Info Security Technology

Topic 5
Cryptography
(Hashing)

Objectives

- Understand and apply Hashing for Integrity checks
- Understand the various algorithms used in Hashing
- How to secure a password with Hash
- How to crack a password using Hash

Key Complexity

- The more complex the key, the greater the security of the system.
  - **Key complexity** is achieved by assigning a large number of possible values to the key.
- The **keyspace** is the size of every possible key value.

Brute Force

- All encryption ciphers besides a "one-time pad" cipher are susceptible to a **brute-force attack** — attempting every possible key.
- E.g

0
00
01
02
03
.
001
002
003
.
010
011
012
.
A
AA
AB

Brute Force

Password Length In Characters	Alphabetic, No Case (N=26)	Alphabetic, Case (N=52)	Alphanumeric: Letters & Digits (N=62)	All Keyboard Characters (N=~80)
1	26	52	62	80
2 (N^2)	676	2,704	3,844	6,400
4 (N^4)	456,976	7,311,616	14,776,336	40,960,000
6	308,915,776	19,770,609,664	56,800,235,584	2.62144E+11
8	2.08827E+11	5.34597E+13	2.1834E+14	1.67772E+15
10	1.41167E+14	1.44555E+17	8.39299E+17	1.07374E+19

Source: Panko, “Corporate Computer and Network Security”

Hashing Algorithm

- A hash is a special mathematical function that performs one-way [encryption] calculation.
- Once the algorithm is processed, there is no way to reverse the process to obtain the original text
- Hashing is primarily used for comparison purposes.

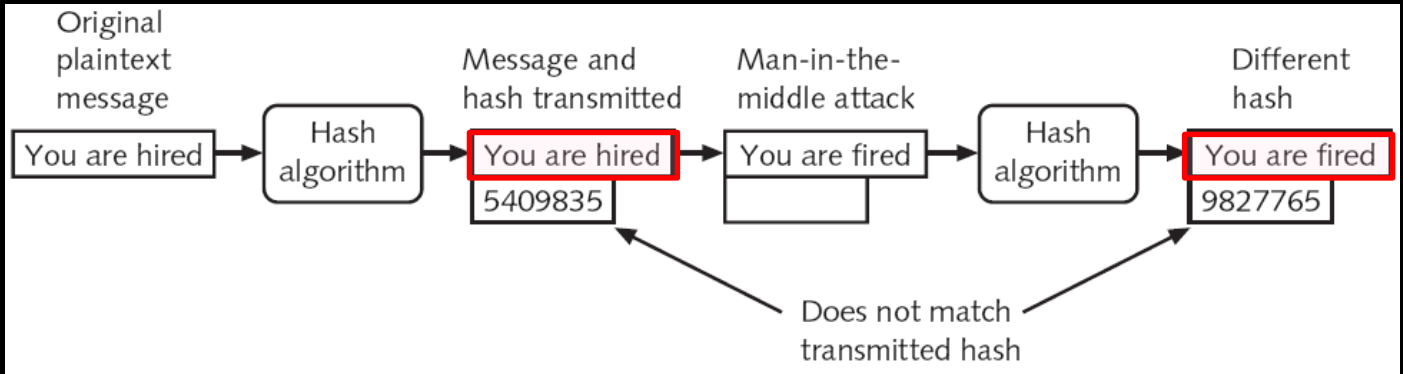
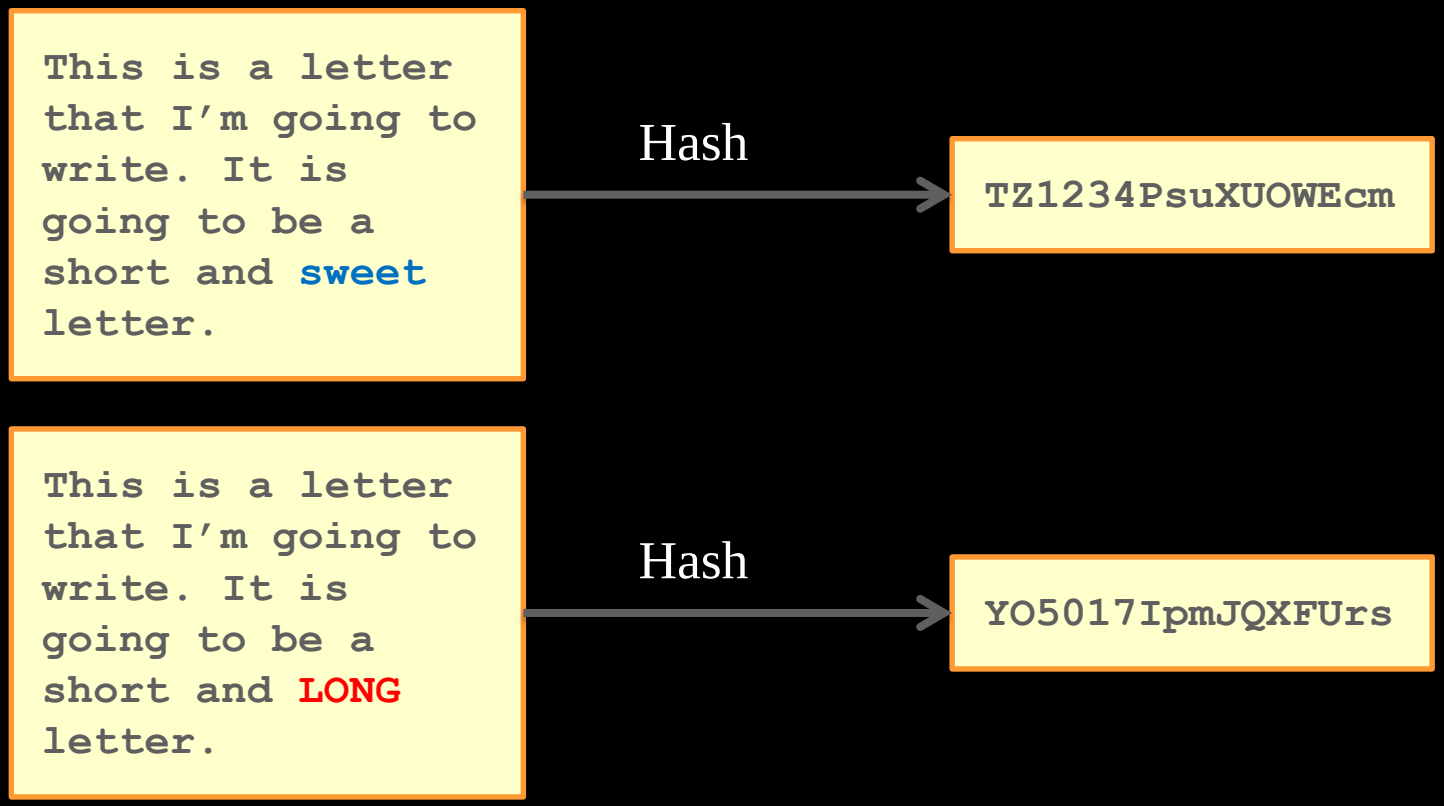


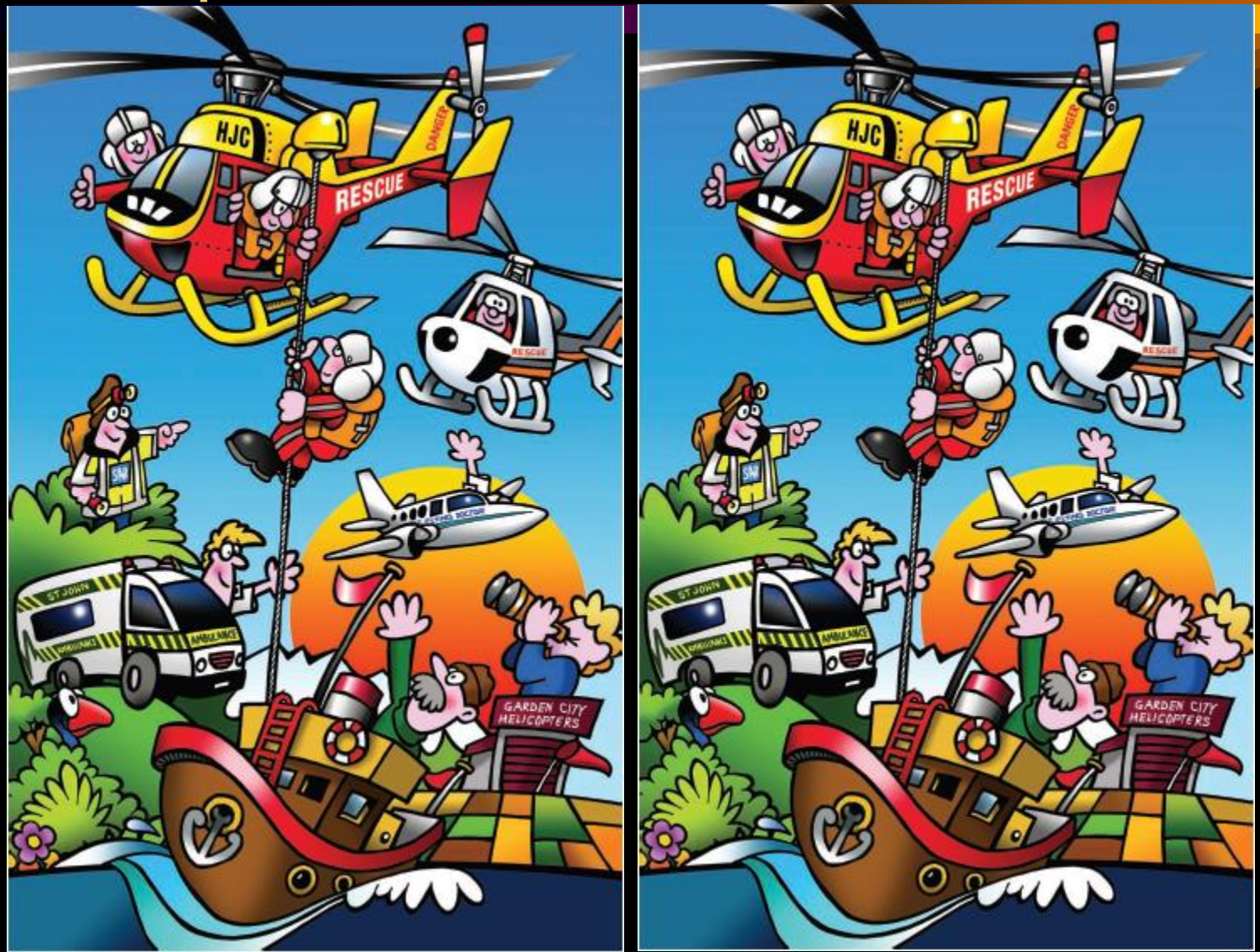
Figure 11-4 Man-in-the-middle attack defeated by hashing

Hashing Algorithm

- Converts a variable length input to a fixed length
  - Creates a "Hash","digest" or "data fingerprint"
- "One-way" – easy to compute, but cannot reverse



Spot the difference



Spot the difference (10)

Hashing Algorithm

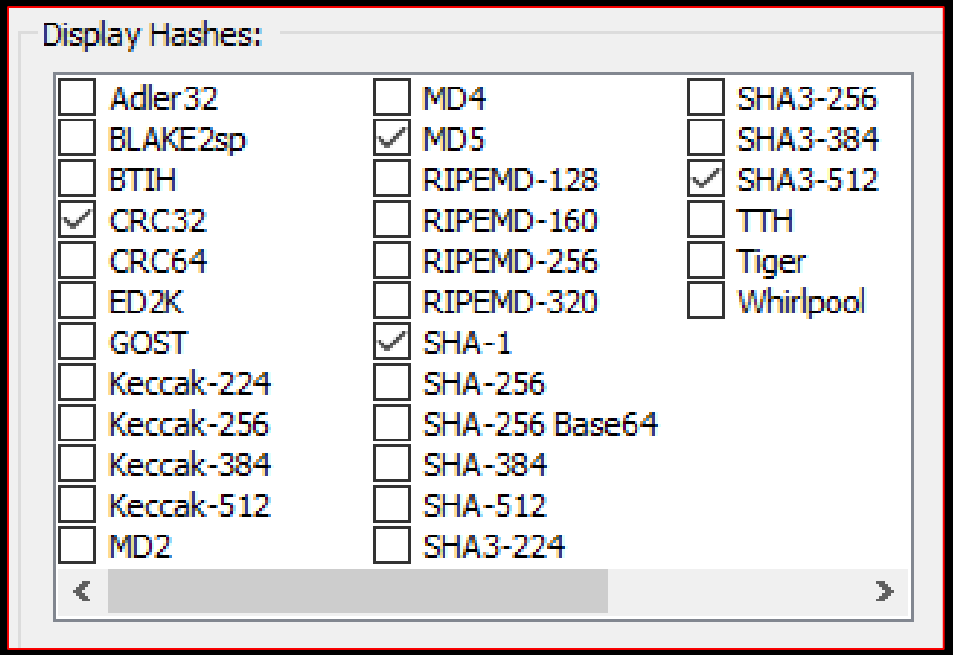
- The hash serves as a check to verify the message contents
- Hashing is used only for integrity to ensure that:
  - No unauthorized person or malicious software has altered the data
- A hashing algorithm has the following characteristics:
  - Fixed size: The hash is always the same size regardless of the length of the plaintext.
  - Unique: Two different plaintexts cannot produce the same hash (i.e. collision)
  - Secure: The resulting hash cannot be reversed to determine the plaintext.

Hashing Algorithm

- Applications of Hash
  - ✓ Post hash values of files (eg software applications) on Internet download sites
    - To verify the file integrity of files that can be downloaded
  - ✓ Use hash to store passwords
    - When a password for an account is created, the password is hashed and stored

Hashing Algorithm

- Popular hashing algorithms
  1. Message Digest (MD) algorithm (128 bits)
  2. Secure Hash (SHA) Algorithm (256 bits)
     - SHA1 (160 bit hash)
     - SHA-256, SHA-384, SHA-512
     - SHA3-256, SHA3-512
  3. Whirlpool (512 bits)



Hashing Algorithm

- Information protections by Hashing

Characteristic	Protection?
Confidentiality	No
Integrity	Yes
Availability	No
Authenticity	No
Non-repudiation	No

Table 11-2 Information protections by hashing cryptography

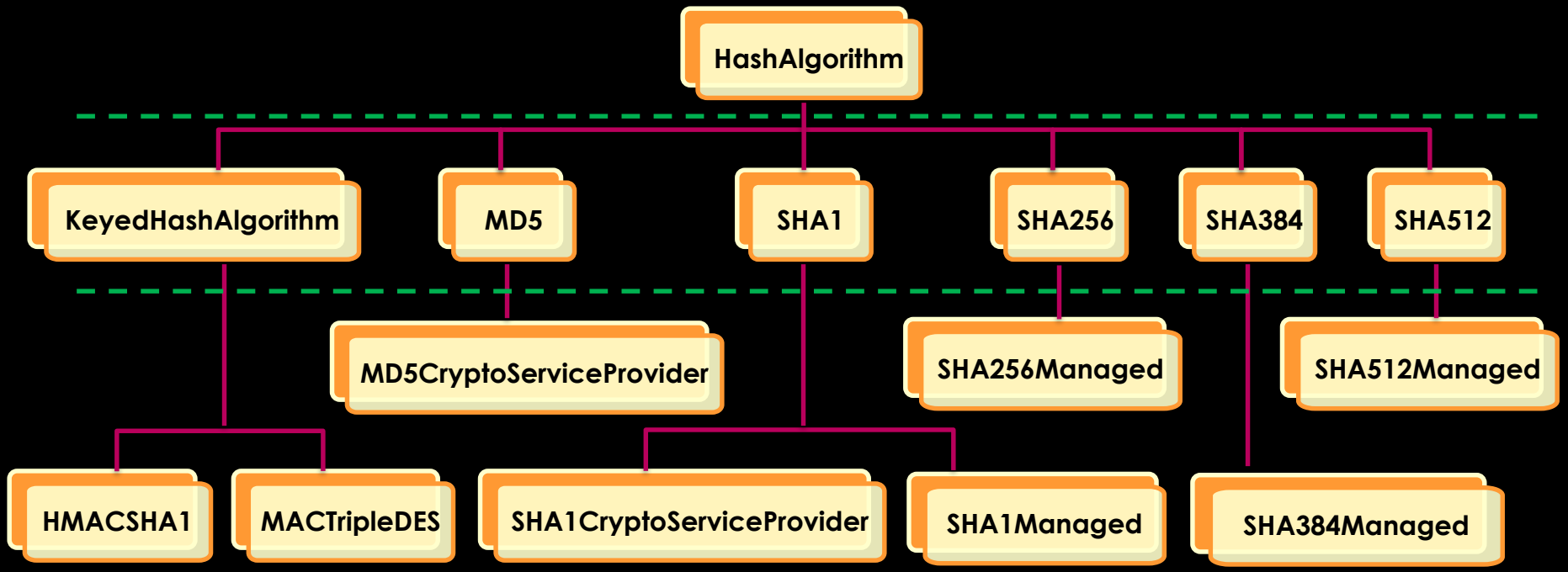
Comparison of Algorithms

Algorithm Types	Description
Hash	▪ Performs only a one-way encryption. ▪ There is no way retrieve the plaintext from the hash value. ▪ Purpose is to **verify the file integrity**

Symmetric	▪ Uses ONE key to:
	  ❑ Encrypt data
	  ❑ Decrypt data
	▪ **Is fast & efficient**

Hash Algorithms

- Hashing is supported in .NET through the HashAlgorithm class.
- HashAlogrithm Class Hierarchy:



Using Hashing Algorithm in .NET

```
String text = "Plain text to be hashed";
byte[] rawTextBytes = Encoding.UTF8.GetBytes(text);

SHA512Managed sha512 = new SHA512Managed();

//SHA512 return 512 bits (64 bytes) hash value
byte[] hashBytes = sha512.ComputeHash(rawTextBytes);

//Converts the 64 bytes hash into a string "C4-2A-FB-54-...",
// each pair represents one byte.
string hex = BitConverter.ToString(hashBytes);

//Prints out the hash
Console.WriteLine(hex);

//Prints out the hash, encoded into Base64
Console.WriteLine(Convert.ToBase64String(hashBytes));
Console.Read();
```

1. Convert string to byte []
2. Create hash algorithm object
3. Call ComputeHash

Hashed Passwords

- Best practice for passwords (cannot be retrieved)
- Used for authentication
- Common attack against hashed passwords is “dictionary attack” – use salted passwords

Hashing

User		Hash
user1	Nobody should see the actual password. Not even the administrator!	1234
user2		2345
user3		3456
user4		1234
user5		2345

Hashing

User	Password	Hash
user1	one	1234
user2	two	2345
user3	three	3456
user4	one	1234
user5	two	2345

Hashed Passwords

```
using System.Security.Cryptography;
string pwd = "pwd12345";

//Generate random "salt"
RNGCryptoServiceProvider rng = new RNGCryptoServiceProvider();
byte[] saltByte = new byte[8];

//Fills array of bytes with a cryptographically strong sequence of random values.
rng.GetBytes(saltByte);
string salt = Convert.ToBase64String(saltByte);

SHA512Managed hashing = new SHA512Managed();

string pwdWithSalt = pwd + salt;
byte[] plainHash = hashing.ComputeHash(Encoding.UTF8.GetBytes(pwd));
byte[] hashWithSalt = hashing.ComputeHash(Encoding.UTF8.GetBytes(pwdWithSalt));

Console.WriteLine("Hash without salt:" + Convert.ToBase64String(plainHash));
Console.WriteLine("Hash with salt:" + Convert.ToBase64String(hashWithSalt));
Console.Read();
```

RNGCryptoServiceProvider

- **RNGCryptoServiceProvider** generates high-quality random numbers. With it, we use an RNG (random number generator) that is as random as possible. This helps in applications where random numbers must be completely random.
- **Caution:**RNGCryptoServiceProvider has a cost: it reduces performance over the Random type.

How Random ?

So when should you use Random and when should you use the slightly more complicated RNGCryptoServiceProvider?

- for most programs, Random is sufficient and preferable due to its simplicity.
- For important programs RNGCryptoServiceProvider is better because it is less prone to problems with its randomness.

Does Random use the RNGCryptoServiceProvider internally?

Hashing with Salt

User	Password	Random Salt	Resultant Password String	Hash
user1	one	12	one12	47783
user2	two	23	two23	98376
user3	three	34	three34	19462
user4	one	45	one45	05483
user5	two	56	two56	87572

hashed

Hashing with Salt (What is stored in the DB?)

User	Nobody should see the actual password. Not even the administrator!	Random Salt	Nobody should see the actual password. Not even the administrator!	Hash
user1		12		47783
user2		23		98376
user3		34		19462
user4		45		05483
user5		56		87572

- Change the 'Salt' the next time user changes his/her password
- Force password change every 3 months to ensure that Salt is renewed