Last updated: 26 January 2023
Compiled by the NAACCR Data Security & Confidentiality Workgroup

Data Encryption and Hashing Primer

Editors:

David Chesnut, Information Management Services, Inc.

Jean-Michel Billette, Canadian Cancer Registry – Statistics Canada

Castine Clerkin, North American Association of Central Cancer Registries

Steven Friedman, National Cancer Institute, National Institutes of Health

Susan Gershman, Massachusetts Cancer Registry

Selina Khatun, Nunavut Cancer Registry

Bozena M. Morawski, Cancer Data Registry of Idaho

Lauren Maniscalco, Louisiana Tumor Registry

Robert McLaughlin, Cancer Registry of Greater California

Recinda Sherman, North American Association of Central Cancer Registries

Qianru Wu, Nebraska Cancer Registry

Heather Zimmerman, Montana Central Tumor Registry

Publication date [1] : January 25, 2023

[1] These guidelines should be reviewed and updated every 12 months.

Contents

Overview

This data encryption and hashing primer provides a high-level overview on the topics of data encryption and data hashing. It provides an overview of different types of encryption and what they are and can be used for. It also provides an overview of data hashing, describing what hashing is and what it can be used for. It also outlines concepts that should be understood when hashing is used to secure data during a data merge with outside parties.

This primer does not attempt to provide a full overview of all encryption or hashing facts, as there have been many volumes written on those subjects. This primer is only meant to provide a high-level overview of those topics.

Definitions

Plaintext:

Any data that can be viewed/used/read (by a human or otherwise) without data decryption. Plaintext is the input to an encryption process, or the result of a decryption process.

Ciphertext:

Data that require a key or decryption to be viewed/used/read, resulting from performing an encryption process on plaintext (data).

Data at Rest:

Data at Rest typically refers to data that are stored on some type of media. This could be a hard drive, USB stick, or even cloud storage. It can also refer to data stored within a database.

Data in Use:

Data in Use (sometimes called Active Data) refers to data that are currently being processed in some way by a computer system.

Data in Transit:

Data in Transit always refers to data that are actively being sent across a network. The network can be a local network or the Internet.

Hash Salt:

A unique random text string that is added to the data being hashed to add randomness. It is sometimes referred to as a key.

Key:

A string of characters used in an encryption algorithm to convert Plaintext to Cyphertext and Cyphertext to Plaintext. Can also refer to the Hash Salt in a hashing algorithm.

Rainbow table:

The list of all field values and their corresponding hash and can be used as a reverse lookup to go from hashed value to original value.

Encryption

Encryption is the process of converting readable text (plaintext) via a mathematical process into ciphertext using a key. The process is reversible, meaning that when the correct key is used to decrypt the ciphertext, the ciphertext will be converted back to the original plaintext.

There are many different types of encryption algorithms and processes used to encrypt data. While some encryption algorithms and processes are more secure, faster, or easier to implement, all are meant to perform a similar function: to protect the confidentiality and integrity of data.

However, to ensure that encryption can protect data confidentiality, integrity and availability, an effective key management strategy is a must. The best encryption algorithm in the world cannot protect confidentiality if keys are compromised or not managed appropriately. The same can be said for data integrity. If keys are compromised data may be modified and re-encrypted without the receiver's knowledge, compromising its integrity. Also, availability may be compromised if keys are lost. Once data are encrypted the key is required to recover it. Without the key the data are no longer available.

Basics

One of the essential properties of encryption is that the process is reversible. In other words, what you put in is what you get out (especially important as encryption is used to protect the confidentiality of data that will later be used in an unencrypted manner):

- Plain text + Encryption algorithm + Key = Ciphertext
- Ciphertext + Decryption algorithm + Key = Plain text

It should be noted that for encryption to function properly, plaintext that comes out of the decryption process must be an EXACT copy of the plaintext what was put in. Encryption can therefore also be used to protect data integrity along with its confidentiality.

A second essential property of encryption algorithms is that if the key or the ciphertext is modified the returned plaintext from the decryption algorithm does not give away any part of the real plaintext. A corollary to this is if the ciphertext is modified in any way the resultant plaintext is unusable. Please note that most decryption processes will detect the ciphertext modification and alert the user that the ciphertext has been modified and will refuse to decrypt it. This can happen because of incidental internet transmission errors or because someone has purposely tampered with the file or the internet transmission.

Uses for encryption

There are many uses for encryption. The following are examples of the common day-to-day uses of encryption:

- Secure communications or data in transit
    - Websites, e.g., https://somedomain.com
    - File sharing applications, e.g., Box, OneDrive
    - Secure emails
- Secure data at rest

- Full disk encryption
- Encrypted USB stick
- File encryption

Algorithms and Methods

An overview of encryption would not be complete without the definition of the types and classes of encryption keys/algorithms. This is not an exhaustive list, but does provide the most popular and secure algorithms at the time of this publication:

Symmetric Algorithms

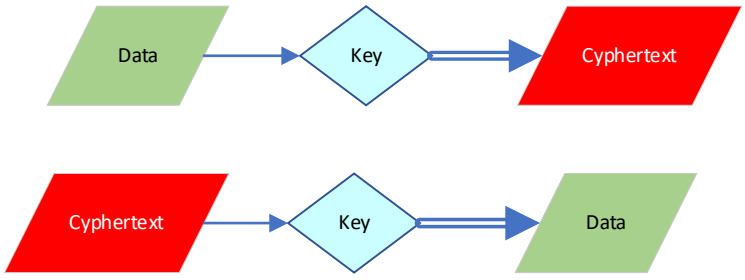
- AES – Advanced Encryption Standard
- Triple DES – Triple Data Encryption Standard
- Blowfish
- Twofish

Asymmetric Algorithms

- RSA - Rivest-Shamir-Adleman
- ElGamal
- Diffie-Hellman
- Elliptic-curve Diffie-Hellman

Symmetric Key Encryption

The above symmetric algorithms are used in symmetric key encryption, meaning that the same key is used to encrypt and to decrypt the data. While this is the fastest type of encryption and easiest to understand, it is also the most prone to compromise, since the key must be common between parties, i.e., be shared with each party that needs to encrypt or decrypt the data. Sharing keys is not always easy to do in a secure manner.



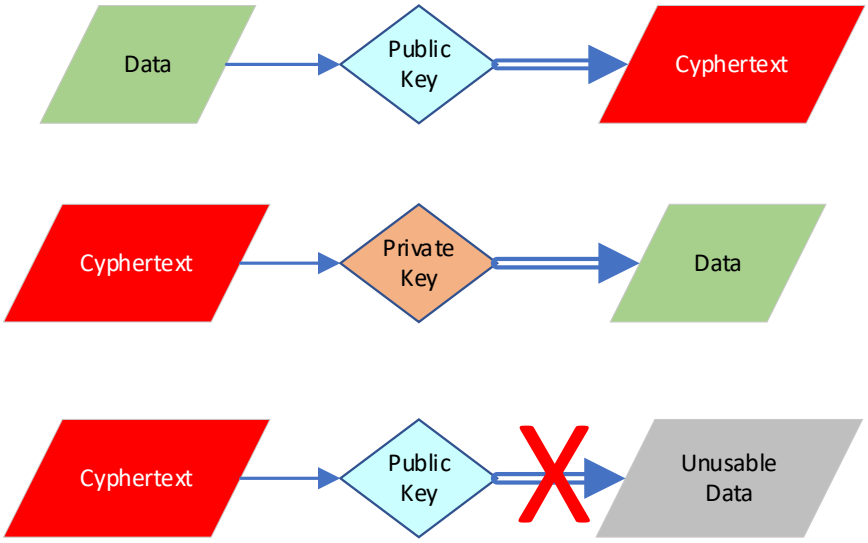
Symmetric key encryption

Asymmetric Encryption

A second type of encryption is called asymmetric encryption, also called public/private key or just public key encryption. This type of encryption does not suffer from the key sharing issue because distinct but

related keys are used to encrypt and decrypt the data. One key is kept secret, while the other is freely sent to another party to be used in the encryption process. The following describes the generalized steps of the asymmetric encryption process however each algorithm listed in the algorithms and methods section may do it slightly differently:

- The Data Receiver creates a public/private key pair and sends the public key to the Data Sender. The private key is kept secret by the Data Receiver and never shared.
- The Data Sender uses the Data Receiver's public key to encrypt the plaintext and sends the encrypted file, or cyphertext, to the Data Receiver.
- The Data Receiver uses their private key to decrypt the cyphertext file received from the Data Sender. NOTE: Only the private key can decrypt the cyphertext created with the public key.



Asymmetric key encryption

Real World Use of Encryption Methods

At this point, you may ask – why not always use asymmetric (public key) encryption for everything? Well, the answer is that public key encryption is painfully slow – 10,000 times slower than symmetric key encryption. While this is a trivial consideration when encrypting small amounts of plaintext, it becomes a much bigger issue when encrypting large amounts of plaintext or when trying to set up real time communications on an "encrypted channel" (think FaceTime on your iPhone or Zoom on your PC).

Most of the time, when we talk about using public key encryption to secure files or data communications, both methods of encryption are used. One typical scenario would be a random symmetric key is generated (usually ≥ 64 characters long) and used to encrypt the plaintext being sent. Then, that symmetric key is encrypted using a public key so that the encrypted symmetric key can be

securely passed along to the receiver with the cyphertext. The process is then reversed on the receiver's side. The asymmetric private key is used to decrypt the symmetric key, which is then used to decrypt the cyphertext.

Key Management

Key management is the hardest part of encryption. Although encryption algorithms can be managed "behind the scenes" by a small number of individuals, key management is something that all parties using encryption must manage properly. Improper key management can result in losses to Confidentiality, Integrity, and/or Availability of the plaintext:

- Keys being stolen or compromised – the data or your communications are no longer secure – you have a loss of Confidentiality and possibly Integrity.
- Keys being lost or deleted – encrypted data cannot be decrypted – you have a loss of availability, since your data are unrecoverable.

We recommend that you establish and follow a good key management policy and – if encryption is widely used to secure data within the organization – use a key management application. If encryption is only being used for transferring data from one party to another, a key management application may be overkill, but a good key management policy will never be. A key management policy will be tailored to the organization, yours may already have one. It is like a password policy and is even sometimes incorporated in with the password policy as a separate section. The key policy would typically touch on items like:

- Required key strength
- Key lifetime – the length of time a key will be valid
- How often keys should be rotated
- Who has access to generate and distribute keys
- Key termination or deactivation
- Where secret keys must be stored
- Where the keys are backed up

While not an exhaustive list of items that should be in a policy the above does provide an overview of the types of issues that need to be considered before starting to manage even a single key for securing data.

An example of a policy template can be found under "Additional Resources."

Key Deletion

When an encryption key is deleted, the plaintext encrypted with that key will become unrecoverable. A good example of this is deleting the encryption key used to encrypt a data file. The data in the file will be unrecoverable and unreadable even though the file is still available. Deleting the encryption key has the same effect as securely deleting the file.

Most of the time key deletion is used to remove sensitive files from backup storage without actually deleting the encrypted file from the backup storage medium. However, this approach must be planned

for in advance to ensure that there is an encryption key per file that can be deleted. This is typically set up within the backup system and will require an enterprise key manager to work.

See the NAACCR Data Destruction Primer for more information on key deletion.

Summary of Encryption

1. Careful with Keys
    a. Compromised keys can lead to undesired disclosures
2. Careful with Changes to Ciphertext
    a. Any changes to ciphertext will result in keys being unusable and plaintext data/information loss
3. Key deletion can be an effective and efficient way to securely remove access to information

Hashing

Hashing is the process of creating a unique cryptographic representation, called a hash value, of the plaintext being processed. A hash value can have many uses. It is most often used to ensure the integrity of a communication or the integrity of a file. The following sections will discuss at a high level the different hash algorithms and properties, and their uses.

Hash Algorithms

There are many different types of hash algorithms, better known as hash functions. Some are used in cryptography to verify communications while others are used to ensure the integrity of files. The following are the more well-known hash functions.

- MD5 – A 128-bit hashing function. No longer considered secure but sometimes still used.
- SHA1 – A 160-bit hashing function. No longer considered secure but sometimes still used.
- SHA256 – A 256-bit hashing function.
- CRC32 – Used mainly for file integrity checking, most install applications have this built in.

At the time of this writing the SHA (Secure Hashing Algorithm) is preferred at 256-bits or above for cryptographic functions. The 256 refers to the number of bits that are output by the function. There is also a SHA384 and SHA512 functions that output the respective number of bits.

Hash Properties

The properties of a hash are as follows:

1. The value output by the hash function will be a fixed length no matter the size of the data being processed. The length depends on the algorithm being used.
2. The hash value generated has a very high probability of being unique.
3. The process of hashing data is non-reversible; the original text cannot be “de-hashed” from the hash value.
4. Hashing the same data will always produce the same hash value.

Property four listed above is the reason that hashed values are important. Because a hash always produces the same value from the same data, hashed values allow us to verify that the plaintext originally hashed is the same plaintext we currently have if the hash values match. In other words, it provides integrity. Although property 4 is a great way to verify data, it does not work well to secure data. This can be illustrated by thinking of a simple number, say “123456789”; the hash of this value would always be the same because that is a property of a hash. By extension, everyone would know that the hash of “123456789” is:

```
2eZ2LdHI6vbWGzxhkvxAjU1tXxF20MKRabwk5xw_J0rSf81YEbMT1oH35V7ALXPUmclUVba1u1A6z1dPuo_-hQ==
```

This would not be very secure.

However, if we add some text to the end of the number, say “abc123” and hash it again, we get a completely different result:

```
Tkp1lgt1kqapgeK9x2j9GrgnpzH8r_97Q6038P9FkUFu8NtBU1K1VnkqAHwEV_znniy6mnVLcqBXX7qXdFpQtA==
```

The process of adding additional and "random" text to plaintext and then generating a hash is called salting, or a salted hash. The added text is called the salt, password, or sometimes a key for the hash. Whatever it is called, it serves the same purpose. It provides randomness to the data and thus security to the hashed data. There are several different ways that this "salting" process can be used, they are described below.

Salting the Hash

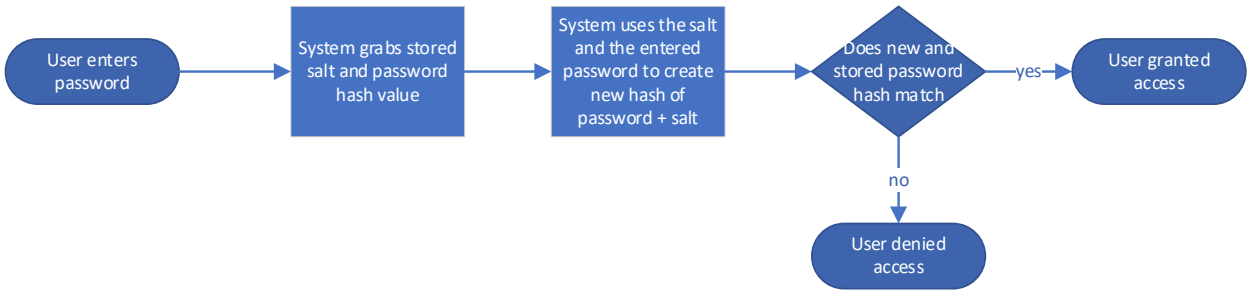
Whether it is called a salt, a password, or a key, the same process is used when using it to secure data. The salt is added to either the beginning or the end of the data to add randomness, and then the entire value is processed through the hashing algorithm. Sometimes the salt may be publicly known, as in the case with passwords stored in a database where a random salt is generated for each password stored. The salt can also be kept secret, as in the case where we want to secure fields within a dataset while keeping their uniqueness.

Public salt values

For the example of passwords stored in a database, the salt is added to the end of the password and then attached to the beginning of the hashed value. This is done so that each hashed password stored looks unique even if it is not. This makes cracking those passwords almost impossible. Using the example from above for the hash of "123456789" with a salt of "abc123," the stored password hash would look like:

```
abc123|Tkp1lgt1kqapgeK9x2j9GrgnpzH8r_97Q6038P9FkUFu8NtBU1K1VnkqAHwEV_znniy6mnVLcqBXX7qXdFpQtA==
```

The salt is prepended to the hash value and separated by a special character "|" to indicate the ending of the salt value and the beginning of the actual hash. This allows the password checking function in a system to always know which salt value to use when generating a new hash value from the entered password to check against the stored hash value. The entire process goes like this:



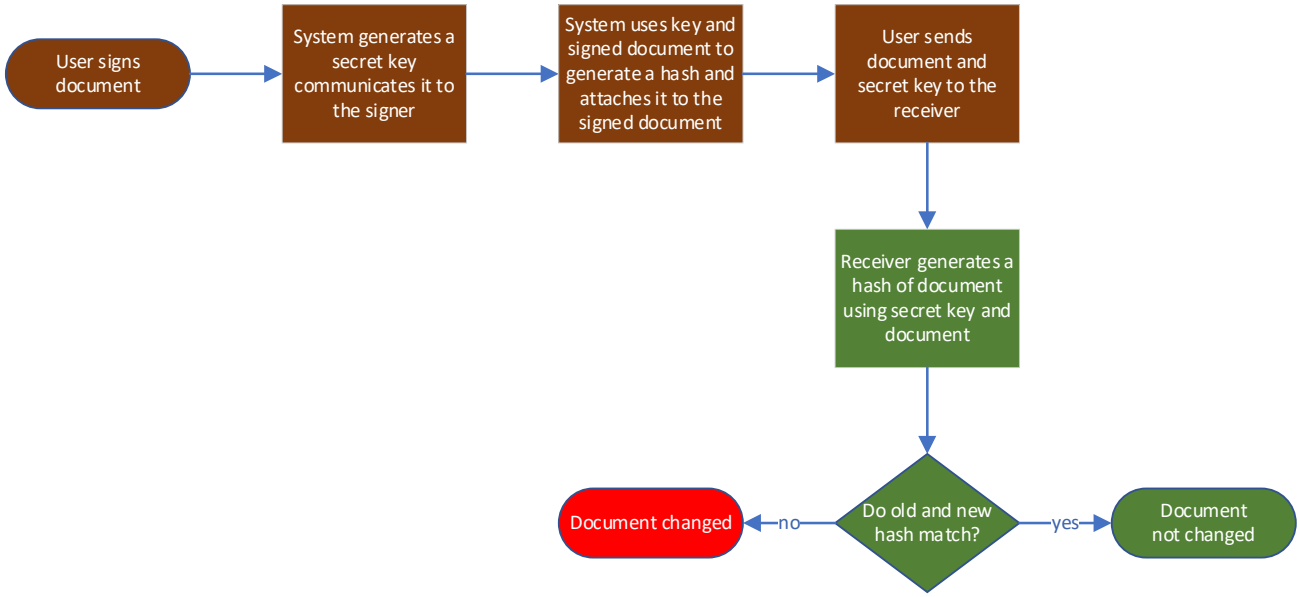
Simple password checking process

Secret salt values

Another way that salting can be used to secure data is by keeping the salt value (key) secret. The salt/key is used the same way as above when generating the hash of the data; however, the salt/key is kept secret and only shared with trusted parties. This is used in the following two scenarios.

Scenario 1

A user wants to digitally sign a document. The document is readable by everyone, but the signature needs to be authenticated by another party, and that party must be sure that the document has not been changed since it was signed. A hash of the document, using a secret key, can be created, including the signature once the document is signed. The party wishing to authenticate the document would then securely receive the secret key and regenerate the hash of the document and signature. If they match, the document is unchanged. The process would look something like this:

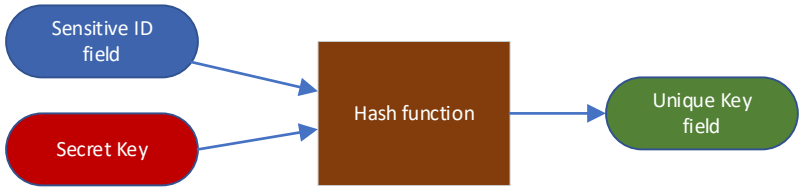


Simple document signing example

While this is not the exact process behind digital signatures, it does illustrate what happens and how hashes are used.

Scenario 2

A primary researcher has data that they want to share with a colleague, but they do not want to include a sensitive ID field in the data. However, they want to a way to provide updates to the data without having to generate a unique ID field and a crosswalk file for the sensitive ID field. This can be accomplished by using a long random salt that only the primary researcher has access to. For our purposes, we will call the long random salt a key. The key is added to the end of the sensitive ID field and the combined value is processed by the hash function. The resulting value is a representation of the sensitive ID field that is secure if the key is kept secret.



Simple field hash with secret key

If the data need to be updated or added, the same secret key can be used by the primary researcher to regenerate the data and the same unique key field will be created.

Using hash values

What can hashes be used for

- Storing passwords in a database following best practices
  - Use at least SHA256 hashing function for password
  - Always use a unique salt for each password stored
- Verifying the integrity of a file or message
  - Use a SHA, MD5 or CBC checksum
- Verifying the integrity of a record or key fields of that record
  - Use SHA hash
- Maintaining the confidentiality of a field when a secret key is used

What hashes should NOT or can NOT be used for

- Maintaining the confidentiality of fields with limited values (i.e., SSN, first name, last name, and text fields of less than 6 characters like short passwords) when a common **public** or common **known** salt/key is used.
  - Small fields are insecure, since the total combinations of items is small (< 1 billion). With small fields, calculating the hashes for all possible combinations, the fields are relatively quick to crack if the key or salt is known.

    For illustration, hashing all the possible SSNs with a known key takes less than 60 minutes on a reasonable sized computer; hashing all common first names or surnames take even less time. Those lists can be found on the Internet.

- Encrypting data. As stated above, hashes are one way. Encryption, by definition, can be reversed.

The first bullet point under what hashes should not be used for deserves a little more explanation. The process of generating all possible hashes using all possible values of a field using a single salt or key is referred to as generating a rainbow table. This process was first used to crack passwords that did not include a random salt on each password. The attack allows an attacker to pre-generate a table of possible hash function outputs for common passwords, and easily match the hashed values to plaintext passwords. Therefore, a random salt is now added to all passwords that are stored in a database to

prevent this kind of attack. In the scenario above, the same key/salt is used to hash all fields in the database. A rainbow table can be generated for the defined field values and the known salt/key. For fields containing less than 1 billion possible values such as SSN, common first names, common surnames, these tables can be generated to re-identify the values, much like a lookup table.

Data Matching using Hash Values

Hashing is often used in data matching when two parties would like to match subjects by some sensitive fields, like SSN, address, or name but do not want to have the actual values of the fields available in the dataset for the other party to see. This is usually used for security purposes when transferring the data, and it can also be used to obscure the sensitive data from the other party. However, there are caveats with how secure this is, the main issue being that shorter fields like SSN can be reverse hashed by the other party, since the salt/key used to create the hash must be known to both parties so that a "matchable" hash can be created.

If the data must be kept secret from the other party, the only way to do so is to use a trusted 3rd party to do the matching. This will allow fields with low entropy like SSN to be matched securely. Some examples are provided in the Trusted 3rd party section below, where these topics are discussed in further detail.

Security considerations when doing matching

- Do the data need to be kept secure from inadvertent disclosure?
- Do the unmatched data need to be kept secure (unable to be re-identified) from the matching party?
- Who will be doing the matching?
- Who will provide the salt/key to hash the data?
- How will the salt/key be provided?
- Which data fields will be matched?
- Who will have access to the hashed data file and the program and/or salts/keys used to generate the hashed data file?

The above are considerations that need to go into keeping data secure during the matching process. The main thing to remember is that whoever controls the salt/key for the hash also controls the overall security of the data being shared.

Keys Embedded Within Software

It should also be noted that some applications have the salt/key embedded within the software. Data stewards should not rely on salts/keys embedded in software for security, as all software can be decoded, and keys found within the decompiled code no matter how well they are hidden.

'Double Hashing'

Some matching procedures talk about "double hashing" sensitive fields that have a short length for extra protection. For example, the SSN is hashed with one salt/key and then the output of the first hash is hashed again with another salt/key. If a party has access to both salts/keys used in the hashing process (which they would since they both must be used to generate the hash files), it is not much more difficult to decode the field than it is for a single hashed value. As noted above in this document, it would take less than 1 hour to crack an SSN hashed field knowing the salt/key. For a double hashed

field, it would take approximately 2 hours to create the necessary files to crack the field. Double hashing does not remove the security issues associated with fields with limited values (< 1 billion combinations). If the salt/key is unknown to the attacker, the field would be secure either being single hashed or double hashed.

Commonly Used Fields at Increased Risk of Cracking

While the discussion has been focused on short fields like SSN, other fields are also susceptible to "cracking." Name fields are also vulnerable since there are lists of first names and surnames commonly available on the Internet that could be fed into a program that can generate possible hashes with a known salt/key. The key point here is that data are only secure if the salt/key or "password" is kept secure. If too many people know the salt/key, or if it can be decompiled or deconstructed from a program, the hashed data are not really secure, even though a hash is only one way.

Note: The low entropy field issue is why your IT department makes you use a "long, complex" password. (Entropy is dependent on the number of characters, whereby shorter fields have lower entropy, i.e., are more easily guessable.)

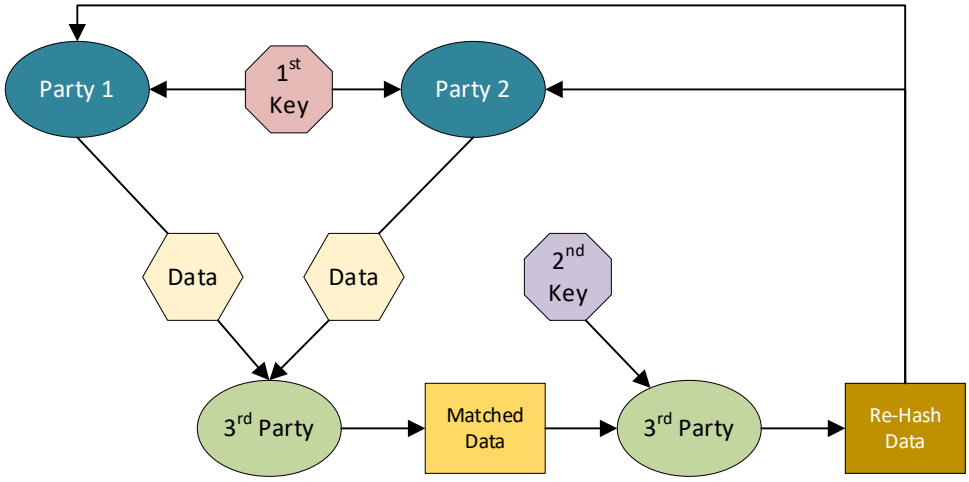
Trusted 3rd parties

A trusted 3rd party can be used for matching so that the parties providing the data would not have access to data from the other party. The trusted 3rd party would be the one controlling the flow of data and the matching process. Two scenarios that may help to explain the process are described below.

Example 1:

> Party 1 and Party 2 want to match data on SSN. However, both parties have a state mandate that no sensitive data can be revealed to entities outside the state. They employ a trusted 3rd party to do the match. Both parties agree between themselves on a salt/key to use for the hashing, but do not share that key with a trusted 3rd party. The trusted 3rd party receives data files from each of the two parties and performs the matching of records between the datasets. The trusted 3rd party then uses another salt/key that only they know to hash the hash value of the SSN in the combined dataset so the combined dataset can be returned to both parties for analysis without either party being able to reidentify the records.
>
> Note: A minor caveat to this is low case numbers for records with unique data. One party may still be able to reidentify some records if unique case counts are contained in the matched data.

Example data flow for Example 1

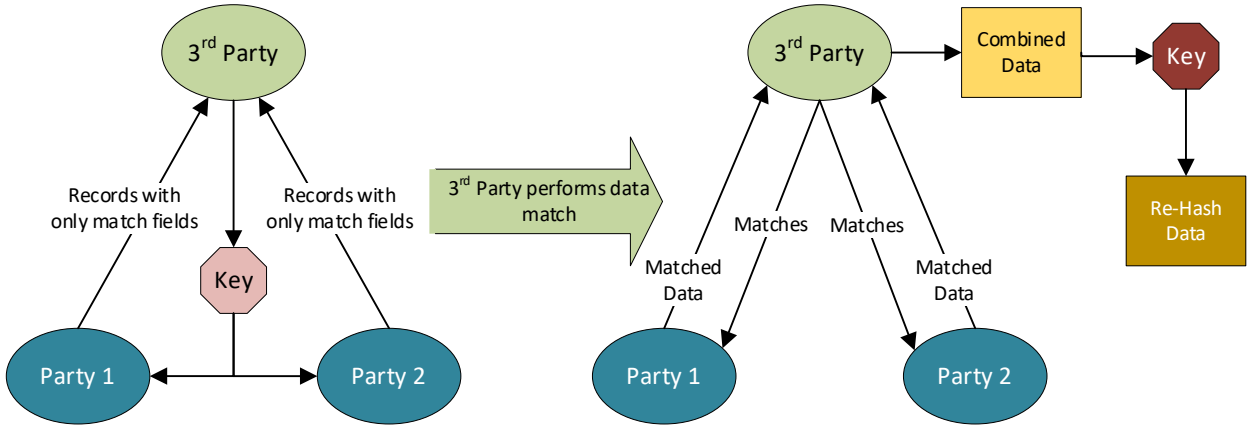


Example 2:

Two or more parties want to see if they have potential matches in their datasets, and would like to combine data on matched cases only without identifying/sensitive fields. The matching criteria is agreed upon and a trusted 3rd party is chosen to oversee the match process. The 3rd party provides a program to each participant that will first normalize fields and then individually hash them. Each participant runs the program and sends the resulting hashed file to the 3rd party to be used for matching. The 3rd party would then execute the matching program to find the records that match in each participant's hashed dataset. The 3rd party could then send the matched record IDs back to each of the participating parties so the full record can be sent to the trusted 3rd party for the creation of the combined dataset. A combined dataset is created from the records by the 3rd party. None of the sensitive fields would be included in the combined dataset.

Depending on the use of the dataset and whether it is important to be updated in the future, an identifying ID could be generated from the original hashed data, probably by using the SSN field or other unique field combination and hashing it again with a new salt/key. This new salt/key would need to stay with the 3rd party so the data could not be reidentified by anyone else and would allow updates to the records.

Example data flow for Example 2



Hashing Summary

Hashing can be used for many things when it comes to data integrity and confidentiality. Hashing can be used to ensure files have not changed since they were created, and it can be used to validate the integrity of an email. It can also be used to secure passwords within a database or sensitive data fields within a data file. However, when hashing is used for maintaining confidentiality extra steps and extra thought must be put into the process to ensure the confidentiality of the protected data is maintained.

The above sections attempted to provide a high-level overview of hashing, its uses, and some issues that need to be considered when using hashes. Those sections should not be considered everything that must be addressed or known on the subject. This only scratches the surface and is intended to provide a starting point for further exploration and understanding.

Additional Resources

We have included some additional resources for encryption and cybersecurity. These resources range from easily interpretable to highly technical, but are provided as a starting point for reliable resources for the registry community.

National Institute of Standards & Technology

https://www.nist.gov/cybersecurity

General guidance on all cybersecurity issues from the perspective of the U.S. government.

National Cybersecurity Society

https://nationalcybersecuritysociety.org/

Membership-based organization that focuses on small to medium sized businesses. Offers some free and reliable practical best practices documents. Membership provides access to more advanced guides and templates; although membership prices seem reasonable, additional resources behind membership paywall have not been vetted.

Example of an encryption (including key management) policy:

https://nationalcybersecuritysociety.org/wp-content/uploads/2019/10/Encryption-Policy-Template-FINAL.pdf

Center for Internet Security (CIS)

https://www.cisecurity.org/

CIS provides lists of security controls, including encryption best practices, and CIS benchmarks.

Cybersecurity & Infrastructure Security Agency

https://www.cisa.gov/

Provides general information about cybersecurity, including whitepapers on best practices.

CISA document describing optional best practices for encryption key management:

https://www.cisa.gov/sites/default/files/publications/08-19-2020_Operational-Best-Practices-for-Encryption-Key-Mgmt_508c.pdf

Appendix – Key Managers / Encryption Applications

The following is a non-extensive list of commonly used applications that can be used to encrypt data and/or manage keys. This is not a recommendation of any product listed. Many online file sharing platforms provide their own key mangers to help the user manage their encryption keys.

Encryption Software

Below is a list of software that can be used to encrypt/password protect files. Please note that there are many file sharing services that provide security for sharing files and use encryption when storing those files on their servers.

- GPG (Gnu Privacy Guard) – Used to manage encryption of files and messages, and also manages private, public keys for users.
- 7-zip – File archive utility that can create a password protected encrypted archive file using ASE 256 symmetric encryption.

Password / Secret Management

The list below gives some examples of password/secret management servers that can be used to manually store and manage encryption keys securely.

- HashiCorp Vault – Password, secret, and encryption key management for local or enterprise use.
- LastPass – Internet-based Password and secret manager, which can be used to manually store encryption keys.
- Thycotic Secret Server – Password, certificate, and secret management server for enterprise use.

Key Management Servers

These are dedicated servers or services that handle managing encryption keys using a standard protocol. These servers can be integrated with hardware devices such as hard drives or storage servers, and software such as backup applications, to provide key management services for the automatic encryption of data.

Please note that these are enterprise-level devices. Most large cloud providers also provide key management servers to secure their infrastructure. Below are a couple of the on-premises providers.

- Townsend Security – Enterprise Key Management Server (KMS)
- Dell OpenManage Secure Enterprise Key Manager