7.3.3 Hashing Facts

This lesson covers the following topics:

- How hashing works
- Hashing uses
- Hash collisions
- Hashing algorithms
- Comparing hash values

How Hashing Works

Hashing is the process of generating a fixed-length hexadecimal string value from any file type or data. Hashes can be generated from messages, image files, data files, and most other types of data. This output is known as the message digest or hash.
The output size varies depending on the algorithm being used.

A proper hash function should have the following characteristics:

Characteristic	Description
Deterministic	The same data always generates the same hash.
Quick and efficient	Generating the hash should be quick and not use too many resources. The hash should also be generated using the entire message or data, not just a small piece of it.
One-way	The hash cannot be reverse engineered.
Collision resistant	Two different pieces of data should not generate the same hash.
Avalanche effect	Changing any bit of data will result in a completely different hash.

Hashing Uses

Hashing is often used for the following:

Hash Use	Description
File Integrity	Hashes are often used to prove the integrity of downloaded files. When a file is uploaded to a site, a hash can be generated. When the recipient downloads the file, they can create a hash of that file. If the recipient's hash matches the hash of the original file, you know that: ▪ The downloaded file is complete (no missing parts). ▪ The downloaded file was not corrupted during transfer. ▪ The downloaded file is the same as the original and has not been altered by inserting malicious code or replaced with a virus or malware file. For this reason, files available for download are typically not encrypted as the hash proves their data integrity.

Digital signature	Hashes are a critical piece of a digital signature. The creator combines a hash of the data along with their private key to generate the digital signature.
Secure logon credential exchange	Hashes can be used to secure logon credentials during an exchange. The password is used as the key to perform a hash on a text value, and only the hashed value is passed (not the password). The receiving host uses the same method to compare the hashes to verify the identity of the user. Examples of protocols that use this method are: ▪ Challenge-Handshake Authentication Protocol (CHAP) ▪ New Technology LAN Manager (NTLM) ▪ Kerberos Passwords can be further secured by salting the hash. This is the process of adding random characters at the beginning or end of the password to generate a completely different hash. If a hacker intercepts the hash, they also must know which portion is the salt before beginning to crack the hash.

Hash Collisions

Hashing is a good file verification method, but it is not perfect. Depending on the algorithm used, there is a potential for hash collisions. A hash collision occurs when two completely different files generate the same hash. Rainbow table attacks take advantage of hash collisions.

- A rainbow table is a table of passwords and their generated hashes. A hacker can use this table to try to match hashes instead of the actual password.
- Hash collisions can be reduced using an algorithm that generates a longer hash and by salting the hash. Salt is random data that is used as an additional input to the function that hashes data.

Hashing Algorithms

Depending on the use, there are different hashing algorithms which can be used.

Hashing Algorithm	Description
Message-Digest Algorithm 5 (MD5)	MD5 was developed by Ron Rivest in 1991. ▪ MD5 generates a 128-bit message digest. ▪ Many security vulnerabilities have been discovered with MD5. As such, it is no longer viable for security purposes. ▪ MD5 is extremely susceptible to hash collisions. ▪ MD5 is mainly used for file integrity.
Secure Hash Algorithm (SHA)	SHA is a family of hashes. ▪ SHA is a government standard. ▪ First published in 1991 by the National Institute of Standards and Technology (NIST). ▪ SHA-2 was published in 2001 and has become one of the standard hash functions in use today. ▪ Used in many security protocols such as TLS, SSL, PGP, SSH, and IPSec. ▪ Generates message digests that are 224, 256, 384, or 512 bits in size.

	SHA-3 was published in 2015, but is not meant to replace SHA-2. SHA-2 has yet to be cracked. NIST wanted an alternative available for people to use. Message digests generated by SHA-3 are fully compatible with SHA-2.
Hash-Based Message Authentication Code (HMAC)	HMAC is a type of message authentication code. Like a digital signature, HMAC allows a user to verify that a file or message is legitimate. ▪ The message sender provides a secret key that is used with a hash function, such as MD5 or SHA, to create a message authentication code. ▪ The recipient then uses the key to verify both the integrity and authenticity of the message.
RACE Integrity Primitives Evaluation Message Digest (RIPEMD)	RIPEMD (RACE Integrity Primitives Evaluation Message Digest, or RIPE Message Digest) is a family of cryptographic hash functions that was first developed in 1992 as part of the EU's RIPE project. The first version was based on the MD4 function. In 1996, in response to security issues discovered in the first version, Belgian researchers developed four updated algorithms. ▪ RIPEMD-128 ▪ RIPEMD-160 ▪ RIPEMD-256 ▪ RIPEMD-320 RIPEMD is not as popular as SHA-2, but is used frequently with Bitcoin and other cryptocurrencies.

Comparing Hash Values

Being able to compare the hash of a file after it's been downloaded to a known good hash helps verify that the file was not altered in transit.

To compare file hashes, you must first generate a hash for the file after it has been downloaded. This can be done using the PowerShell cmdlet named **Get-FileHash**. In the following example, a hash is being generated for the file named Download.zip. The **-a** (or – algorithm) switch specifies the algorithm to use when generating the hash. It is followed by the desired algorithm, in this case MD5.

Example: **Get-FileHash Download.zip -a md5**

Sample output:

```
Algorithm       Hash                                   Path
---------       ----                                   ----
MD5             A9ESDS4B7288811EC080948E10909A         C:\Downloads\Download.zip
```

Companies often provide a separate file containing the hash of a file that can be download and used to compare to the hash you generate for the downloaded file. This is often in the form of a text file. To view the hash you can simply open the text file or use PowerShell's **Get-Content** cmdlet to extract the hash from the file.

Example: **Get-Content Download.txt**
Sample Output: **4A84C7958C246E39439C784349F4ZDB4**

Once you have access to the two hashes, you can visually compare them to see if they are the same or an easier way is to use the **-eq** command to compare to two hash values.

Example: **“39C784349F4ZDB44A84C7958C246E394” -eq “4A84C7958C246E39439C784349F4ZDB4”**

The output will be “True” if the hashes match or “False” if they do not match.