices are seized by law enforcement, the debate changes to whether it should be possible to decrypt the data in order to see if they contain important information, such as terrorist plots or photos of child abuse victims.

However, with encryption in place data still gets leaked or stolen. The Republican Party in America had a huge database of 198 million registered voters, including addresses, phone numbers and email addresses, as well as voting preferences and links to social media accounts, which was found on a publicly-accessible server in June 2017 [1]. Wonga, the payday loan company, found that 245,000 records, including bank account details were accessed in April 2017 [2]. TalkTalk became known as a prime target to attackers in 2015, after suffering multiple intrusions [3], and was fined £400,000 by the Information Commissioner's Office (ICO) in October 2016 [4].

It's not clear as to whether the data was encrypted for any of these incidents, or, if it was encrypted whether the data owners knew about the breaches. In the Republican Party's case, the data was found on an Amazon S3 storage account. Amazon is known to offer a number of methods for encrypting data in S3 buckets. While it is clear that client-side encryption was not used, it would be reasonable to assume that the data was encrypted on the underlying drive and transparently decrypted for use. In other words, it was encrypted at rest, but still got stolen.

The ICO's "Action we've taken" section on its website [4] reveals more incidents that have occurred in the UK where the ICO felt they could bring a good case against an organisation. In one case, credit card details were taken from a site selling construction materials and tools and the card details were stored properly by a third party. Following all applicable rules, the data was transmitted over encrypted channels and only a tokenised form of the card number was stored. No one was found to have gained access to the card processor's data, which was fully encrypted. However, the front-end of the shop was less secure and an attacker had managed to edit the webpage that collected the card numbers and modified it so that card details were transmitted to an attacker-controlled server as well as the card processor. This neatly bypassed the problem of reading encrypted data on the page that was encrypted in transit (used HTTPS) and on the connection between the client and the card processor that was also encrypted. Even the modified page could well have been encrypted at rest, however, since the attacker used the expected method of editing, it would have been decrypted automatically for them to edit.

There are also dozens of cases where staff with legitimate access to databases looked up records belonging to people where they had no need to do so. This was likely either in order to sell data to third parties, to use for the purposes of blackmail, or simply to find out details that the owner hadn't told them directly. In most cases, this data is encrypted and the users only access the data legitimately for work purposes. Users with legitimate access to data in the work place can include administration staff in clinics, recruiters in employment agencies and anyone with authorised access to a work CRM package. Encryption at rest doesn't help with these cases as it's not a factor in these kind of attacks.

However, if we look at smaller scale attacks it becomes clear that encryption isn't being used where it should really be. Greater Manchester Police were fined £150,000 in May 2017 when three DVDs containing footage of interviews with victims of violent or sexual crime went missing in the post [6]. The DVDs were not encrypted, so the footage might have been viewed by anyone who found the missing disks. While the number of people affected was a lot smaller than the number of people affected by bulk data theft, the level of impact could well be higher, a point considered by the ICO when setting the level of the fine that was applied.

Data loss associated with a lack of encryption is a common pattern for relatively small volumes of data, but with a much larger impact on the affected individuals. In many cases, where a device was stolen, it isn't clear as to whether the thief knew what data was on the system and that the theft was likely targeting the device, rather than the data. In the six months prior to writing this whitepaper, thefts of this kind included data on 945 students from a school in Niskayuna [7], an unknown number of records on a laptop used by a health centre in Waco and nearly 60,000 records (including bank details) stored on a NAS device by Royal Sun Alliance in the UK [8].

So, if encryption at rest doesn't help in the larger cases, and isn't being used in the smaller ones, what are the reasons to use it?

Are there good reasons to use encryption at rest?

2. Encryption at rest

2.1 What counts as at rest?

Amazingly, the definition of at rest data isn't agreed on by all interested parties. Some consider data to be at rest when a device is switched off, arguing that this makes the encryption of content on always-on servers pointless. Others say that data is at rest when stored on a persistent storage medium, such as an SSD or hard drive, arguing that even when the device is turned on decryption would require a higher level of access to the system in order to obtain decryption keys from memory. Objectors to this definition tend to cite the transparency of decryption as a flaw, this is because if an attacker gains access to the legitimate method to access data, it won't help. However, this definition tends to go in hand with mobile devices that encrypt data when locked. They are referred to as devices which perform encryption at rest, even if they are rarely turned off.

Similarly, is data stored on disk within a database at rest or is it in use? It depends on the type of attack that it is being guarded against. If an attacker is able to extract the database files themselves, they probably can't read the data, but if they can execute queries, using the decryption keys held in memory by the database management system (DBMS), they'll be able to read it just as legitimate users do.

2.2 Active encryption or passive encryption?

Methods of encryption can vary, often falling into two basic forms. Passive encryption is where the end user doesn't need to take any action for data to be stored in an encrypted form. Alternatively, active encryption is where the end user has to specifically encrypt the data before storage.

The most common form of passive encryption is Full Disk Encryption (FDE). This is where, after entering an initial password, all data written to a device is encrypted. When the device is turned off, the data is unreadable unless the password is supplied again. This is obviously useful for devices which are easily stolen, such as laptops and mobile phones, but can appear less helpful for devices which usually have high levels of physical security, such as servers in a data centre.

However, there are cases where servers have been stolen. In 2003, two burglars who claimed to be from the company employed by the Australian Custom's agency to perform server maintenance, were shown to the server room on the third floor of a building in Sydney Airport [9]. They were left unattended for several hours, before walking out past a security desk with two servers. In this case, the servers were used for email services and didn't appear to contain sensitive data. However, the fact that they were taken from a secure room within an airport building shows that even supposedly secure devices can be stolen.

Drives can also be reused, even when securely destroyed. An NHS trust was fined £200,000 in 2013 when hard drives from trust PCs were found to be for sale on eBay [10]. This was despite the PCs having been provided to a third party data destruction company that had provided the trust with destruction certificates. One of these drives was found to contain the records of 2000 children and 900 adults, which had been insecurely deleted but could be recovered with freely available undelete tools. Had the drives been using FDE, even a basic format would have made the recovery of data more difficult, if not impossible.

Cloud storage systems often implement storage encryption too. Amazon's S3, as used by the RNC data, offers server-side encryption, allowing a choice of key management options. Microsoft's Azure platform offers similar, with BitLocker or dm-crypt encryption using a key vault. In each case, the selling point is that the system encrypts your data when you send it and decrypts it when you read it again. You don't have to know that the stored data is encrypted, but can still have a peace of mind, confident in the knowledge that it is.

Except, that isn't actually the case. Reading further into the Amazon S3 documentation, there are lines such as "as long as you authenticate your request and you have access permissions, there is no difference in the way you access encrypted or unencrypted objects." In other words, anyone who reads the data through methods you have built, such as your web application or mobile app, can read anything you've stored in that bucket, whether that specific object is encrypted or not.

It's also possible to store data on S3 and similar platforms with active encryption. This can be done by specifically encrypting files before upload, for example, to ensure that the cloud provider never sees the clear data. However, this has the same attack method, in the case of a web application or similar being used to access data and automatically decrypt it, however, an attacker using the normal access method isn't affected by the encryption.

When data takes the form of distinct files, rather than database records, it's also possible for individual files to be encrypted before storage. Options for this include encrypted zip or document files, where a password has to be provided in order for the file contents to be viewed. However, details of the file such as size and modification times aren't concealed. Another option is the use of dedicated encryption tools like VeraCrypt, where it is possible to create hidden volumes which can themselves contain files, with no way to prove the existence of a specific file without the volume password.

When file level encryption is used, one aim is to ensure it is safe to transfer the file over potentially insecure transport methods without the risk of the data being exposed, even if an attacker is able to get hold of a copy of the encrypted files. The effectiveness of this depends mainly on the choice of password, assuming that the underlying encryption algorithm has been chosen carefully and implemented correctly. To give one example, had the GMP DVDs included zipped versions of the video interview files, with strong passwords provided by another method, their loss would have been inconsequential. This is because the chances of an attacker being able to guess a random 12 character complex password are small enough to be disregarded.

There is no reason why systems cannot use both passive and active encryption for particularly sensitive data. Combining FDE for all files on a laptop with password protected documents that are considered particularly sensitive, or making use of a password safe application that requires unlocking on a device which is already encrypted are examples of this. This has the benefit of preserving security in some unexpected circumstances, such as sudden illness while using an unlocked device or if a laptop is stolen while unlocked.

2.3 Database encryption?

Sometimes, specific fields are encrypted within databases, where the data contained is considered particularly at risk. Care should be taken to ensure that the content of the data cannot be inferred from surrounding data and that the use of encryption is appropriate. For example, the Adobe breach in 2013 revealed that the password field in the data had been encrypted [11]. This was an unusual choice by Adobe, since it is rarely required for passwords to be decrypted. They are usually passed through a one way hashing function before storage, where the user input is also passed through the same hash function prior to comparison taking place to enable login functions.

The leaked data also included a plaintext “password hint” field which contained a user-provided string, intended to remind the user of the password they had chosen. In a lot of cases, users had applied hints of the form “password is <password>”, thereby revealing the contents of the encrypted field. As bad as this was, the choices made led to further compromises as the encrypted passwords used a block cipher with a fixed key. This meant that if the same password was chosen by multiple users the encrypted value was the same for each of them. Therefore, even if users had chosen a more secure password hint, but someone else had used a weak one for the same password, it was possible to determine the password for the user with the stronger hint. The use of a block cipher also leaked the possible lengths of passwords: for one eight byte block of ciphertext, the input could be between one and seven characters long. For two eight byte blocks, the input could be between eight and 15 characters long, and it was even possible to find which encrypted passwords contained exactly eight characters, by looking for a specific value of the second eight byte block.

If both password field and the password hint had been encrypted, it would still have been possible to identify the password lengths and to spot multiple users with the same password (or with the same hint). However, finding the actual passwords would have required determining the encryption key. By encrypting with a *per record* key (perhaps derived from combining a fixed key and the user id value in a secure way) only the lengths would have been revealed. However, the most robust method would have been to avoid encryption completely and use a password hashing algorithm with built-in salt. This would mean that all output was of a fixed length, preventing an attacker from determining the lengths of passwords. In addition duplicate passwords would have different values stored, based on the generated salt values. The only feasible method of attacking these values would have been by trial and error, testing each possible password for each user. The breach would still have included other data, of course, but this would have been less useful in enabling secondary breaches, where a user utilised the same password on Adobe and other sites. This kind of attack is still active four years later.

2.4 Key management

In order to perform encryption, there are two important requirements. Firstly, a suitable algorithm must be chosen and implemented carefully. Secondly, there must be a suitable key for the encryption, which must be appropriately strong, and stored carefully. As Kerckhoffs's principle states, “a cryptosystem should be secure even if everything about the system, except the key, is public knowledge.”

This is an area that Adobe appeared to have followed best practice with, as the key used to encrypt the passwords has never been released. This suggests that it was stored away from the data which was extracted. However, other breaches from other companies have shown that this practice is not universal.

Ashley Madison, an “affair seeker’s” website, was breached in 2015 with 36 million records relating to accounts on the site dumped onto the Internet, along with other data found on the company’s internal network, including emails and historical backup data [12]. Various investigations into the data, including some by the ICO, revealed that key management for encrypted data was not handled well. The VPN “shared secret” was stored on a shared Google Drive and therefore was accessible to anyone with access to an employee account. Also to anyone able to access a hard drive for a computer that had been configured to keep the contents of the shared drive offline, which appeared to be the company preferred setting. Passwords for internal systems and servers were emailed around the company, resulting in them being stored in plaintext in email archives. Encryption keys for sensitive data were stored in easily accessible locations on the network, meaning that the attackers were able to decrypt data which was thought to be protected by strong encryption.

In this instance, most of the encrypted data was embarrassing for the company, rather than causing direct problems for the site users (the unencrypted data, including email addresses and messages sent on the system, had more impact on the site users), meaning that this exposure was mostly side-lined in media coverage. It is possible for failures in key management to have more direct impact on end users.

One example of this is the 2017 breach of OneLogin [13], a company providing Single Sign-on (SSO) services to businesses wanting to use a range of cloud-based services like Office 365, G Suite and AWS. OneLogin used AWS to power their services, which in turn relies on the security of access keys. An attacker obtained a copy of the OneLogin AWS keys and used these to create a number of reconnaissance instances within the OneLogin cloud environment. This gathered an unknown amount of data from the system until IT staff were alerted to the unusual activity seven hours later. As the company relied on storing credentials in such a way that they could be decrypted without human interaction, they were unable to definitively state that the stored credentials could not have been accessed by the attacker. Especially as both the keys and the encrypted data were within the compromised environment.

There is also a risk of data loss if a key becomes unavailable. This can occur when a single password holder leaves the company or has an accident (sometimes known as the “bus factor”). This is defined as the number of people who would need to disappear to lose a specific piece of knowledge from a project or business, because of device failure or theft, or through simple forgetfulness.

Both the risks of compromised keys and forgotten keys can be addressed to some extent through a Key Management System (KMS). These aim to securely store cryptographic keys, ensuring that the use of them is audited, restricted to authorised people and, in some cases, only used for specific purposes. Examples of KMS include those provided by cloud platforms such as AWS, Azure as well as various on-site systems which interface with a Hardware Security Module HSM) to ensure that the encryption used for the keys themselves is very difficult to compromise.

2.5 Access control

It is important to note the difference between data which is encrypted and data to which access is restricted by operating system (OS) controls. There may be little visible difference to standard users, however, there is a significant difference when it comes to administrative users.

All commonly used OS allow some form of user account separation. It provides an area where files relating to a specific user can be stored away from both system files that are needed for the operation of the computer and files relating to other users. These areas are usually restricted by access controls which are enforced by the OS, meaning that the owner and administrative users on the system can access the contents. In Windows, this is controlled through the File and Folder Permissions dialog which allows either groups (such as “Administrators” or “Users”) or specific user accounts to be given different privileges relating to each file or folder. A group can also be given permission to read files but not to save changes to them. In UNIX based OS, including Linux and OS X/macOS. There are two options for setting file permissions. These are POSIX permissions and Access Control Lists (ACLs). POSIX permissions allow for a file to be read, written to or executed by some combination of the file owner, the group assigned to the file and “others” – a catch all that covers all users who don’t fall under the other two options. ACLs provide more detailed permissions and can, in some cases, simplify management of access to files.

The key detail from all of these systems is that administrators or root users can always override the settings. By default, the Administrators group in Windows has “full control” permission over all files and folders. As a result, any member can gain access to user files by providing the password to an administrator level account. The root user on a Linux system can ignore the permissions set on a file or folder and hence read or modify files at will. In both cases, they can also change the metadata for a file to hide this behaviour. Additionally, in most cases, an attacker’s ability to use another OS to access the data will not be restricted by the permissions. This is because it is less of an issue for servers, but a real risk for personal computers where it may be possible to boot from an external drive, or where a disk has been attached to another device for data extraction.

However, where a file or area has been encrypted, only users who know the appropriate password can access the data within the file. An administrator can still delete or replace the file with another file with the same name, but can only access the encrypted contents. The difference here is that while the OS permissions modify only the metadata relating to the file (which is enforced by the OS itself) encryption replaces the contents of the file. In order to regain the original file, both the encrypted contents and the password or key are required, and an administrator would only be able to get the encrypted contents, assuming reasonable key management precautions are taken. The administrator in this case is in the same position as any other attacker; the only option is to try and brute force the encryption key.

An example of this difference was demonstrated in March 2017 where the ICO ruled against an unnamed barrister, following the exposure of six documents which contained details of clients involved in proceedings in the Court of Protection and the Family Court [14]. The barrister had used their home computer to work on the documents and saved them within a password protected user account. However, the barrister's domestic partner also had an account on the computer, with administrator level access, and was therefore able to access all files on the system. As part of the process of updating the computer's OS the partner took a full backup of the documents without looking at them, and uploaded them to an online backup location. Unfortunately, as the documents themselves were not encrypted, instead just stored within a password protected account on the computer, some of the documents were found and indexed by a search engine. The documents remained online and accessible for three months, until a solicitor working for a local authority notified the barrister of the data being on the visible online.

In this case, the ICO ruled that the barrister should have proactively encrypted the specific files, even if they had thought that the password protection on the user account would have provided the same effect in normal usage. Additionally, the barrister was aware of the partner's administrative account, and the Bar Council had previously provided guidance about the use of shared computers, warning of this specific risk. As a result, even the use of FDE was considered to be unsuitable. In this case, FDE would not have prevented the partner from being able to access the data as most common methods of implementing it allow anyone with the password to access any files on the drive and as the computer administrator, the partner would reasonably be expected to know the drive password.

3. Full Disk Encryption

When it comes to implementing FDE, there are options for all commonly used OS, on servers, personal computers and on mobile devices. There are also some important limitations which apply to FDE.

In a 2014 paper, NCC Group's Cryptographic Services team provided four requirements for disk encryption. These are speed, random access, atomicity and space efficiency [15]. Disk encryption should have minimal impact on reading from the disk. This is both in terms of allowing data to be read off the drive at the hardware supported speed and not requiring excessive processor power which affects other operations while disk reads or writes are happening. It must allow for disk sectors to be updated without impacting disk sectors. It would be unacceptable to require all subsequent sectors to be changed following a single character change at a file near the start of the drive or if there was a need to decrypt all previous sectors in order to read a file that is stored near the end of the drive. It must minimise the risks of data loss in the event of power loss. The encrypted data should also take up the same amount of space as the original plaintext data, otherwise the effective space available on the drive would be reduced. And, a user with a nearly full drive should still be able to enable FDE.

These requirements mean that the encryption methods are restricted. Any method which provides a checksum, for example, will result in data taking up more space when encrypted, which is unacceptable. This makes the use of any fully-authenticated encryption method impossible. This led to the development of XEX-based tweaked-codebook mode with ciphertext stealing (XTS). This is an encryption mode designed specifically for disk encryption and attempts to balance the above requirements. It is not perfect and there may be some cases where an attacker is able to change the contents of the encrypted disk. However, it is generally considered a reasonable solution for protecting against reading data.

In general, while data on a disk which has been in the possession of an attacker and then subsequently recovered can be reasonably assumed to have remained unread, there is no method to verify that the data encrypted solely with FDE has not been modified. This may seem counterintuitive but is a side effect of the unauthenticated nature of the encryption. Changing a single bit of the ciphertext results in the decryption produces a different result for the affected 16 byte block, when using XTS mode, just as changing a single bit of plaintext should result in a much larger change to the ciphertext. Due to an attacker not knowing the specific location of files on the disk it would be very unlikely for this to result in any specific output, although it could allow for the corruption of data. FDE is not an alternative to a robust backup strategy.

A more realistic attack vector from a returned encrypted drive would be replacing the contents with targeted malware or even modifying the hardware to act in a malicious way. Any device or drive which has been in possession of a potentially malicious party should be treated with care and never connected directly to production systems without first being inspected for tampering. If there is any doubt about the device, don't use it.

3.1 BitLocker

Microsoft's BitLocker disk encryption software, built into Windows versions from Vista onwards, uses AES (in XTS mode for Windows 10 and in CBC mode for earlier versions). There is also a composite key built from some combination of data provided from a Trusted Platform Module (TPM) in a computer, a PIN or password entered by the user and a value stored on a removable device. The specific combination that is used can drastically change the security of the system given different threats:

- TPM only: Data encrypted using just a TPM is accessible should the whole device be taken, but protected in the event of just the hard drive in question being taken. The system doesn't require any interaction in order to boot. Therefore, it can be a suitable choice for servers, where drives can often be removed without opening the case and where unattended reboots may be required.
- TPM+PIN: Data can only be accessed on the original device and only when a PIN is provided on boot (or when the storage device is connected for removable drives). There are also options for allowing a domain joined computer to boot without a PIN when connected to the network, but to that require a PIN to be entered when disconnected, giving an improvement over the TPM only option for servers.
- Physical key: The device can only be booted or accessed when a specific USB device, containing a key file is plugged into the computer or when a cryptographic smart card is inserted into the device. This allows the use of BitLocker on computers without a TPM and can provide security when travelling through dangerous areas. This is done by sending the device and key separately, although extra effort would be needed for attackers to obtain both requirements and to decrypt the data. If the key is kept with the encrypted device or worse still, left plugged into the device, it doesn't offer any benefits over TPM only usage.
- TPM+key: Where a computer has a TPM, it is also possible to use both a value provided by that and a value from a physical key. This has the same benefits as the key alone but also requires that the original device not be tampered with, thus increasing the effort required for attackers. Again, this can be rendered useless if the physical key is stored with the device or left plugged into the device.
- TPM+PIN+key: Combining all of the methods means that an attacker would need the original device, a PIN known to the legitimate owner and a physical key device in order to access the data. As a result, this method of operation is generally unsuitable for any device requiring remote reboot but ideal for laptops containing potentially sensitive data.
- Recovery password: It is also possible to configure BitLocker to support a recovery password if the physical key is lost or if the system is legitimately modified in such a way that the TPM is reset by a detected change. Depending on the device, this can be something as simple as removing the battery. This is a 48 digit value which should be stored securely away from the device it corresponds to. The recovery password bypasses all of the protections above, so should be used infrequently, and, ideally, only in a secure physical location.

In order to allow for the user controlled parts of the key (mainly the PIN) to be changed without requiring all of the data to be re-encrypted, BitLocker uses a surrogate key system. The majority of data on the drive is encrypted with a Full Volume Encryption Key (FVEK), which is generated using random data and has no direct relation to any of the key elements above. This key is itself encrypted by a Volume Master Key (VMK), with at least two encrypted versions of this being stored. One is normally encrypted with the derived key from the normal boot process requirements (the combination of TPM, PIN and physical key), and the other is encrypted with a key derived from the recovery password.

This means that changing the recovery password without changing the normal boot process is possible, and vice versa. It also allows for a very quick secure format, where only the encrypted version of the FVEK needs to be removed, as all data on the drive would then be unreadable. It also means that there are specific circumstances where changing a password or PIN which has been compromised may not prevent access to the data.

If a password has been compromised and the attacker was able to either copy the stored version of the VMK or obtain the decrypted FVEK from memory after booting the device, they would still be able to decrypt the data. This is even if the VMK has changed. They wouldn't be able to decrypt the new version of the VMK, however, they wouldn't need to as the rest of the data on the drive would still be encrypted using the existing FVEK. Windows has various protections in place to make this difficult, but it may still be possible for skilled attackers. The only way to prevent this is to re-encrypt the entire drive as disabling and re-enabling BitLocker isn't sufficient.

It is important to note that regardless of the type of key used, BitLocker does not prevent access to files when the computer is unlocked. Once Windows has booted, the files are transparently decrypted and re-encrypted when required, and can be copied to other devices or sent over a network connection. The received files won't be encrypted unless another layer of encryption has been applied, such as the files being wrapped in a password protected archive file.

3.2 VeraCrypt

An alternative to BitLocker is VeraCrypt, a maintained fork of the TrueCrypt drive encryption application. While BitLocker is Windows specific, VeraCrypt supports Windows, OS X and Linux for data volumes, although it can only provide full system drive encryption for Windows. It works in a slightly different way to BitLocker in that in addition to encrypting data, in some cases, it attempts to offer plausible deniability of the existence of encrypted data..

As a result, it does not store any details of the encryption method, the key generation method or the key size on the unencrypted part of the drive. In theory, a VeraCrypt volume is indistinguishable from random data, although in practice, having VeraCrypt installed on a device or available on an attached removable storage device strongly suggests that any large blocks of random data might be encrypted volumes. When opening an encrypted volume, the user provides a password and the software then attempts to decrypt a specific portion of the encrypted data.

In order to do this it has to try all supported key derivation functions (which expand the password provided into a fixed length value, suitable for use as a key), all supported encryption algorithms (it supports multiple algorithms, but always uses the XTS cipher mode) and all supported key sizes until it finds a combination which works. Or if it has tried all combinations without success. Specifically, it looks for a combination that results in decrypted data starting with the ASCII string “VERA” and where another section decrypts to a value with a checksum matching the value found at byte 8 of the decrypted data. The chances of these values matching and providing the wrong details are tiny, so if they are correct, the assumption is made that the decryption parameters used are correct. The rest of the decrypted data at this point includes a VMK and this is used to decrypt the rest of the volume, in the same way as a FVEK in BitLocker.

As with BitLocker, changing the user provided password simply changes the encryption on the volume key. Changing the volume key itself involves recreation of the volume from scratch.

VeraCrypt does not make use of a TPM if one is installed. The developers claim that they provide a false sense of security, especially since continued use of a device where the TPM has detected tampering is risky. In particular, using a BitLocker recovery password to regain access to a drive encrypted with a key derived from a TPM may result in an attacker being able to obtain the recovery password and subsequently, the contents of the encrypted drive. The logic here is that an attacker with physical access could implant a keystroke logger into the device, relying on the TPM being reset by the tampering and refusing to provide part of the key. A naïve user may then attempt to regain access to their data using the recovery password, which is collected by the keylogger, and then used by the attacker to decrypt a clone of the encrypted drive.

While this may be an extreme example, it is becoming more common for companies to warn against taking computers to potentially hostile regions as a result of this type of attack. For example, the University of Rhode Island advises staff to if possible, avoid traveling to China or the Russian Federation, and advises using a throw away temporary system if a computer is needed for work in those areas. Similar advice exists for travellers from other countries to the USA, UK and other nations which co-operate on security matters.

3.3 dm-crypt

For Linux-based systems, dm-crypt can be used to provide FDE. This is implemented as a generic block device encryption layer, meaning that it does not attempt to handle key management or key stretching. It simply makes use of a specified encryption algorithm, with a specified key, and encrypts and decrypts all data read from or written to the device. This is logically simple but does present some drawbacks as whatever key is provided is used, the only method to change the password is to re-encrypt the full volume. Users are also required to provide all the details of the encryption algorithm whenever the volume is unlocked, as these are not stored in any way. This also means that there is no method to detect when invalid parameters or an incorrect password have been provided. The software will simply provide whatever data is output from using the supplied parameters in order to decrypt. One benefit of this is that in the event of data corruption on the drive, only the directly affected sectors are impacted. dm-crypt supports any encryption algorithm which is implemented in the Linux Crypto API, although the use of hardware accelerated algorithms is recommended.

In order to improve the security and increase the usability, dm-crypt is often used in conjunction with Linux Unified Key Setup (LUKS), which adds a metadata header and a number of key-slots. The metadata header holds the specifications of the encryption algorithm used for the associated volume, while the key-slots hold an encrypted version of the device encryption key. Each key-slot contains a copy of the device encryption key which is encrypted with a slot specific password and salt. The device encryption key is equivalent to the FVEK in BitLocker, and LUKS supports the simultaneous use of up to eight distinct passwords, any one of which can be used to unlock the volume. This can allow for FDE without a shared password, although the password provided to decrypt the volume does not need to be the same as the user account password.

The use of a header makes the existence of encrypted data more obvious, but means that it is possible to verify the supplied password. A checksum for the device encryption key is also stored in the header. Due to the use of key stretching, it is also reasonable to use a shorter passphrase when using LUKS with dm-crypt than would be considered safe for plain dm-crypt. This is even when taking into account the ability to verify the passphrase. The header is also a potential weak point for the system. If it is damaged, the volume may become unreadable, as there is no method to regenerate the salt values it holds. Storing a backup of the header data can help to guard against this.

3.4 FileVault

Apple has an equivalent to BitLocker in FileVault, which was introduced in OS X 10.3, in 2003, but substantially redesigned and improved with the release of OS X 10.7 in 2011. Like BitLocker, this uses AES in XTS mode. As with BitLocker and VeraCrypt, the bulk of the drive is encrypted with a key which is itself encrypted with a key derived from the user password. Unlike BitLocker and VeraCrypt, but similarly to LUKS, FileVault encrypts the disk key with each system user's password. This potentially reveals the existence of user accounts to an attacker and leaves the encryption security as equivalent to the weakest user password. Where the other applications allow encryption of the system drive, this requires a password before the majority of the OS has loaded, and user accounts, which may themselves implement encryption, are accessed with a second password.

FileVault also does not support the use of TPMs, although this appears to be more from hardware limitations than any other reason. Apple laptops have not included a TPM since 2006, although the latest models, at time of writing, now include a Secure Enclave Processor (SEP), which offers some of the same functions, along with some other abilities which a TPM does not offer. Since the same kind of setup is used on iPhones, it seems reasonable to extrapolate that Apple may introduce a similar encryption method to their computers at some point.

3.5 iOS encryption

Apple also introduced the SEP to their phone lines in 2013 and enabled FDE using a user supplied password or PIN in iOS 8. This works by mixing the user supplied key with a hardware key, in the SEP. The hardware key is never exposed outside the SEP, meaning that any attempts to crack the encryption either needs to take place on the device or attempts to guess both the user password and the hardware key. Since the hardware key is a 256-bit device specific secret, which Apple claim never to store and to be unable to access, the chances of this being guessed are remote. The output from the mixing is then used to encrypt the data stored on the device.

This is a FDE scheme, meaning that once the password is provided, any application running on the device potentially has access to any of the data stored – the access is controlled through application sandboxing, rather than by an inability to read clear data.

3.6 Android encryption

Google offer support for FDE, based on dm-crypt, on Android devices running Android 5.0 upwards. They also provide a more granular file-based encryption system, starting with Android 7.0, to overcome some of the shortcomings of full device encryption in a mobile device. This allows for the device to have a file storage area which can be accessed while the device is fully locked from the user's point of view (the Device Encrypted or DE area), and then a separate area which can only be accessed by the active user, once they have provided a password (the Credential Encrypted or CE area).

Examples for using this include storing a limited address book in the DE area. This means that incoming calls can still display caller details, even before the phone has been unlocked after being turned off. This separation of encryption also means that data for each user can be encrypted with a distinct key, reducing the risk of one user being able to compromise data belonging to another.

3.7 Chromebooks

While Chromebooks are designed to take advantage of always-on connections to the cloud, minimising the amount of data stored locally, they also implement per-user encryption using eCryptfs for all locally stored data. This combines data from the device's TPM, part of the user's Google account password, and salt stored on the unencrypted part of the drive, meaning that each user's data is unreadable without their password, even to device administrators. This is technically file-system encryption, rather than FDE.

The main drawback of this is that the data is unrecoverable, in the event that the password is lost. Google considers this to be an acceptable compromise as Chromebooks are designed to hold mostly transient data, with the master copies and longer term storage handled by the cloud services (G Suite, formerly Google Apps, for enterprise use, and the individual Google Docs and Google Drive applications for individuals).

It should be noted that while this encryption requires the user password to decrypt data, it is not the same as Apple's FileVault. In FileVault, any of the user passwords are sufficient to provide access to the contents of the drive, with the OS providing access controls preventing users from viewing each other's files. In a Chromebook, only the specific user's password will decrypt their area, with other user's areas remaining encrypted. Only data for the currently active user as well as system files required by all users, is accessible.

4. File level encryption

In some cases, there is a requirement for specific files to be encrypted on a system, even when FDE has been used. This can be as an extra precaution against access while the device is turned on, to prevent access to specific file contents by rogue administrators, or simply because the data is considered sufficiently sensitive to require extra protection.

Some software directly supports encryption such as, Microsoft Office, and uses AES and a key derived from the provided password. In other cases, it is necessary to wrap a file in an encryption envelope, such as a ZIP file, where most modern tools support the use of AES encryption. Where an envelope approach is used, care should be taken that the contained file is not left accessible when in use. Some applications will make a copy of files which they have open in temporary or cache directories, however, this can provide an unintentional method for attackers to obtain their contents.

There is also a hybrid option in file-system encryption whereby encryption is provided by the underlying file system. This can either use keys provided by the user, or that are derived by the operating system from some user specific data, such as their account password. Microsoft's Encrypting File System (EFS, built into NTFS) is one example of this, which allows users to encrypt files or folders using their Windows user account. This is transparent to the user, following the initial encryption, as long as they are signed into their own account. However, any other users of the system are not able to open the files unless given permission by the original owner (they will just be shown an "access is denied" message) and even administrators are not able to override this. Microsoft do provide a method to enable data recovery and this can be configured prior to files being encrypted. It also works similarly to the recovery password for BitLocker, but even this is unable to read files encrypted without the prior configuration steps. One Linux equivalent to this is eCryptfs, as used by Chromebooks, but the ext4 file system also supports a similar transparent encryption method on a per directory basis.

One key difference between file level encryption and file system level encryption is whether the encryption is preserved when a file is copied to another device. A file which has been encrypted at file level remains encrypted when copied or moved to another device, since the encryption is a property of the file itself. A file which has been encrypted with file system encryption, while encrypted on the original device, is copied as read by the user – and when read by a user with the appropriate password, it is treated as an unencrypted file. When read by a user without the password, the OS or file system may refuse to allow a copy, or the encrypted bytes may be copied, depending on the specific configuration. Therefore, file system encryption is not suitable when a file needs to be transferable while remaining encrypted but it does address the risk of other users being able to view file contents present in FDE.

5. Malware

Media coverage of cyber crime commonly mentions ransomware, where malicious software encrypts files and demands payment to decrypt them. Probably the most well-known example of this in recent years is WannaCry, which hit systems worldwide in May 2017 [16]. There are also many others, dating back as far as 1989.

Ransomware tends to use file level encryption, instead of a password which the user knows, and a key protected by asymmetric encryption tends to be used. When the user pays the ransomware operators, they are, in theory, given the decryption key.

One fairly common misconception is that ransomware cannot affect encrypted files. However, this is not true and just as it is possible to use both FDE and per file encryption to protect a single file, it is possible for a single file to be encrypted multiple times. Some ransomware variants do only affected specific file types, which may have helped this rumour spread. Newer ransomware variants tend to just encrypt all files found in the user directories and since they typically run as either user or system accounts, the files which they encrypt are transparently stored. They are stored using any lower level encryption that is in place, such as FDE or file system encryption.

In theory, it would be possible for a full disk or file system level ransomware application to be written. This would be even more damaging than the existing file level versions, since the system could be rendered completely unusable, rather than just preventing access to files. As with existing variants though, the only reliable protection against such an attack would be regular backups.

6. When to use encryption at rest?

One common comparison which is often made when looking at whether to use encryption for data at rest, is that of insurance. When buying insurance, it is to help deal with an unwanted occurrence, such as a car crash, illness or an accident on the premises for example. It is, therefore, intended to be a cost without benefits as it is rare people want the insured occurrence to happen. However, if the unwanted does happen having insurance means that the impact is reduced. Similarly, if devices are never stolen or lost, or files are never exposed to the outside world, the costs of encryption at rest (which comes in the form of performance losses and a requirement to provide passwords to see data) may be considered to be without benefits. The moment a device is lost though, the use of suitable encryption can be the difference between a minor inconvenience, in the loss of a device, and a major incident, in the loss of sensitive data.

The UK ICO takes the use of encryption into account when deciding whether to impose fines and when deciding on the level of fine to impose. In particular, the seventh data protection principle in the Data Protection Act states “appropriate technical and organisational measures shall be taken against unauthorised or unlawful processing of personal data and against accidental loss or destruction of, or damage to, personal data”. A similar requirement is also present in the General Data Protection Regulation (GDPR). Being able to demonstrate that data was suitably encrypted can therefore improve an organisations compliance with regulations.

Clearly a one size fits all solution is not possible when it comes to encryption at rest. However, it is still possible to provide some general guidance.

- Devices used by one individual: Full disk/device encryption is likely to be suitable to protect data in the event of the device being lost or stolen. File level encryption should be used for data being transferred off the device.
- Devices used by multiple individuals: FDE will protect data if the device is lost or stolen, but all users of the device will be able to access files, given sufficient access privileges. In particular, users with administrative level access will be able to see the contents of files even when stored within password protected user areas. Therefore, file level encryption or encrypted volumes with passwords known only to the intended users, or file system level encryption linked to user accounts, should be used. Even in this case, the possibility of the files or volumes being subjected to a brute force attack should be accounted for with the use of suitably strong passwords.
- Servers in physically secure locations: The main risks to data on networked servers tends to come from remote attackers. Therefore, FDE is unlikely to provide benefits during the normal usage of the server, but could make the safe destruction of data at the end of device life simpler. Sensitive data which needs to be accessible in clear form (for example within a database to allow searching) may be difficult to encrypt without impacting functionality or performance to an unacceptable level. Technical controls other than encryption are likely to be more suitable with careful consideration given to account segregation, access controls and general system security being the primary preventative aspects, and proactive monitoring enabling reactive responses to attacks.

- High performance devices: There are some circumstances where the overhead required to encrypt data may cause problems, such as in real-time systems. These systems may be in physically secure environments (for example power station control systems), limiting the risks of data loss or it may be that the main risks for these systems are not related to the loss of data. Instead the main risk may be the introduction of unexpected data. Options in these examples include running with a clear operating data set, encryption of data which is not currently being used (possibly on a separate system) and implementation of other controls, such as enforced air gaps.
- Sensitive files to be stored on third party servers: Even if the servers implement encryption, it is generally sensible to encrypt files before uploading them. This is unless it is acceptable for the server administrators to have access to the data contained in the files. In this instance it is equivalent to the “devices used by multiple individuals” requirements, but with the additional difficulty of knowing whether the server implements encryption at another level. From the point of view of a user, there may be no way to determine whether FDE per user encryption, or per file encryption is being used, even if there is a requirement to provide a password for each accessed file.
- Databases: Encrypting the entire database is roughly equivalent to applying FDE; it protects against offline attacks, but does not affect attackers that are able to gain equivalent access to the legitimate users. Column level encryption, where specific data within each record is encrypted using a column specific key, can allow for more fine grained control of data. Standard users can still search and privileged users with access to the decryption keys can read the encrypted data. An attacker able to gain standard user level access may be able to extract the encrypted data, but only in encrypted form, limiting the impact. Application level encryption, where only encrypted data is written to the database, is also possible, although this tends to somewhat limit the use of the database functions. In particular, it is generally not possible to search the encrypted data using the database search functions. This means that searching must instead be implemented on the application level, and performed after reading data from the database. This usually has a significant performance impact.
- Passwords and PINs: When passwords are used to provide access to a service or application (and so are provided by end users) there are very few circumstances where encryption is the right solution. Instead, use a cryptographically strong password hashing algorithm, such as PBKDF2, bcrypt or Argon2 (with a per-record salt) and compare the hashed version of the password supplied by the user to the hash stored when the password was created. Where there is a requirement to be able to obtain the original password, such as in a password vault application, or when interfaces to legacy applications only allow password based authentication, encryption may be the only option. In these cases, assuming a reasonable choice of algorithm, the secrecy of the key becomes even more important than is normally the case and the compromise of the key can result in the compromise of other systems or passwords.

In all cases, minimising the amount of sensitive data being stored on any system is a good initial position, especially with data protection laws such as GDPR stating that personal data should be “limited to what is necessary for the purposes for which they are processed”. The same sentiment can be applied to other data through relatively simple precautions, such as deleting commercially sensitive documents from laptops once the requirement for them has passed. Where there is a need to preserve data generated on portable devices, moving it to more secure storage environments, such as a physically secured server, making use of strong encryption can be a good option.

It is also important to regularly review how data is stored in order to ensure that the methods used are still fit for purpose. This is particularly the case for archived material which remains sensitive, where in some cases it may be appropriate to re-encrypt data with a different algorithm, and securely delete the existing copies. For example, the Data Encryption Standard (DES) was approved as a standard in 1977 and remained approved until 2005. This is even though research in 1999 showed that it was possible to break a DES key in less than 24 hours. As a result, it is likely that some documents which were originally stored through having been encrypted with DES were still sensitive when the Advanced Encryption Standard (AES) became the approved algorithm, hence they needed to be re-encrypted to remain secure. It is important to note that re-encryption in this way does not help should the encrypted data have already been compromised. Holders of the previous version can still attack it, so this generally only applies where there is a significant level of confidence that the original encrypted data remained secure.

7. Backups

When using encryption, the question of how to perform backups can be difficult. Should the drives or tapes used for backups be encrypted themselves? Should files on the backup devices be encrypted? What about online backups? This is complicated further when the need to store decryption keys or recovery passwords arises.

As with choosing a method of encryption, there is no one size fits all approach, instead it depends on the risks which are deemed acceptable or unacceptable to the business or individual. If a business would be unable to function in the event that data was lost, using unencrypted backups which are stored in a physically secured location may be acceptable. In this case, the loss of encryption keys or passwords would mean that the data would remain accessible. However, the loss of a backup device would expose the data. Alternatively, applying extra layers of encryption to backups can mean that even in the event of a backup device being stolen by someone with legitimate access to the data, they would not be able to make use of the stolen device.

Whatever method is chosen should be documented and tested. The failure of a backup can be catastrophic to a company, even when an older backup can be found. If both current and historic backup data is unavailable, due to destruction of keys, the only option would be to rebuild the data from scratch – even data recovery companies would be unable to help in most cases.

The use of physical copies of recovery keys should not be ignored as the day-to-day passwords can be changed when using FDE and without affecting the recovery keys. The recovery passwords can be very long, mitigating the risk from brute force attacks, and only used very infrequently, balancing the inconvenience of a long password with the need to use it.

It is also important to remember that data copied from an encrypted device to another encrypted device may be transferred in plaintext, which may be a potential weak point. Care should be taken to ensure that appropriate encryption is used for data in transit, such as using SFTP rather than FTP to transfer files over a network.

As with the original data, the encryption method used for backups should be reviewed on a regular basis and there should be a clearly defined process for secure destruction of backups once no longer required, whether encryption was used or not.

8. Conclusion

The encryption of data at rest is a very complex subject and it can potentially carry very high risks if performed incorrectly. Overuse of encryption without sufficient key management can result in data which is inaccessible to the creator. At the same time poor key management, or a failure to use suitable encryption, can result in sensitive data being exposed to other parties and hefty fines from regulators.

The specific types of encryption to use for any given data can also be difficult to choose. This means that in some cases a mixture of different types of encryption can be the best choice, in order to protect against as wide a range of threats as is possible. However, for different threat models, the use of any encryption may increase risks.

Encryption at rest is not a panacea to data protection. As mentioned many times in this paper, the utility of data means that at some point it needs to be decrypted. As for online services, this often means a permanent state of decryption, as the data is seldom, if ever, actually at rest to be encrypted as such. This is important to note when considering new regulations such as GDPR. Misconceptions can (and do) arise whereby it is believed that encrypting everything brings in easy compliance. This paper has hopefully provided rationale as to why this is not the case and more importantly, what the barriers or challenges are in the realm of encryption at rest.

9. References

[1] http://money.cnn.com/2017/06/19/technology/voter-data-leaked-online-gop/index.html

[2] http://www.bbc.co.uk/news/business-39544762

[3] http://uk.businessinsider.com/talktalk-hacked-credit-card-details-users-2015-10

[4] http://www.bbc.co.uk/news/business-37565367

[5] https://ico.org.uk/action-weve-taken/

[6] http://www.bbc.co.uk/news/uk-england-manchester-39805805

[7] http://news10.com/2017/05/24/niskayuna-school-lap-top-stolen-945-students-personal-information-on-device/

[8] https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2017/01/150-000-fine-for-insurance-company-that-failed-to-keep-customers-information-safe/

[9] https://www.theregister.co.uk/2003/09/05/the_case_of_the_two/

[10] http://www.bbc.co.uk/news/technology-23286231

[11] http://www.bbc.co.uk/news/technology-24740873

[12] https://arstechnica.co.uk/tech-policy/2016/12/hacked-cheating-site-ashley-madison-will-pay-1-6-million-to-ftc-for-breach/

https://www.wired.com/2015/08/happened-hackers-posted-stolen-ashley-madison-data/

[13] http://www.bbc.co.uk/news/technology-40118699

[14] https://ico.org.uk/about-the-ico/news-and-events/news-and-blogs/2017/03/fine-for-lawyer-who-stored-client-files-on-home-computer/

https://www.theregister.co.uk/2017/03/16/barrister_fined_over_data_breach/

[15] https://cryptoservices.github.io/fde/2014/12/08/code-execution-in-spite-of-bitlocker.html

[16] https://www.theguardian.com/technology/2017/may/12/global-cyber-attack-ransomware-nsa-uk-nhs