Security in Computing & Information Technology (COSC2536)

Lectorial 1: Basic Cryptographic Techniques

Lecture Overview

- During this session, we will discuss:

Introduction to the course

Part -I: Symmetric Key Cryptography

- Concepts of Symmetric Key Cryptography
- One-Time Pad, Stream Cipher, Block Cipher

Part –II: Cryptographic Hash Functions

- Basics of Hash Functions
- How Hash works, properties of Hash
- Applications of Hashes

Staff

- Course Coordinator and lecturer:
  - Shekhar Kalra (SK)
  - shekhar.kalra@rmit.edu.au
  - Consultation: set up a time via email
- For a detailed list of other staff members teaching in this course, please look at
  - Course Canvas – Modules- Welcome - Teaching Team page

Course has been rejuvenated

- Various new topics have been introduced to make the course relevant to contemporary industry relevant
  - Focus on programming of cryptography using Python
  - Discussion on AI cybersecurity
  - Discussion on use of Machine Learning (ML) for detecting and preventing fraud
  - Discussion on emerging technologies and careers in cybersecurity
  - Reduction of abstract Math from the course material

General tone of the course

- Mix of concepts and practice
- Not an entirely theoretical course
- You will need to program in Python
- To do well in assessments, attend both the lectorial and practical sessions
- Making assumptions and not checking with staff may lead to an unsatisfactory outcome
- Easy to get a CREDIT as overall grade but
  - hard to get a DISCTINCTION and
  - harder to get a HIGH DISCTINCTION

Important advice about assessments

Assignment 1 - (35%)

- Submission: Due SUNDAY, 25 August 2024 (week 5)
- Assignment 1 will focus on writing an analytical report on industry focused secure applications where you apply your abilities, conceptual understanding and knowledge on the topics that are covered in the course.
- To be completed **individually or a group of 2**

Assignment 2 - (45%)

- Submission: Due SUNDAY, 06 October 2024 (week 10)
- Assignment 2 will focus on programmatic implementation (*writing code*) of cryptography techniques, digital signatures & fraud prevention using Machine Learning
- Submit complete code project and documentation
- To be completed **individually or a group of 2**

Assignment 3- End-of-Semester in-class face-to-face timed-exercise (20%)

- Assignment 3 will focus on all aspects of security, cryptography, protocols, and applications taught in this course. The objective of Assignment 3 is evaluating problem solving ability and critical analysis in the form of several short questions.
- Duration: 2 hours and 15 minutes on campus
- To be held in week 15 (MONDAY , 04 November 2024, *3.15 pm - 5.30 pm*)
- **To be completed individually**

Group work blues

- Maintain communication logs
- Maintain meeting details
- Create a task allocation list
- Come to me when the problem starts
- Do not complain about your group partner near or after deadline
- I will need proof to intervene and moderate the discussion between you the group members

Do I need my own laptop?

- Yes
- You will need to install Python and Visual Studio Code
- And other tools, libraries as we advise you
- DO NOT MAKE ASSUMPTIONS ABOUT THE USE OF TOOLS, LANGUAGES and FRAMNEWORKS
- Use the officially advised tools and software(s).

Is there a prescribed textbook?

- We will follow
  - Security in Computing, Charles Pleeger et al.
  - Prentice Hall
  - *both 5th or 6th edition(s) are fine*

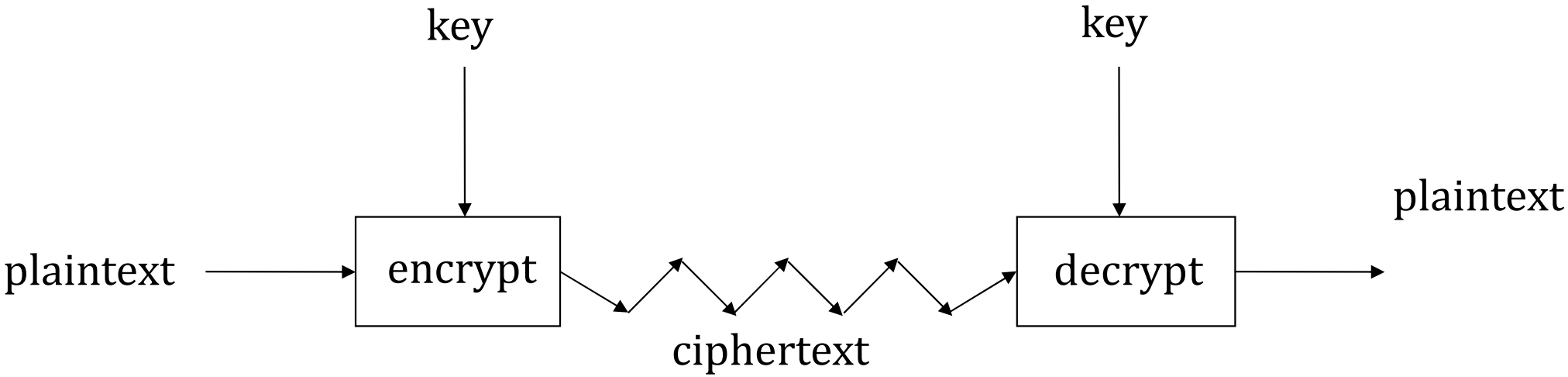
Can I use ChatGPT?

- Yes
- You need to learn how to use generative AI to work and exist in the contemporary industry
- But use of ChatGPT is NOT THE SAME AS cheating and using it to do your work
- You need to learn its **e**ffective, **e**thical and **e**fficient use
- It is a skill
- ChatGPT is NOT a search engine
- If you ask silly questions, it will churn out silly answers

Segment 1: Symmetric Key Cryptography

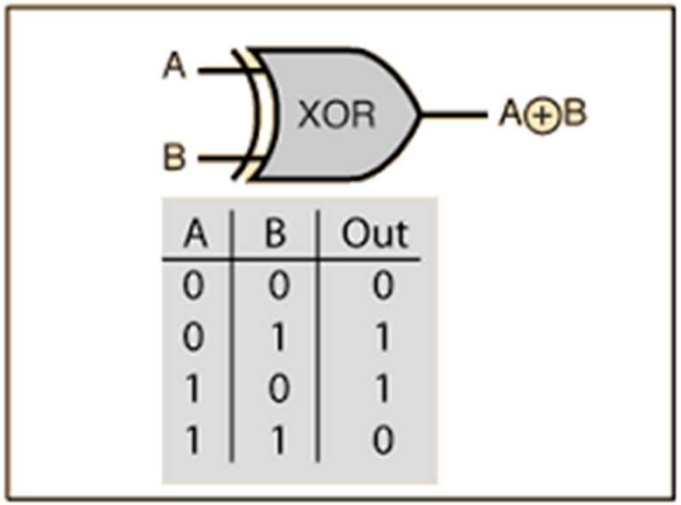
How to Speak Crypto

- A *cipher* or *cryptosystem* is used to *encrypt* the *plaintext*
- The result of encryption is *ciphertext*
- We *decrypt* ciphertext to recover plaintext
- A *key* is used to configure a cryptosystem
- A *symmetric key* cryptosystem uses the same key to encrypt as to decrypt
- A *public key* cryptosystem uses a *public key* to encrypt and a *private key* to decrypt (we will learn these in lectures 3 and 4)



A generic view of symmetric key crypto

What is XOR and how XOR works



A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

We denote the XOR of bit A with bit B as $A \oplus B$. You can notice that only $0 \oplus 1$ or $1 \oplus 0$ produces output 1

One-Time Pad: Encryption

The one-time pad requires a key consisting of a randomly selected string of bits that is the same length as the message. The key is then XORed with the key to yield the plaintext

e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Encryption: Plaintext⊕Key = Ciphertext

$P \oplus K = C$ Or, in some texts $e_k(m) = k \oplus m$

	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000
Ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

One-Time Pad: Decryption

The ciphertext is XORed with the key to yield the plaintext

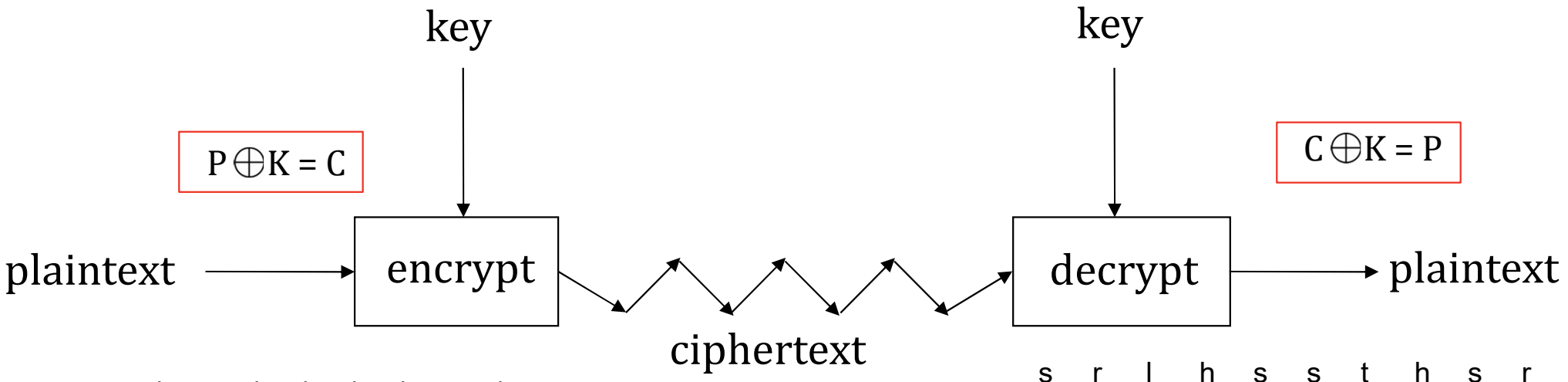
e=000 h=001 i=010 k=011 l=100 r=101 s=110 t=111

Decryption: Ciphertext⊕Key = Plaintext

$C \oplus K = P$ Or, in some texts $d_k(c) = k \oplus c$

	s	r	l	h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
Key:	111	101	110	101	111	100	000	101	110	000
Plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

One-Time Pad: The Whole Process



	h	e	i	l	h	i	t	l	e	r
Plaintext:	001	000	010	100	001	010	111	100	000	101
Key:	111	101	110	101	111	100	000	101	110	000
Ciphertext:	110	101	100	001	110	110	111	001	110	101
	s	r	l	h	s	s	t	h	s	r

	s	r	l	h	s	s	t	h	s	r
Ciphertext:	110	101	100	001	110	110	111	001	110	101
Key:	111	101	110	101	111	100	000	101	110	000
Plaintext:	001	000	010	100	001	010	111	100	000	101
	h	e	i	l	h	i	t	l	e	r

- One-time pad as an example symmetric key crypto
- Key is as large as plaintext
- In fact, size of plaintext= size of key= size of ciphertext

One-Time Pad: Issues

- The problem with the one-time pad is that, in order to create such a cipher, its key should be as long or even longer than the plaintext. In other words, if you have 500 MegaByte video file that you would like to encrypt, you would need a key that's at least 4 Gigabits long.

- Clearly, while Top Secret information or matters of national security may warrant the use of a one-time pad, such a cipher would just be too impractical for day-to-day public use.

Reference: http://www.jscape.com/blog/stream-cipher-vs-block-cipher

One-Time Pad: Issues with Plaintext attack

The cipher

$$e_k(m) = k \oplus m \qquad \text{and} \qquad d_k(c) = k \oplus c$$

defined by XOR of bit strings is not secure against a known plaintext attack. Demonstrate your attack by finding the private key used to encrypt the 16-bit ciphertext $c = 1001010001010111$ if you know that the corresponding plaintext is $m = 0010010000101100$.

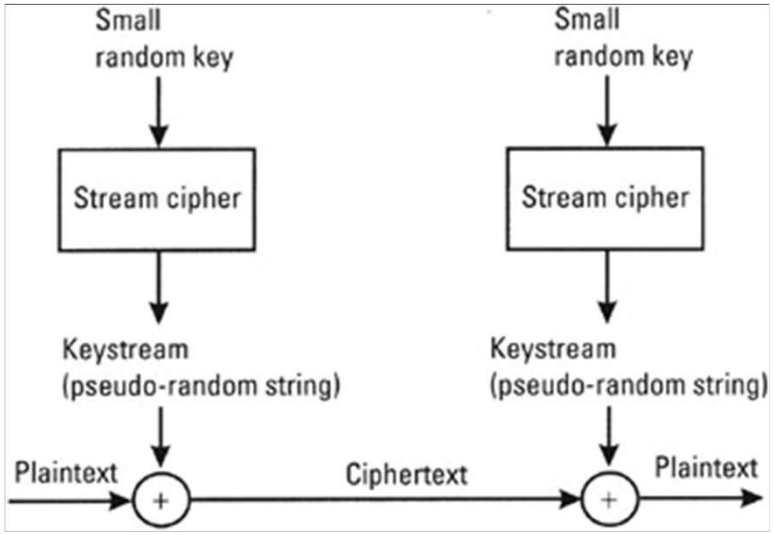
If you know m and c, since they are related by $c = k \oplus m$, it follows that $c \oplus m = k \oplus m \oplus m = k$. For the example,

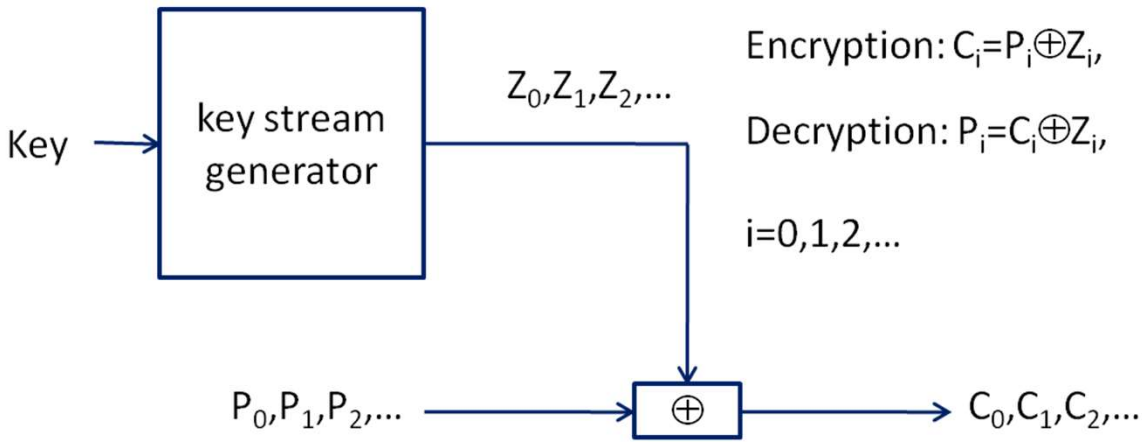
$$k = c \oplus m = 1001010001010111 \oplus 0010010000101100 = \boxed{1011000001111011}.$$

One-Time Pad Summary

- **Provably** secure
  - Ciphertext gives **no** useful info about plaintext
  - All plaintexts are ***equally likely***
- The One-Time Pad, which is supposed to employ a purely random key, can potentially achieve "perfect secrecy". That is, it's supposed to be fully immune to brute force attacks.
- BUT, only when be used correctly
  - Pad must be random, used only once
  - Pad is known only to sender and receiver
- Note: pad (key) is same size as message

Stream Ciphers

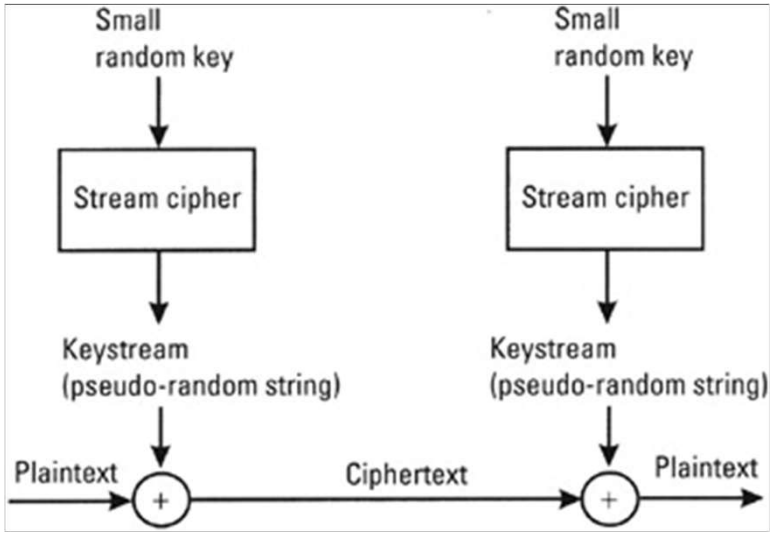


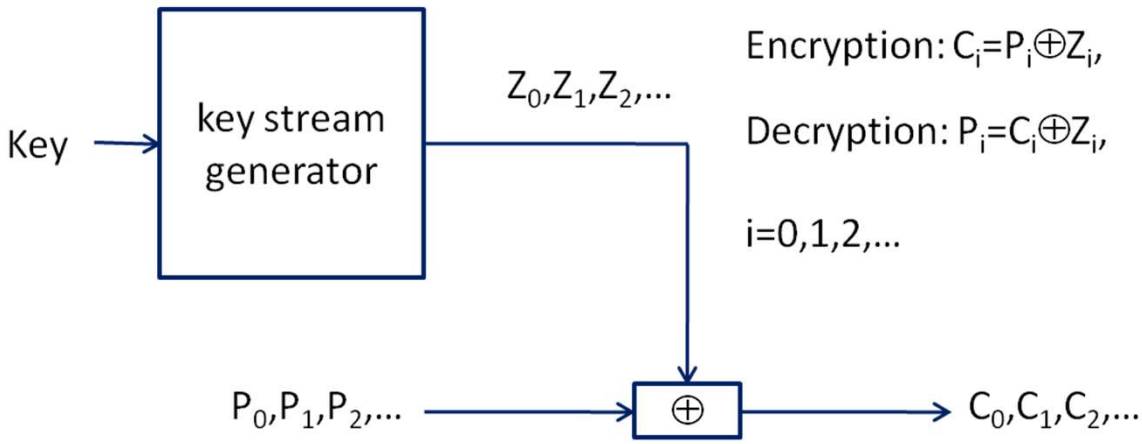


- A stream cipher is an encryption algorithm that encrypts 1 bit of plaintext at a time.
- Stream ciphers are designed to approximate an idealized cipher -One-Time Pad.
  - Except that key is relatively short
  - Key is stretched into a long **keystream**
  - keystream is used just like a one-time pad

Reference: http://www.jscape.com/blog/stream-cipher-vs-block-cipher

Stream Ciphers





- It uses an infinite stream of pseudorandom bits as the key.
- For a stream cipher implementation to remain secure, its pseudorandom generator should be unpredictable and the key should never be reused.
- The key of a stream cipher is no longer as long as the original message. Hence, it can no longer guarantee "perfect secrecy". However, it can still achieve a strong level of security.

http://www.jscape.com/blog/stream-cipher-vs-block-cipher

Usage and Examples of Stream Ciphers

- Stream ciphers are often used for their speed and simplicity implementation in hardware, and in applications where plaintext comes in quantities of unknowable length like a secure wireless connection (e.g. *older technology* GSM mobile phones)

- advantage of stream ciphers in military cryptography is that the cipher stream can be generated in a separate box that is subject to strict security measures and fed to other devices such as a radio set, which will perform the XOR operation as part of their function. The latter device can then be designed and used in less stringent environments.

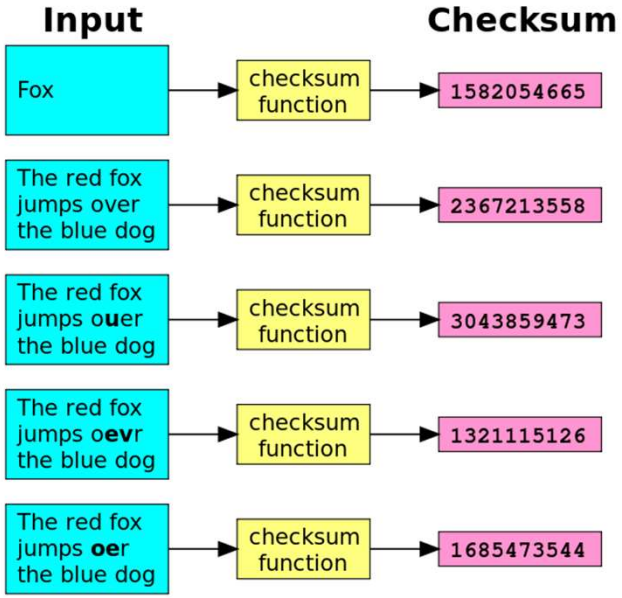
- Examples: A5/1, A5/2, RC4 etc.

Reference: https://en.wikipedia.org/wiki/Stream_cipher

Segment 2: Cryptographic Hash Functions

What is Cryptographic Hash

- A "hash" (also called a "digest", and informally a "checksum") is a kind of "signature" for a stream of data that represents the contents.
- The closest real-life analog we can think is "a tamper-evident seal on a software package": if you open the box (change the file), it's detected.
- MD5 was the most well known hash function.
- But MD5 is now outdated!!
- The SHA-2 family with hash values of 224, 256, 384 or 512 bits: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**



What is Cryptographic Hash

- Application of popular "MD5" method to streams of data yields a fixed, 128-bit number that summarizes that stream
- *all* input streams yield hashes of the same length.

```
$ cat smallfile
This is a very small file with a few characters

$ cat bigfile
This is a larger file that contains more characters.
This demonstrates that no matter how big the input
stream is, the generated hash is the same size (but
of course, not the same value). If two files have
a different hash, they surely contain different data.

$ ls -l empty-file smallfile bigfile linux-kernel
-rw-rw-r--    1 steve    steve           0 2004-08-20 08:58 empty-file
-rw-rw-r--    1 steve    steve          48 2004-08-20 08:48 smallfile
-rw-rw-r--    1 steve    steve         260 2004-08-20 08:48 bigfile
-rw-r--r--    1 root     root      1122363 2003-02-27 07:12 linux-kernel

$ md5sum empty-file smallfile bigfile linux-kernel
d41d8cd98f00b204e9800998ecf8427e  empty-file
75cdbfeb70a06d42210938da88c42991  smallfile
6e0b7a1676ec0279139b3f39bd65e41a  bigfile
c74c812e4d2839fa9acf0aa0c915e022  linux-kernel
```

What is Cryptographic Hash

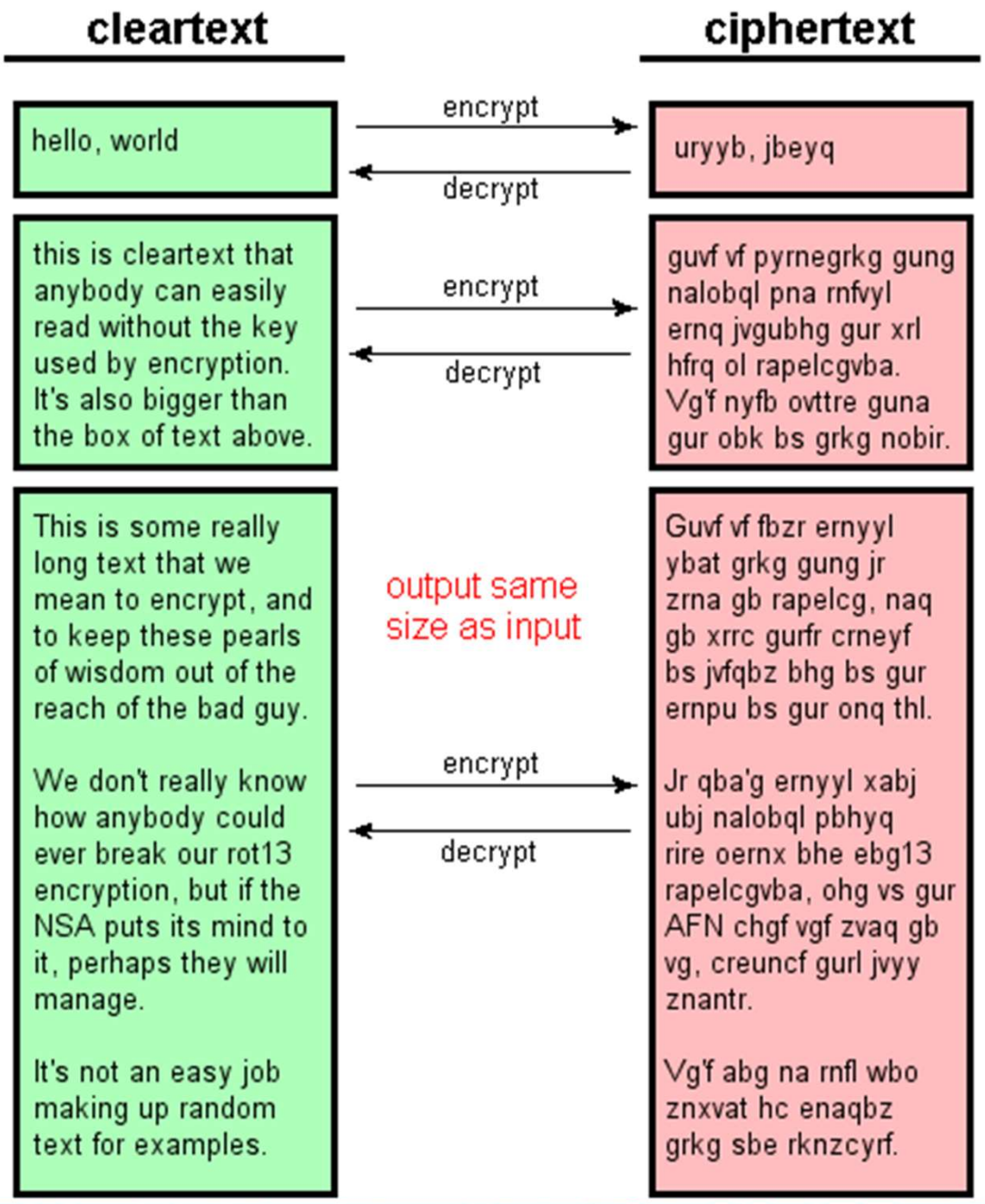
- If we try changing just one character of a small test file, you'll find that even very small changes to the input yields sweeping changes in the value of the hash, and this is known as the avalanche effect.
- The avalanche effect can be best seen by hashing two files with nearly identical content. We've changed the first character of a file from **T** to **t**, and when looking at the binary values for these ASCII characters, we see that they differ by just one bit

```
$ cat file1
This is a very small file with a few characters

$ cat file2
this is a very small file with a few characters

$ md5sum file?
75cdbfeb70a06d42210938da88c42991  file1
6fbe37f1eea0f802bd792ea885cd03e2  file2
```

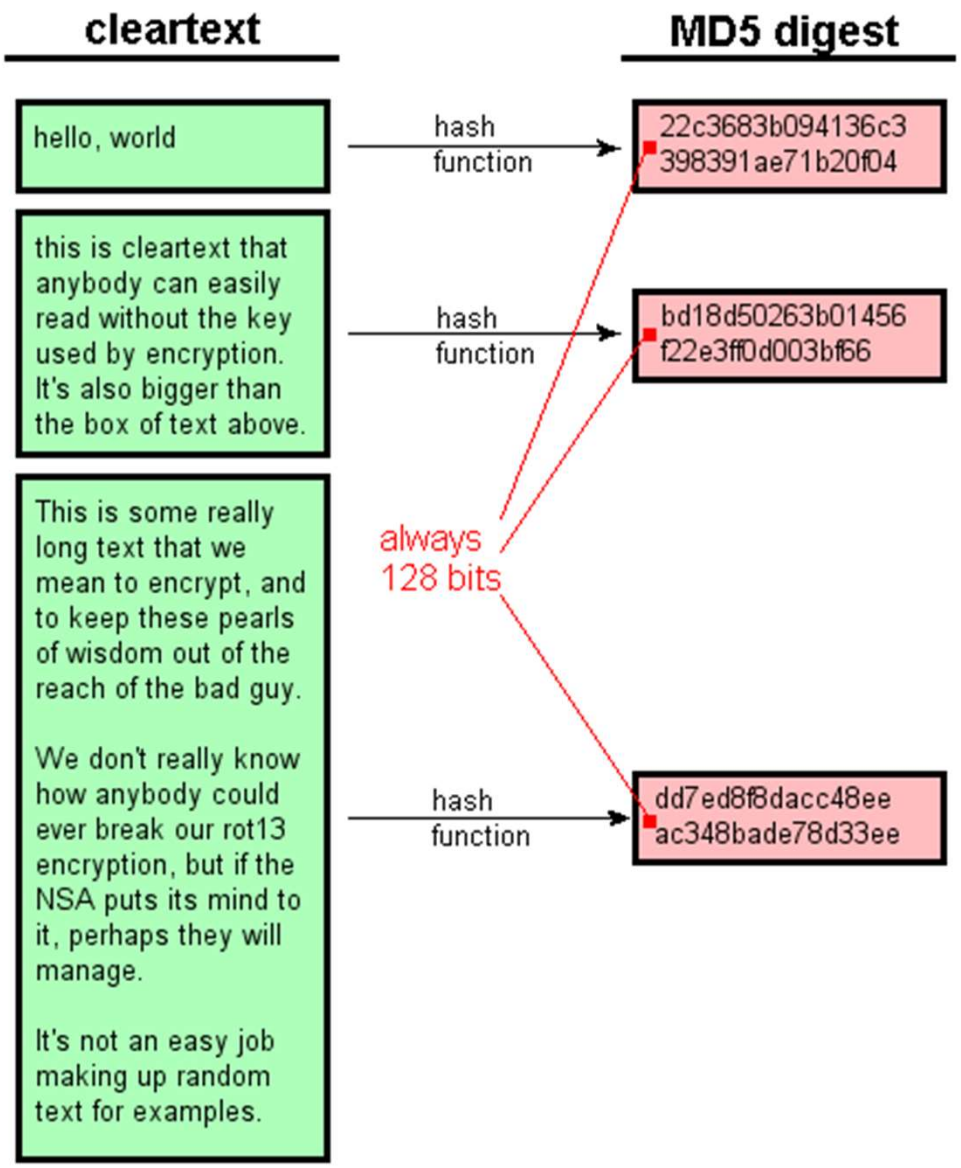
Encryption Vs Hash (IMPORTANT)



Encryption transforms data from a cleartext to ciphertext **and back** (given the right keys), and the two texts should roughly correspond to each other in size: big cleartext yields big ciphertext, and so on. "Encryption" is a **two-way** operation.

This is a common confusion, especially because all these words are in the category of "cryptography", but it's important to understand the difference.

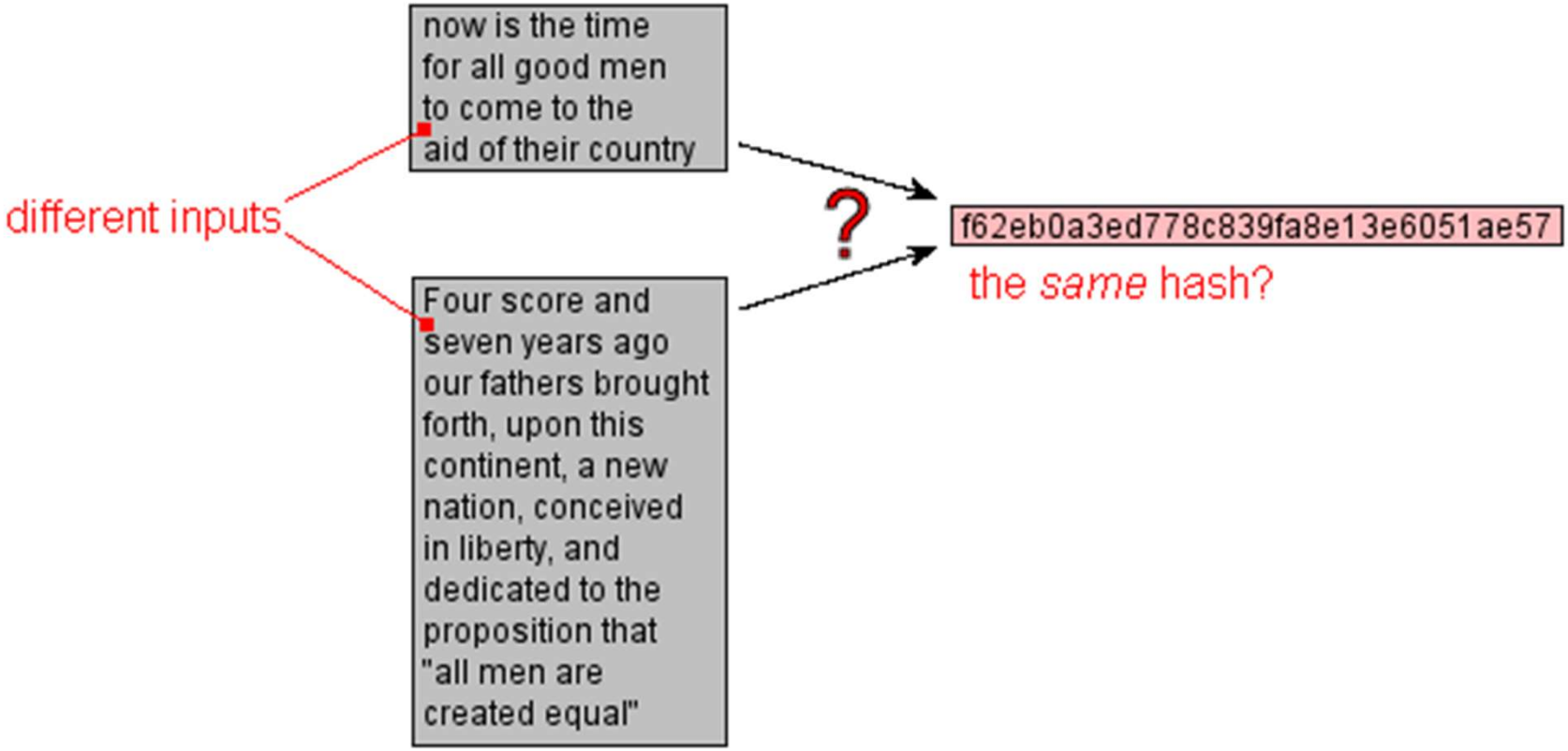
Encryption Vs Hash (IMPORTANT)



- Hashes compile a stream of data into a small digest, and it's strictly a one way operation.
- All hashes of the same type - this example shows the "MD5" variety - have the same size no matter how big the inputs are.

Hash Collision

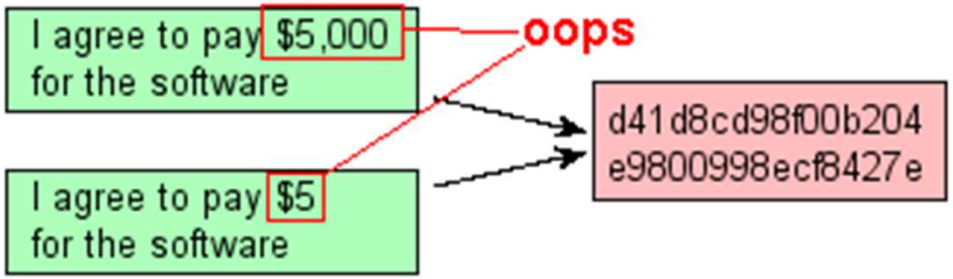
When different chunks of data produce the same hash value, this is known as a **collision**.



Hypothetical example of Hash Collision

Hash Collision: Applications Fail

- Many applications depend on the correctness of hashing. They would fail.
- If we are able to create two inputs that generate the same hash, digital signature become suspect. In our example below, the document signed was "a promise to pay", and being able to substitute one signed document for another would certainly lead to havoc



Properties of Crypto Hash Function

- Crypto hash function $h(x)$ must provide
  - **Compression** — output length is small
  - **Efficiency** — $h(x)$ easy to compute for any x
  - **One-way** — given a value y it is infeasible to find an x such that $h(x) = y$. In other words, it is infeasible to generate a message from its hash value
  - **Weak collision resistance** — given x and $h(x)$, infeasible to find $y \neq x$ such that $h(y) = h(x)$
  - **Strong collision resistance** — infeasible to find ***any*** x and y, with $x \neq y$ such that $h(x) = h(y)$. In other words, it is infeasible to find two different messages with the same hash value

- Lots of collisions exist, but hard to find ***any***

Applications of Hash

- Authentication (HMAC)
- Message integrity (HMAC)
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- Anything you can do with symmetric crypto
- Also, many, many clever/surprising uses…

Applications of Hashes: Data integrity

- ProFTPD -

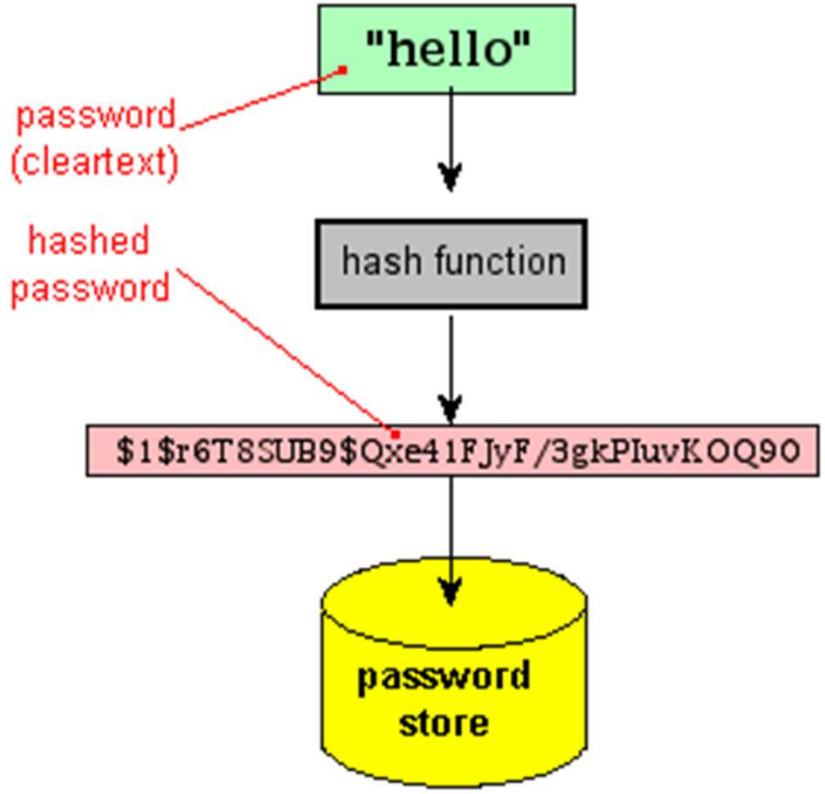
Highly configurable GPL-licensed FTP server software

MD5 sums and PGP signatures of release files

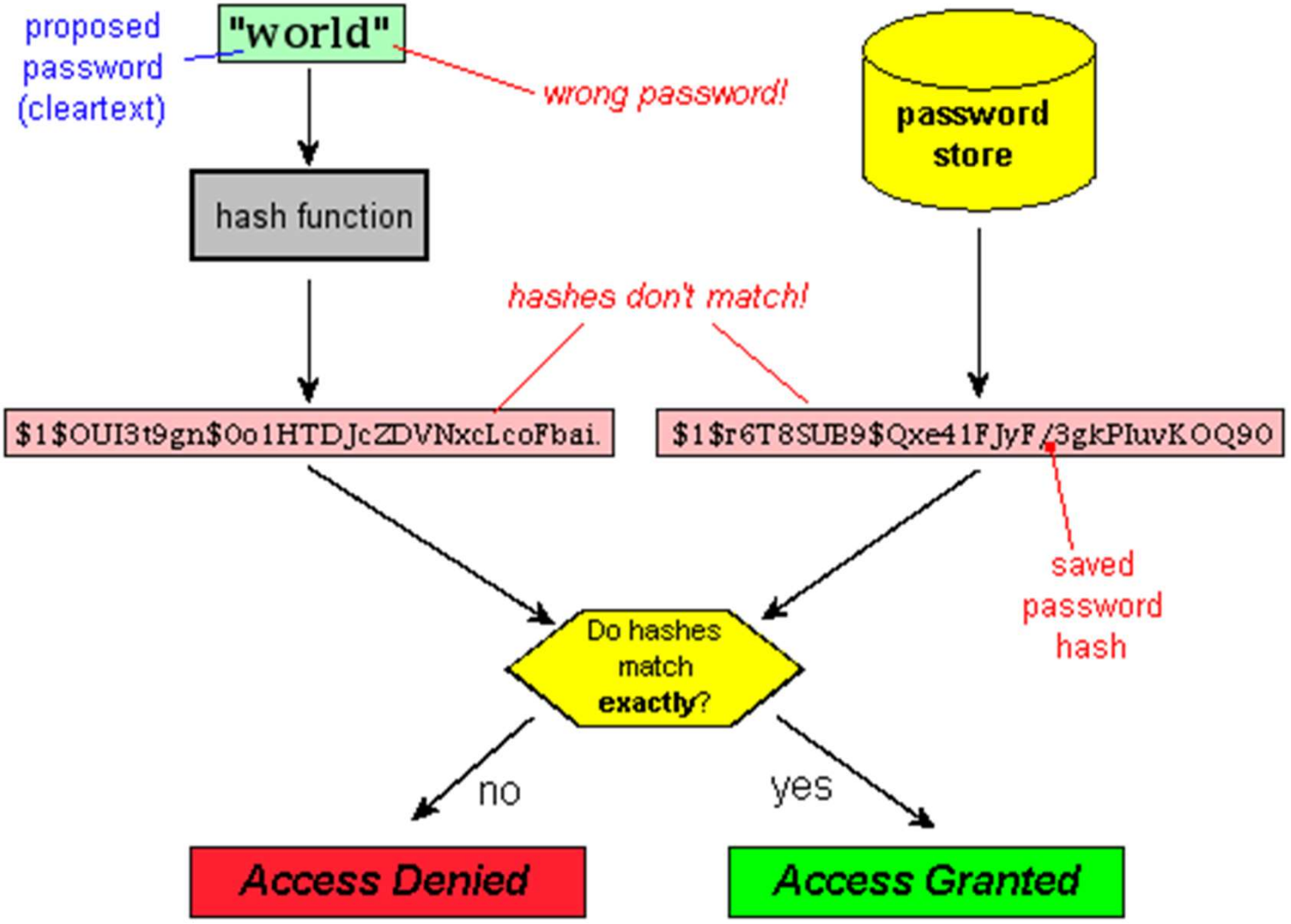
MD5 Digest Hashes

```
417e41092610816bd203c3766e96f23b proftpd-1.2.8p.tar.bz2
abf8409bbd9150494bc1847ace06857a proftpd-1.2.8p.tar.gz
7c85503b160a36a96594ef75f3180a07 proftpd-1.2.9.tar.bz2
445fbf24e2ec300800a184eb81296bda proftpd-1.2.9.tar.gz
d834bb822816a2ce483cc2ef1a9533e7 proftpd-1.2.10rc3.tar.bz2
1e306d2f54ea92895ecff6659498b911 proftpd-1.2.10rc3.tar.gz
```

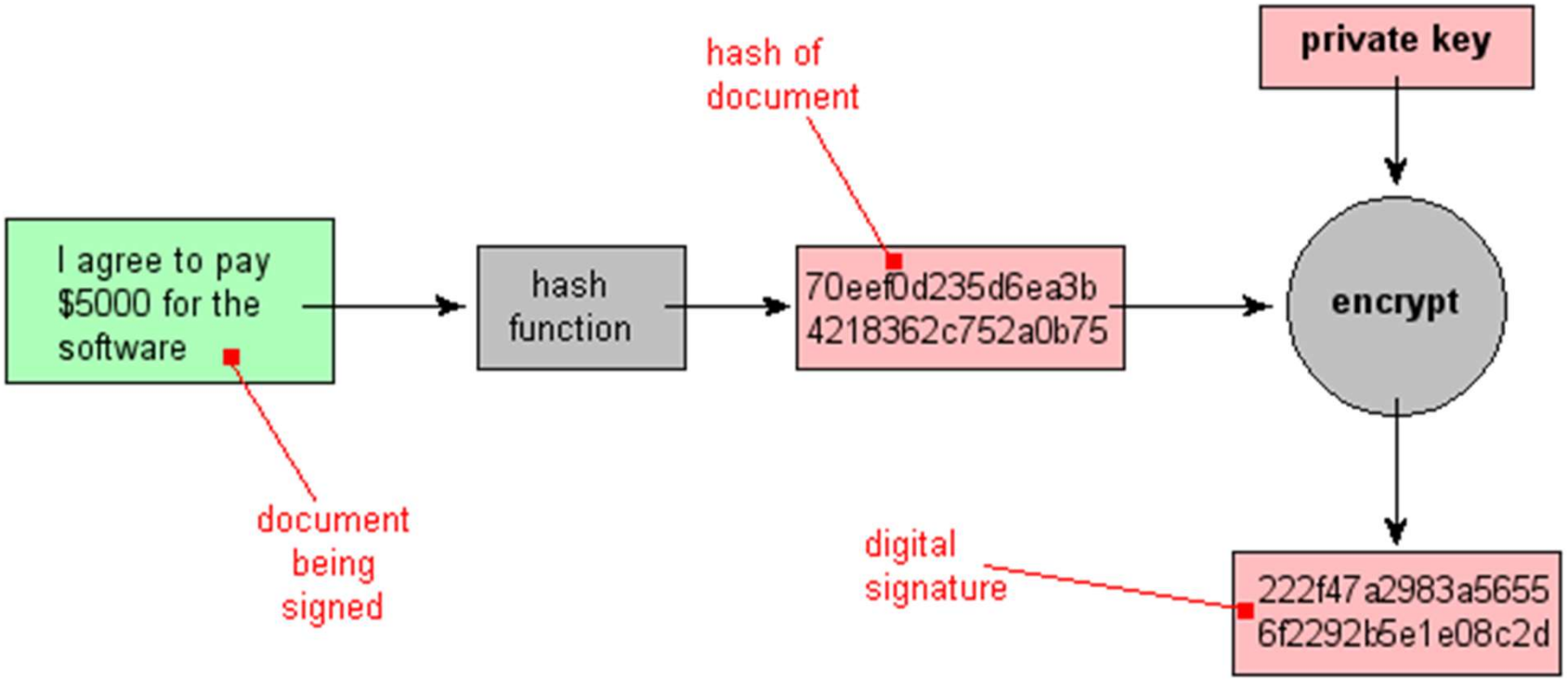
Applications of Hashes: Passwords



Testing a proposed password against the stored hash



Applications of Hashes: Digitally Signing Documents



Conclusions

- One big issue with using symmetric algorithms is the key exchange problem, which can present a classic catch-22.
- The other main issue is the problem of trust between two parties that share a secret symmetric key. Ensuring the integrity of received data and verifying the identity of the source of that data can be very important.
- The key exchange problem arises from the fact that communicating parties must somehow share a secret key before any secure communication can be initiated, and both parties must then ensure that the key remains secret. Of course, direct key exchange is not always feasible due to risk, inconvenience, and cost factors.
- Fortunately, asymmetric algorithms can be used to solve these problems by performing the same basic operations which we will learn in the forthcoming lectorial(s).

Reference: http://www.informit.com/articles/article.aspx?p=102212

Popular choices of hashing algorithms in 2024

- Python (hashlib module, SHA family)
- Node and other JS frameworks
  - Argon2
  - scrypt
  - bcrypt