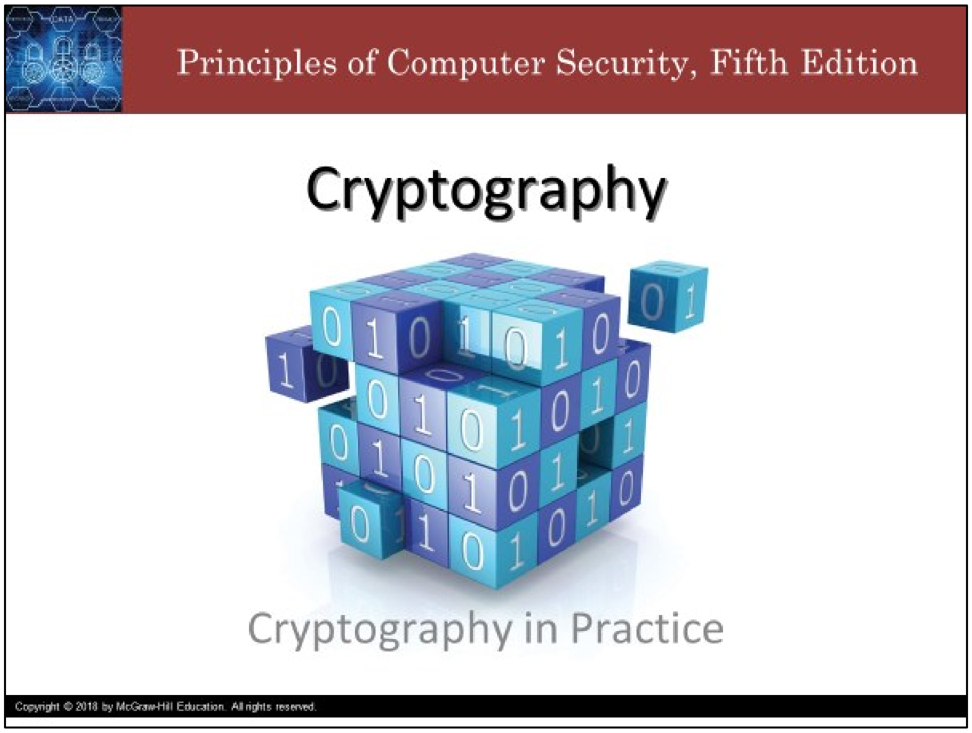
Cryptography: Cryptography in Practice

Slide 1



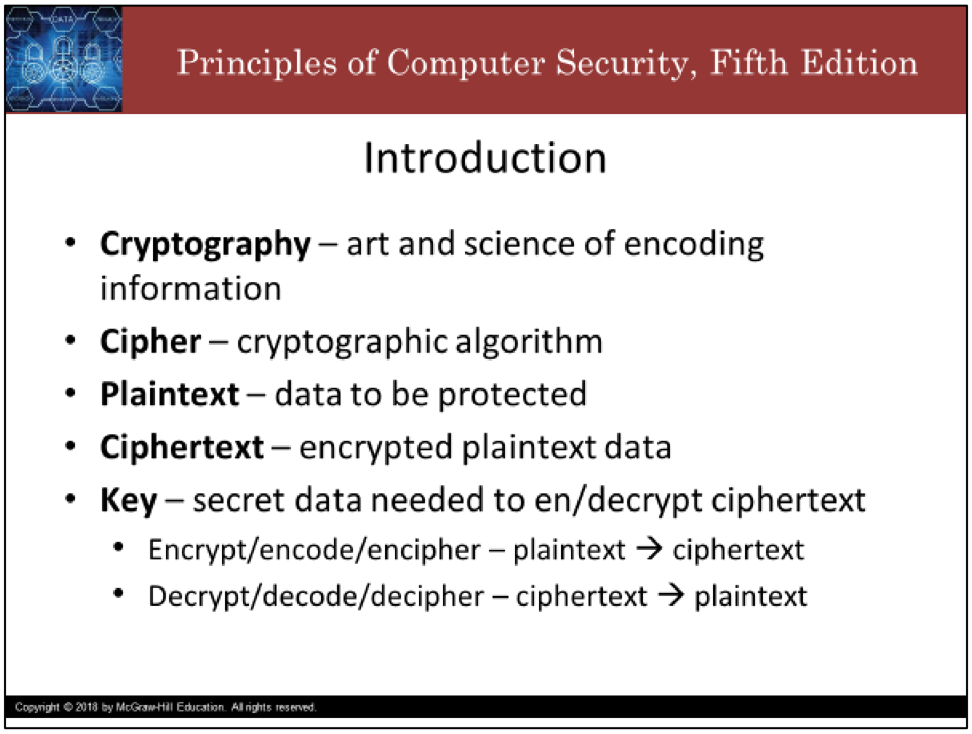
Principles of Computer Security, Fifth Edition

Cryptography

Cryptography in Practice

Howdy! In this video, we discuss cryptography in practice.

Slide 2



Principles of Computer Security, Fifth Edition

Introduction

- **Cryptography** – art and science of encoding information
- **Cipher** – cryptographic algorithm
- **Plaintext** – data to be protected
- **Ciphertext** – encrypted plaintext data
- **Key** – secret data needed to en/decrypt ciphertext
  - Encrypt/encode/encipher – plaintext → ciphertext
  - Decrypt/decode/decipher – ciphertext → plaintext

In the beginning, people in power attempted to hide and control information (and therefore power) by controlling who was taught to read and write.

Eventually, more complicated methods of concealing information by substituting and transposing symbols to make the text unreadable were developed.

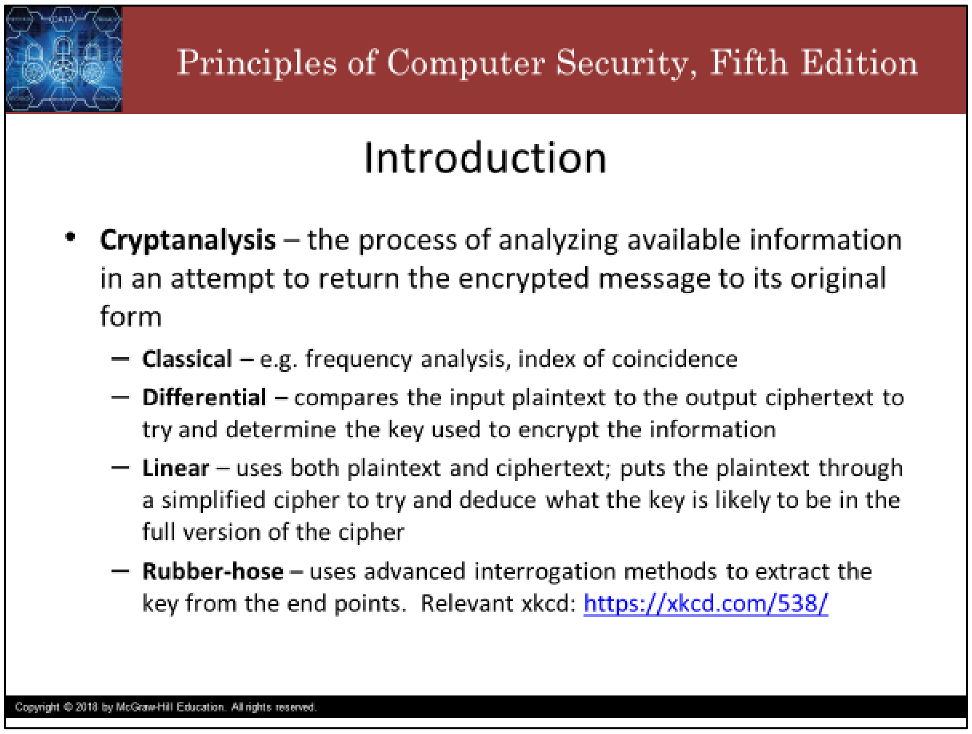
These complicated methods are called cryptographic algorithms, also known as *ciphers*. The word cipher comes from the Arabic word *sifr*, meaning zero.

The data that is being hidden, or protected, is the plaintext.

The encrypted plaintext is called the ciphertext.

The cipher uses a key for encryption and decryption. When following the principle of open design, the key is the only aspect of the cipher which is kept secret.

Slide 3


Principles of Computer Security, Fifth Edition

Introduction

- **Cryptanalysis** – the process of analyzing available information in an attempt to return the encrypted message to its original form
  - **Classical** – e.g. frequency analysis, index of coincidence
  - **Differential** – compares the input plaintext to the output ciphertext to try and determine the key used to encrypt the information
  - **Linear** – uses both plaintext and ciphertext; puts the plaintext through a simplified cipher to try and deduce what the key is likely to be in the full version of the cipher
  - **Rubber-hose** – uses advanced interrogation methods to extract the key from the end points. Relevant xkcd: https://xkcd.com/538/

If you build it, they will break it. And hopefully, they will write a report and help you rebuild it stronger than before.

The process of breaking an encryption algorithm (which means being able to decrypt ciphertext without knowning the key or discovering the key by interrogating the ciphertext and the algorithm) is called cryptanalysis.

Cryptanalysis may be thought of as the most important part of cryptology, the study of cryptography.

We cannot know which cipher is better unless we know which cipher is harder to break.

The field of cryptanalysis is very rich, with lots of history.

Arab scholars were probably the first to write about in the 9th century. The English word "cryptanalysis" wasn't invented until 1920.

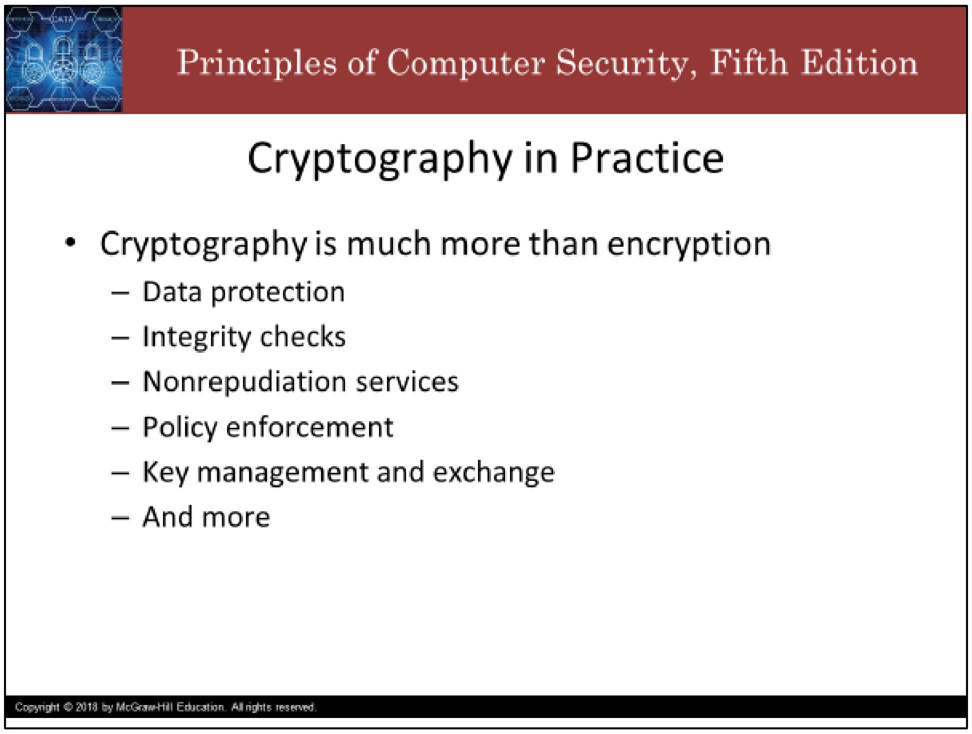
The basic classical cryptanalytic methods include things like frequency analysis, which uses the frequency of symbols or groups of symbols to aid in breaking the cipher.

Modern cryptanalysis is a very deep and complex field.

Two of the most widely used attacks on symmetric block ciphers are linear and differential cryptanalysis. They both use full knowledge of the cipher (following the principle of open design) to conduct known- or chosen-plaintext attacks with the goal of discovering the relationship between the input (the plaintext and the key) and the output (the ciphertext). The more that relationship is understood, the more advantage the cryptanalysis has. When successful, the result is the complete recovery of the key and, therefore, the ability to decrypt all ciphertexts which use that key.

There are many other types of attacks, including side-channel attacks, which attack the weaker links in cryptography like the implementation of the cipher or the people who use it. One particularly disturbing side-channel attack is rubber-hose cryptanalysis, in which a person is compelled to reveal secret data by force (typically physical, but could also be mental, emotional, or legal force, although legal force must devolve into more physical force if the target is unwilling to cooperate, but even then the target may still be unwilling or even unable to comply).

Slide 4



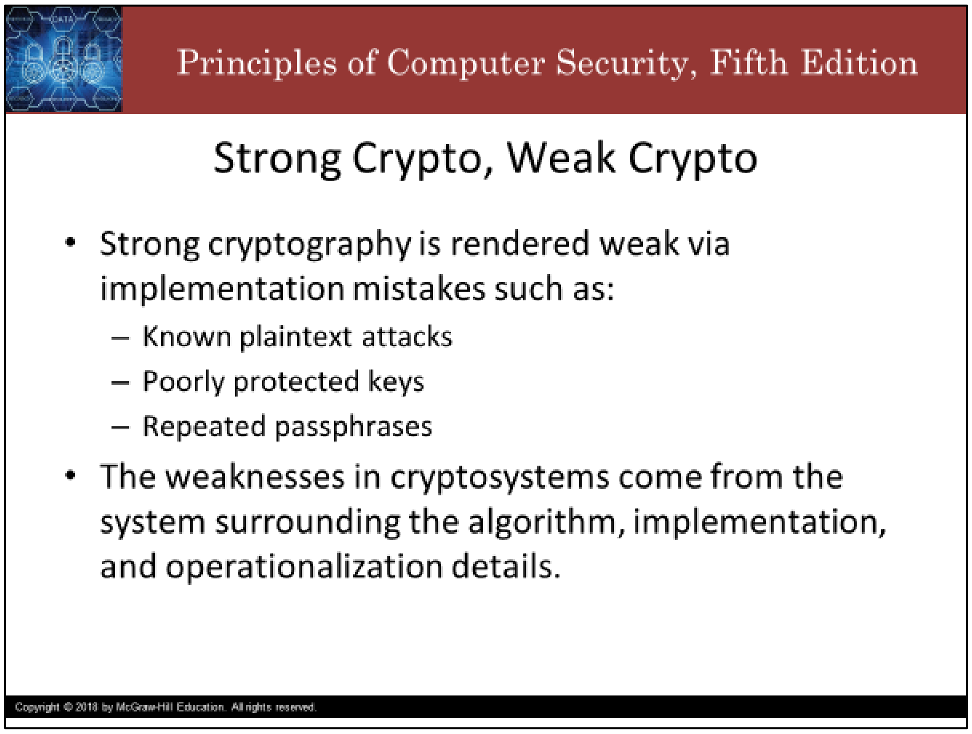
Principles of Computer Security, Fifth Edition

Cryptography in Practice

- Cryptography is much more than encryption
  - Data protection
  - Integrity checks
  - Nonrepudiation services
  - Policy enforcement
  - Key management and exchange
  - And more

Cryptography is much more than encryption for the purposes of protecting data. It is also used for Integrity checks, Nonrepudiation services, Policy enforcement, Key management and exchange, and much more.

Slide 5


Principles of Computer Security, Fifth Edition

Strong Crypto, Weak Crypto

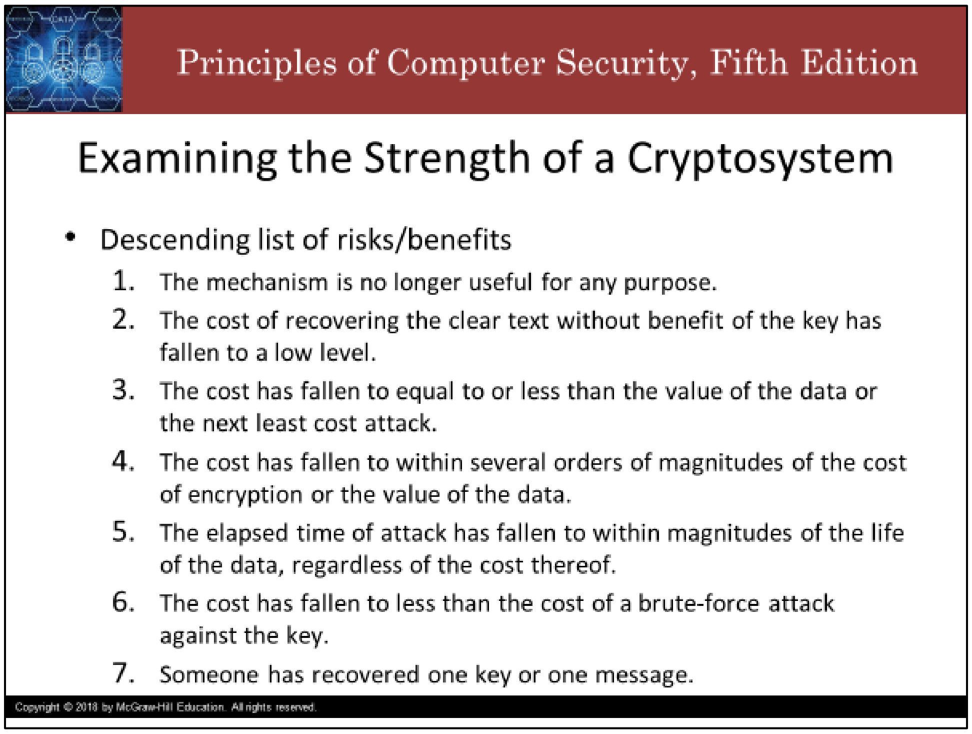
- Strong cryptography is rendered weak via implementation mistakes such as:
  - Known plaintext attacks
  - Poorly protected keys
  - Repeated passphrases
- The weaknesses in cryptosystems come from the system surrounding the algorithm, implementation, and operationalization details.

Modern cryptographic algorithms are subjected to very intense and sustained cryptanalysis. When a successful attack is discovered, the algorithm must either be improved or retired. This is the case for the predecessor to AES, which was DES, the Data Encryption Standard. Both linear and differential cryptanalysis are successful against DES. In response, several new ciphers were developed as possible replacements, as well as an adaption of DES, which applied the algorithm three times with three different keys, called triple-DES. Ultimately, DES was officially retired (although it is still in use by some legacy systems in the form of 3DES), and the AES has replaced it.

Strong cryptographic algorithms are those which do not have any publicly-known attacks which offer sufficient advantage to an attacker. However, that does not mean that strong crypto is unassailable and unconditionally secure; actually, far from it. The weakest link in any system is where theory meets reality. Implementation and usage errors can completely undermine whatever theoretical security a cipher has.

One example of this, and why many security researchers, especially cryptographers, are wary of working with the NSA (and some flat-out refuse to do so), is that the NSA is known to aid in the development of cryptographic algorithms for their own benefit. They did this with DES. The DES standard was published in 1977 and was developed at IBM. Differential cryptanalysis was discovered in the late 1980s and applied to DES, which was surprisingly resistant. The advantage that differential cryptanalysis has over brute force is a reduction in the keyspace of a factor of 512, but that still leaves a space of 2^47, which for a long time was unreasonable due to the state of computing. In the 1990s, with additional study, people got suspicious of the design of DES and the involvement of the NSA in its design and selection (here, I am inaccurate. DES was known to be insecure due to its small keyspace from the start, but the argument was that no one would bother to build such powerful machines despite their technical feasibility). Indeed, the people were right to be suspicious. The NSA and IBM knew of differential cryptanalysis in the 70s, and DES was designed to be resistant to it. The NSA asked IBM to keep this a secret. This might sound OK; the NSA helped make it stronger. Except, the changes they made to increase defense against differential cryptanalysis weakened it against brute force by drastically reducing the key size. So, for a period of about 15 years, the NSA was able to brute force DES ciphertexts in secret (because the unreasonable cost of building a machine capable of brute-forcing DES keys was not unreasonable to the NSA). Then, between the 1990s and when AES finally replaced DES in 2002, and the DES standard was withdrawn in 2005, the NSA and many others could brute force it mostly in secret, but the crypto community at large knew about it. Today, everyone should know that DES is too weak to be used for anything more than hobby or educational projects.

Slide 6



Principles of Computer Security, Fifth Edition

Examining the Strength of a Cryptosystem

- Descending list of risks/benefits
  1. The mechanism is no longer useful for any purpose.
  2. The cost of recovering the clear text without benefit of the key has fallen to a low level.
  3. The cost has fallen to equal to or less than the value of the data or the next least cost attack.
  4. The cost has fallen to within several orders of magnitudes of the cost of encryption or the value of the data.
  5. The elapsed time of attack has fallen to within magnitudes of the life of the data, regardless of the cost thereof.
  6. The cost has fallen to less than the cost of a brute-force attack against the key.
  7. Someone has recovered one key or one message.

When you are examining the strength of a cryptosystem, you should consider the following list of conditions, which are listed in decreasing order of risk:

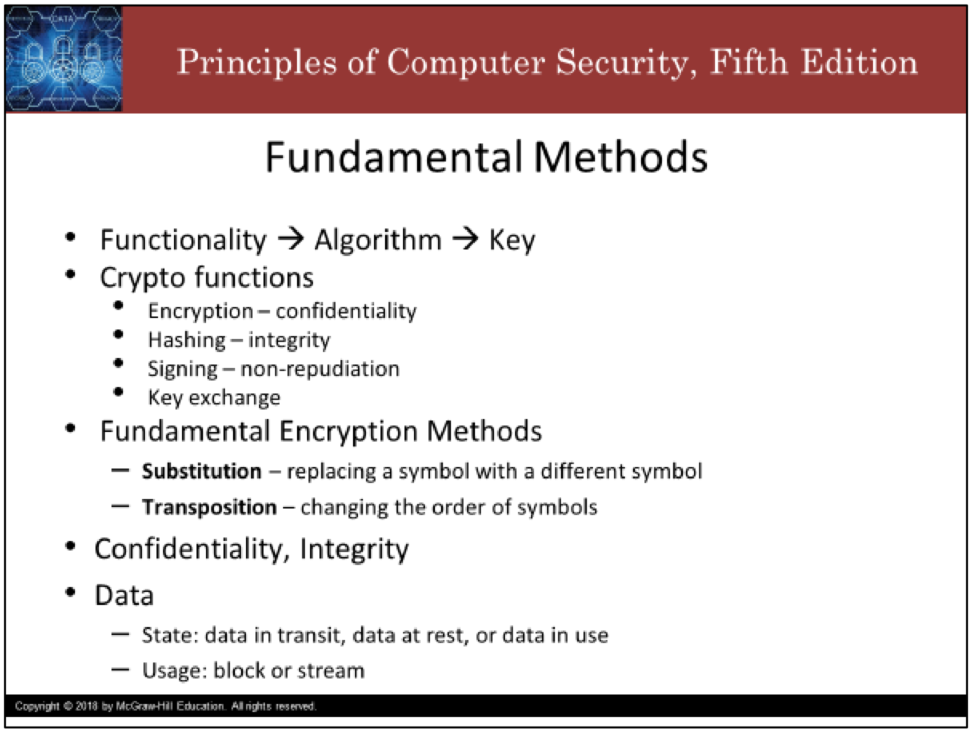
1. The mechanism is no longer useful for any purpose.
2. The cost of recovering the clear text without benefit of the key has fallen to a low level.
3. The cost has fallen to equal to or less than the value of the data or the next least-cost attack.
4. The cost has fallen to within several orders of magnitudes of the cost of encryption or the value of the data.
5. The elapsed time of attack has fallen to within magnitudes of the life of the data, regardless of the cost thereof.

6. The cost has fallen to less than the cost of a brute-force attack against the key

7. Someone has recovered one key or one message.

If any of the first three conditions are true, the cryptosystem is too weak and needs to be replaced immediately. If the 4th condition is true, now is a good time to select a replacement and plan an orderly transition in the near future. The transition for conditions 1, 2, and 3 is one likely marked by panic (but do try to be calm and careful, as rushing can cause mistakes, and mistakes will undermine the value of the transition). Many organizations typically start planning replacement at conditions 5 or 6. Condition 7 is only of concern for those organizations with the highest standards of security, such as those that work with classified information.

Slide 7



Principles of Computer Security, Fifth Edition

Fundamental Methods

- Functionality → Algorithm → Key
- Crypto functions
  - Encryption – confidentiality
  - Hashing – integrity
  - Signing – non-repudiation
  - Key exchange
- Fundamental Encryption Methods
  - **Substitution** – replacing a symbol with a different symbol
  - **Transposition** – changing the order of symbols
- Confidentiality, Integrity
- Data
  - State: data in transit, data at rest, or data in use
  - Usage: block or stream

Modern cryptographic operations are performed using both an algorithm and a key.

The choice of algorithm depends on the type of functionality required.

The choice of key depends on the algorithm

Cryptographic functions include:

- Encryption for the protection of confidentiality,
- hashing (for the protection of integrity)
- digital signatures (to manage nonrepudiation),
- and other operations such as key exchange.

There are two fundamental operations used for encryption, which cover both classical and modern cryptography.

Substitution, which replaces one symbol with another, and

Transposition, which changes the order of symbols

Classical ciphers used simple applications of these operations, while modern cryptography is built around complex mathematical functions that perform the same operations.

You do not need to understand the intricate details of how a cryptosystem operates in order to use it effectively.

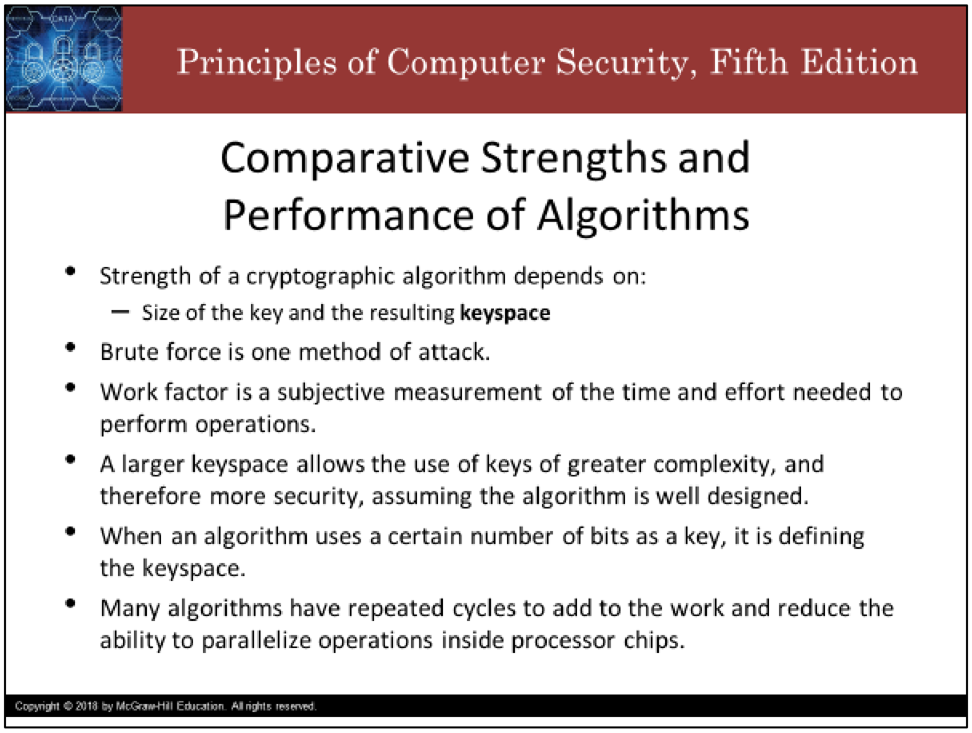
Cryptographic operations are characterized by the quantity and type of data, as well as the level and type of protection sought.

Confidentiality protection operations are characterized by protection against unauthorized disclosure, i.e., only someone with the key can read the data.

Integrity protection operations are characterized by protection against unauthorized modification, i.e., only someone with the key can create or modify the data.

Data can be characterized by its state (in transit, at rest, or in use) or by how it is used (as blocks of bits or as a stream of bits).

Slide 8



Principles of Computer Security, Fifth Edition

Comparative Strengths and Performance of Algorithms

- Strength of a cryptographic algorithm depends on:
  - Size of the key and the resulting **keyspace**
- Brute force is one method of attack.
- Work factor is a subjective measurement of the time and effort needed to perform operations.
- A larger keyspace allows the use of keys of greater complexity, and therefore more security, assuming the algorithm is well designed.
- When an algorithm uses a certain number of bits as a key, it is defining the keyspace.
- Many algorithms have repeated cycles to add to the work and reduce the ability to parallelize operations inside processor chips.

Several factors determine the strength of a cryptosystem.

Although you can't actually put them in order since it is the weakest one which determines the actual strength, a critical factor is the size of the key or the keyspace.

The larger the keyspace, the stronger the algorithm. A keyspace which is too small will leave the algorithm open to brute-force attack, where the attacker simply tries all possible keys.

All of the strength factors fall into the category of resistance to cryptanalysis.

If the algorithm is weak to even a single form of cryptanalysis, the algorithm is weak, period.

The guiding principle for assessing the strength of a cryptographic algorithm is work factor.

The more work, the attacker, has to do the break the code, the stronger it is considered to be.

Work factor is often expressed in terms of the keyspace or reductions to it.

A small keyspace results in low work factor, and thus a weak system.

Susceptibility to linear or differential cryptanalysis, which effectively reduces the keyspace, also lowers the work factor.

Side-channel attacks can also reduce the keyspace, sometimes to 0.

The size of the keyspace is measured in bits. A key with N bits usually corresponds to a keyspace of size 2^N. When the key is not a binary number, you just need to convert it to one. In a computer, all values are binary numbers, including text.

But, just because an algorithm uses a key of a certain size does not mean that the keyspace includes all of the possible keys. Triple-DES uses 3 DES keys, each of length 56 bits, for a total key length of 168 bits. But, the effective key length is only 112 bits, or 2 DES keys, due to a meet-in-the-middle attack.

Another aspect of the strength of cryptographic algorithms is performance.

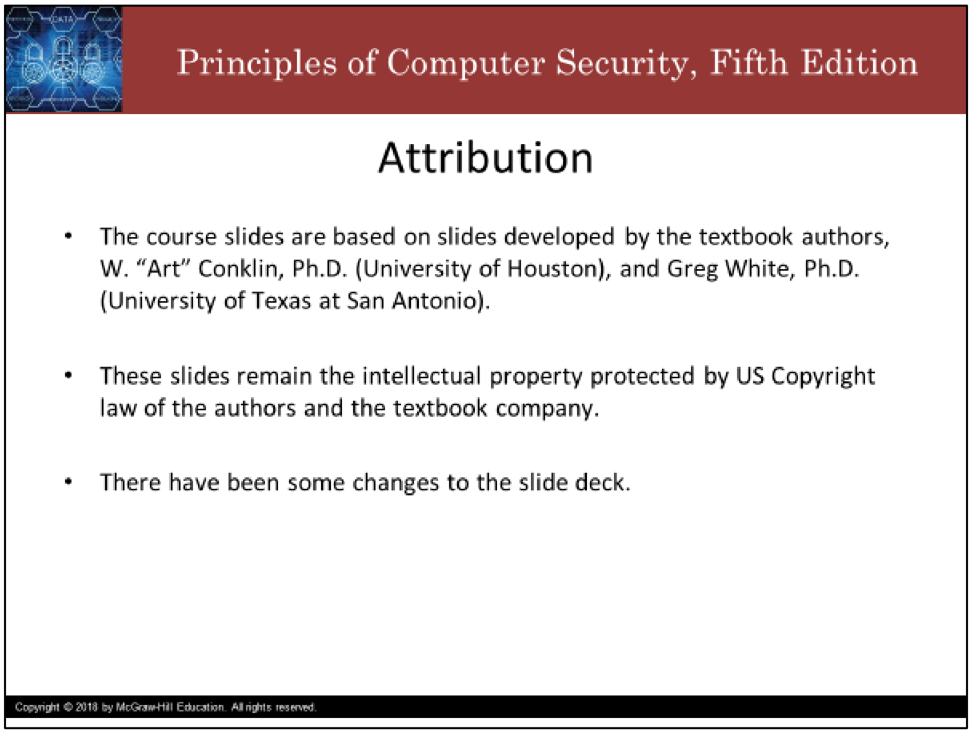
This is also an element that contributes to the work factor, but once which must be balanced between usage and defense.

Encryption, in order to be useful for many things, needs to be fast. But, the faster the algorithm can run, either in software or with hardware acceleration, the faster it can be brute-forced.

So, many modern cryptosystems are designed to use multiple cycles or rounds and other techniques for increasing the work factor just enough to make a brute force attack infeasible, but not so much that normal usage is impeded.

One common tactic for asymmetric cryptosystems is to make the encrypt and decrypt steps imbalanced so that one is much more expensive than the other. The one which the attacker must do a lot of, typically decryption, is made to be the slower operation.

Slide 9



Principles of Computer Security, Fifth Edition

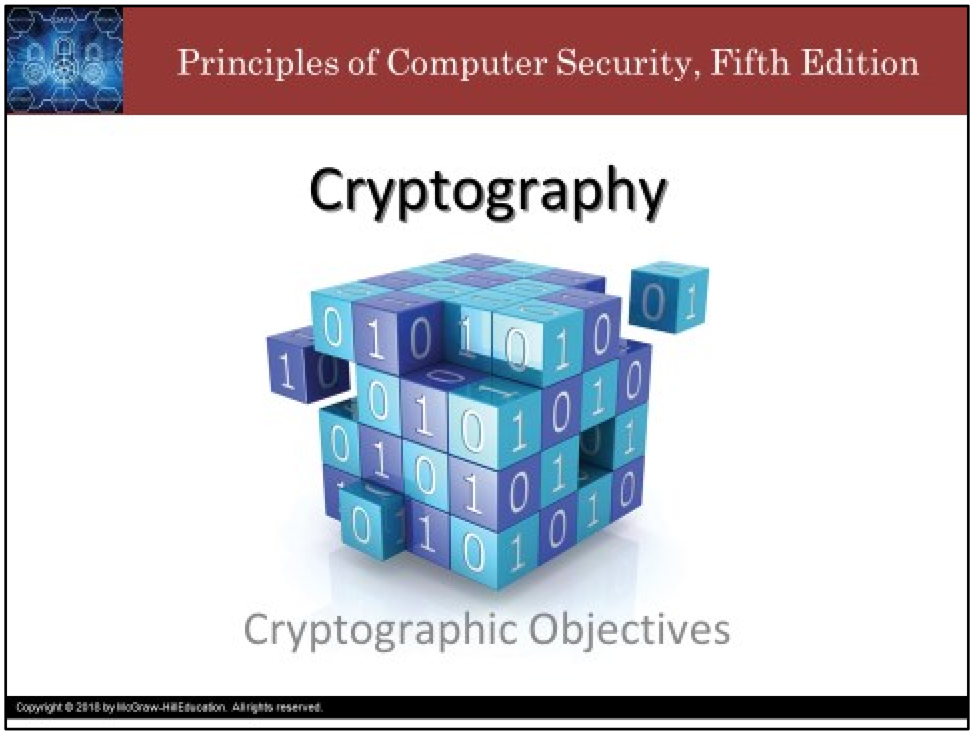
Attribution

- The course slides are based on slides developed by the textbook authors, W. “Art” Conklin, Ph.D. (University of Houston), and Greg White, Ph.D. (University of Texas at San Antonio).
- These slides remain the intellectual property protected by US Copyright law of the authors and the textbook company.
- There have been some changes to the slide deck.

Thank you and take care.

Cryptography: Cryptographic Objectives

Slide 1



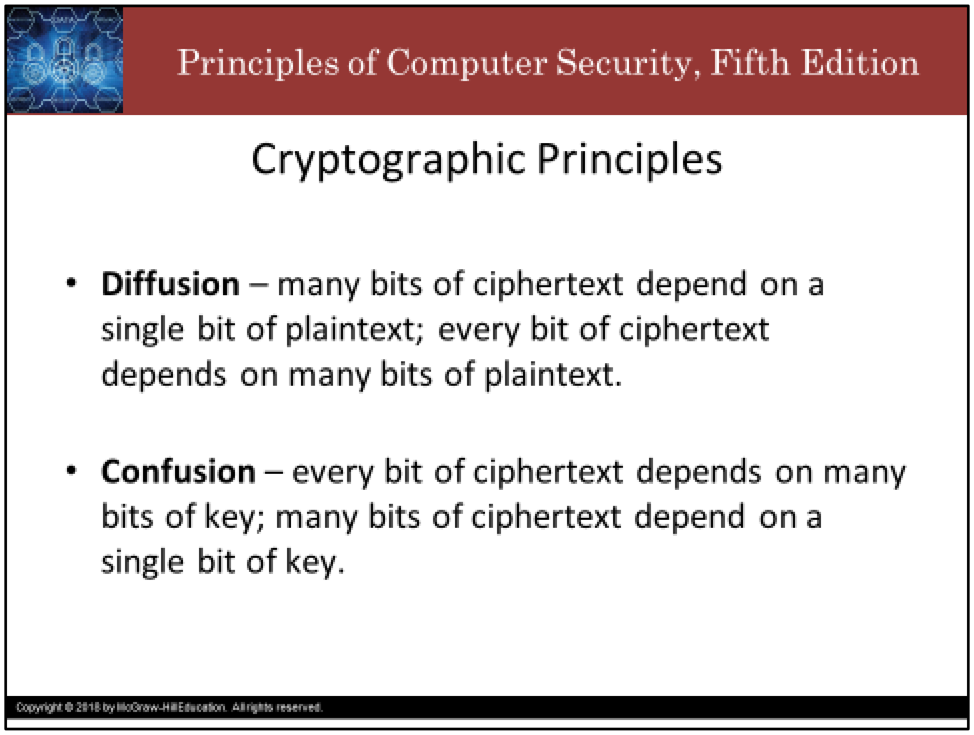
Principles of Computer Security, Fifth Edition

Cryptography

Cryptographic Objectives

Howdy! In this video, we introduce the principles of diffusion and confusion, as well as the objectives of obfuscation, perfect forward secrecy, and the role of obscurity.

Slide 2



Principles of Computer Security, Fifth Edition

Cryptographic Principles

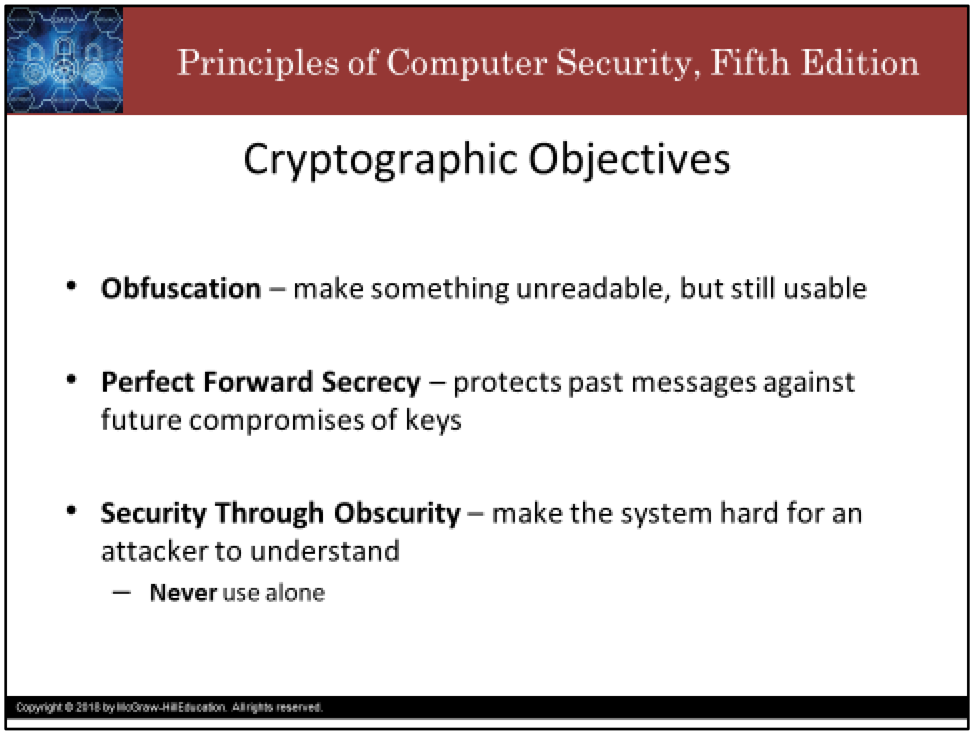
- **Diffusion** – many bits of ciphertext depend on a single bit of plaintext; every bit of ciphertext depends on many bits of plaintext.
- **Confusion** – every bit of ciphertext depends on many bits of key; many bits of ciphertext depend on a single bit of key.

In another video, we mentioned that cryptosystems use substitution and transposition to protect data through encryption. Those are general mechanisms which can be implemented concretely in several ways. More abstractly, the purpose of using techniques like substitution and transposition is to achieve what is known as confusion. In addition to confusion, cryptosystems also aim to achieve diffusion. Modern cryptosystems make use of both diffusion and confusion to achieve their security.

Diffusion is the principle that the plain text and the ciphertext should be structurally independent from the perspective of a statistical analysis; in other words, a change in one small part of the plain text should result in many changes in the ciphertext so that changes in the ciphertext do not reveal information as to the structure of the plain text if I changed one small part of the plain text and only one small part of the ciphertext. Changes that gives me information I can use to get an advantage over the cipher and help me break it. Conversely, every bit of ciphertext should depend on many bits of plain text. If you speak the language of mathematics, the desired relation between bits of plain text and bits of ciphertext is many to many.

Confusion is an analogous principle to diffusion, except it relates the ciphertext and the key; again, the desired relation between key and ciphertext is many to many. If I change a small part of the key and it only changes a small part of the ciphertext, this gives me an advantage which I can press to crack the code. What relation should exist between the plain text and the key? Pause now if you want to think about it.

Slide 3



Principles of Computer Security, Fifth Edition

Cryptographic Objectives

- **Obfuscation** – make something unreadable, but still usable
- **Perfect Forward Secrecy** – protects past messages against future compromises of keys
- **Security Through Obscurity** – make the system hard for an attacker to understand
  - **Never** use alone

Copyright © 2018 by McGraw-Hill Education. All rights reserved.

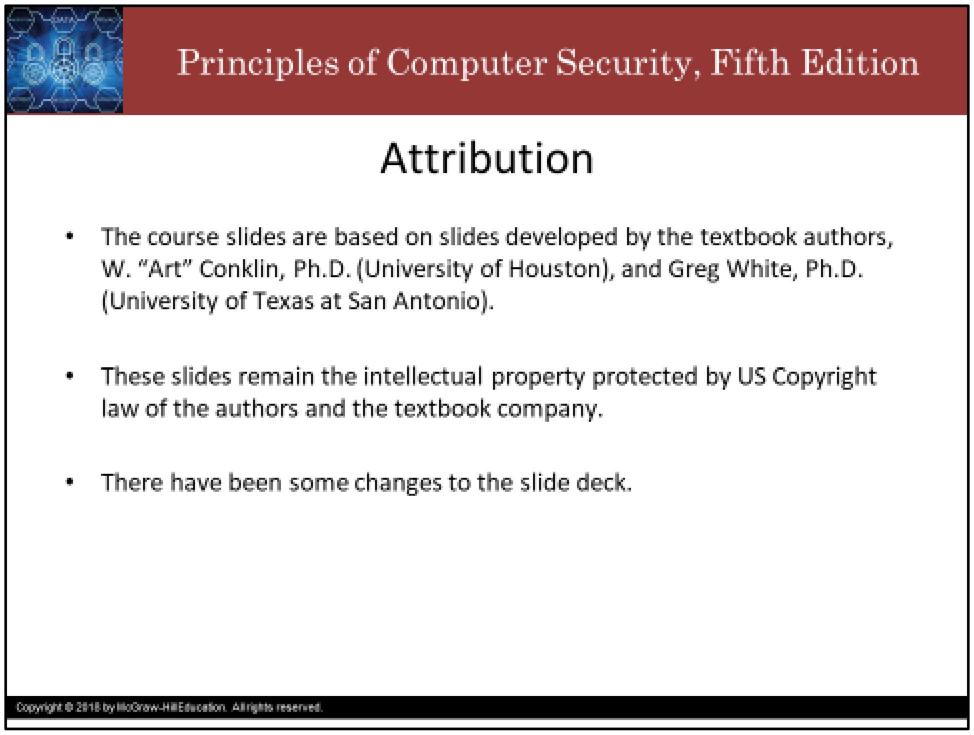
If you came up with none, that's right. The key in the plain text should be essentially independent. There are some ciphers in which there was a relationship, such as the one-time pad, which required that the length of the key be the same as the length of the plain text. But besides that, the plain text and the key should be completely independent. If they are not, then that means that the choice of plain text reduces the keyspace, which reduces the work factor, which weakens the system.

The purpose of cryptography is to protect security goals like confidentiality and integrity. I should say, instead of purpose, the primary use cases, since the purpose of anything built by humans is contingent upon how it is used. The purpose of a paper clip is to temporarily hold papers together until someone repurposes the paper clip to short circuit an electrical system, or pick a lock, or make art. One of the main objectives of cryptosystems is obfuscation, which is the masking of an item to render it unreadable, yet still usable. When you encrypt something, you make it unreadable to all except those who possess the decryption key who can decrypt it and use it. Actually, there was a way to encrypt data in such a way that it can be used without needing to decrypt it; this is what is known as functional encryption and is a recent development in cryptography which is still not quite practical. It is very expensive to do.

Program code is often obfuscated to prevent reverse engineering or at least make it much more difficult.

Perfect forward secrecy is a property of a public key system in which a short-term key derived from a long-term key is not compromised even if the originating key is compromised. In the future, a cryptosystem has the property of forward secrecy if plain text inspection of the data exchange that occurs during the key exchange phase of the session initiation does not reveal the key that was used to encrypt the remainder of the session. Google provides forward secrecy to Gmail and Google Docs. Twitter provides it to the Wikimedia Foundation. All Apple ios apps and many secure messaging services, like signal, also provide forward secrecy. Security through obscurity alone has never been a valid method of protecting secrets; this has been known for centuries, but obscurity still plays a role in security. Making a system hard for an attacker to understand or predict increases the work factor for the attacker.

Slide 4



Principles of Computer Security, Fifth Edition

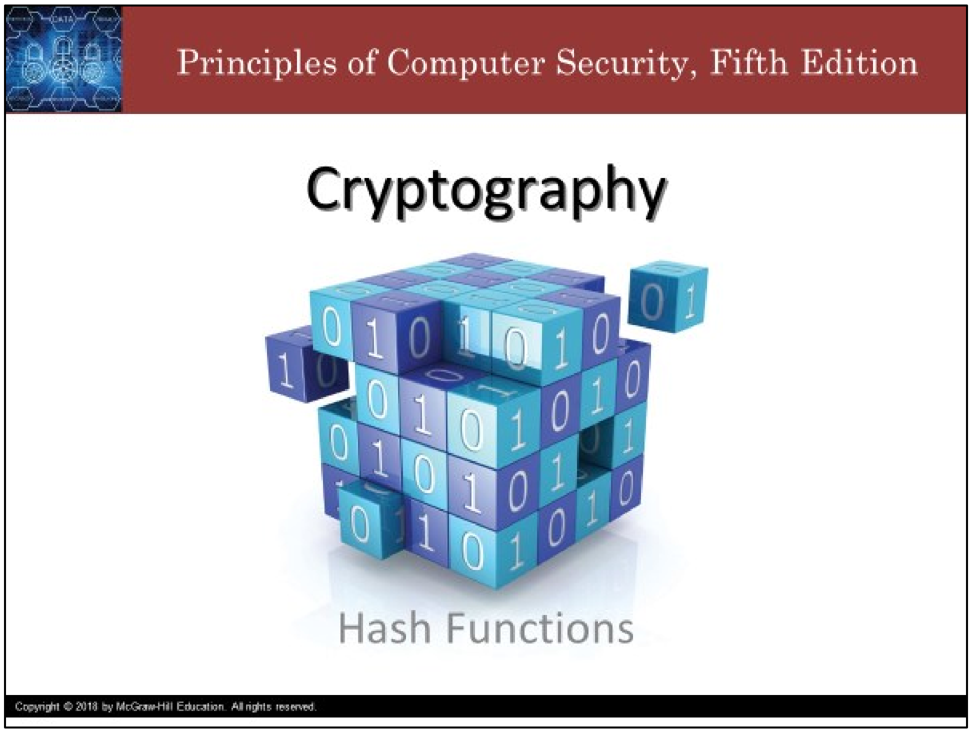
Attribution

- The course slides are based on slides developed by the textbook authors, W. "Art" Conklin, Ph.D. (University of Houston), and Greg White, Ph.D. (University of Texas at San Antonio).
- These slides remain the intellectual property protected by US Copyright law of the authors and the textbook company.
- There have been some changes to the slide deck.

Thank you, and take care.

Cryptography: Hash Functions

Slide 1



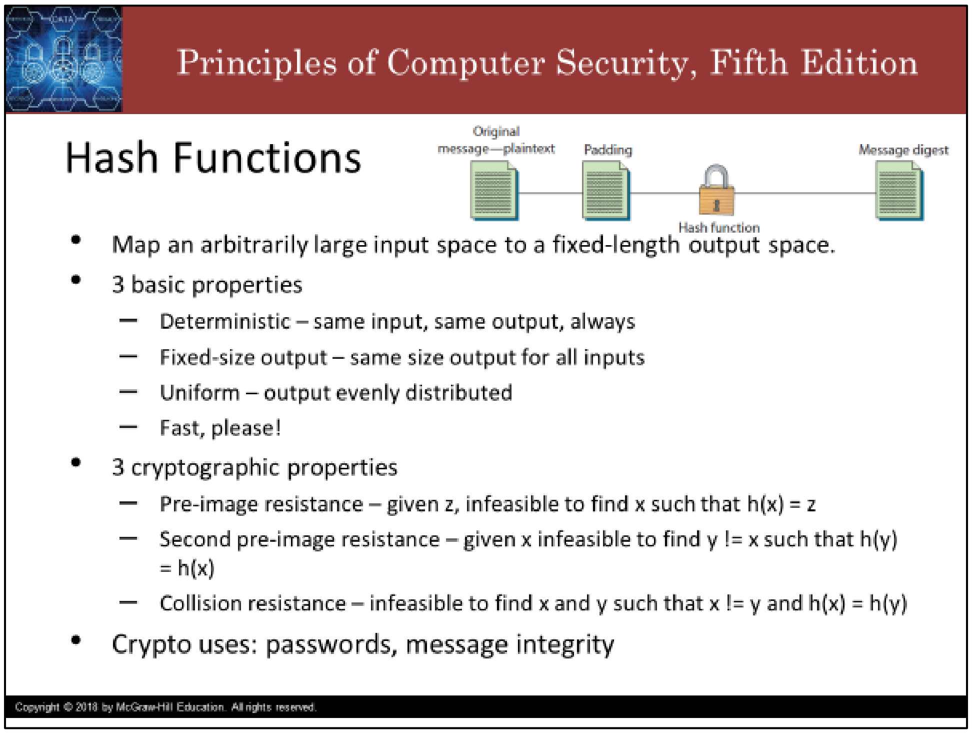
Principles of Computer Security, Fifth Edition

Cryptography

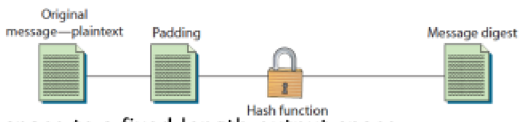
Hash Functions

Howdy! In this video, we discuss hash functions.

Slide 2


Principles of Computer Security, Fifth Edition

Hash Functions



- Map an arbitrarily large input space to a fixed-length output space.
- 3 basic properties
  - Deterministic – same input, same output, always
  - Fixed-size output – same size output for all inputs
  - Uniform – output evenly distributed
  - Fast, please!
- 3 cryptographic properties
  - Pre-image resistance – given z, infeasible to find x such that h(x) = z
  - Second pre-image resistance – given x infeasible to find y != x such that h(y) = h(x)
  - Collision resistance – infeasible to find x and y such that x != y and h(x) = h(y)
- Crypto uses: passwords, message integrity

A hash function is a function which maps an arbitrarily large input space to a fixed-length output space.

Hash functions have 3 important properties:

1. They are **deterministic** – a given input always hashes to the same value. This is so that the hash value is reproducible by anyone else running the same algorithm with the same input.
2. They have **fixed-size output** – the output is always the same size. A 128-bit hash function always outputs exactly 128 bits (values less than 2^127 have 0s at the front).
3. The output is **uniform** – the hash values are evenly distributed over the entire output space.

Also, ideally, the hash function is fast to compute.

But, when used for cryptographic purposes, hash functions must live up to a higher standard.

A cryptographic hash function should have an additional 3 important properties:

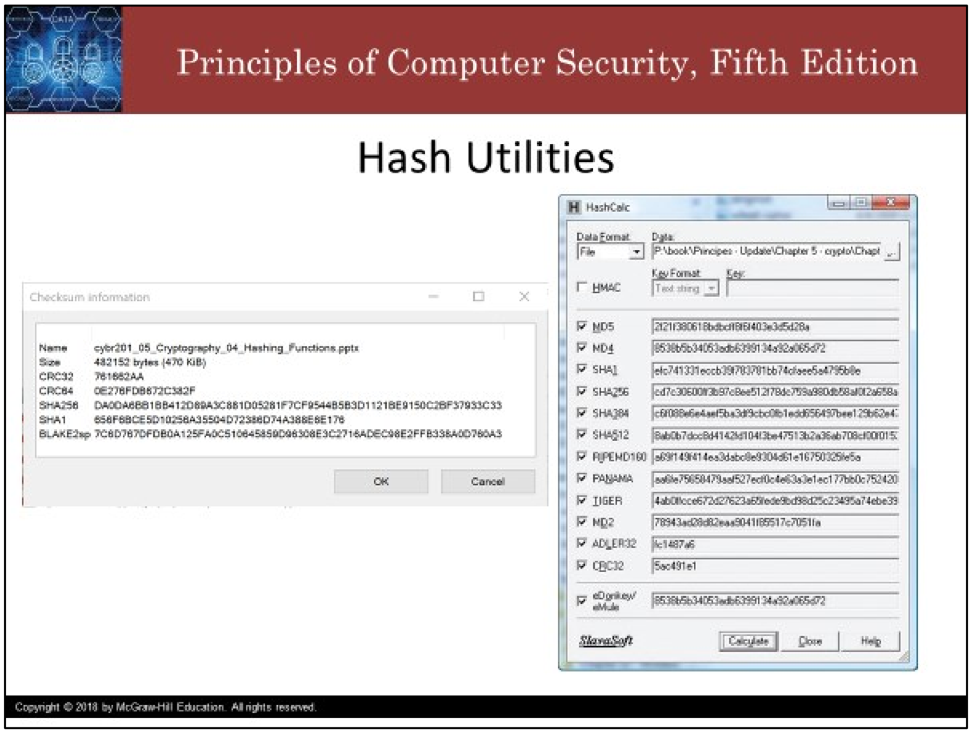
It should be **preimage resistant** – it should be computationally infeasible to find an input that hashes to a given value. This property is why cryptographic hash functions are referred to as 1-way functions. Since a hash function has arbitrary-length input and fixed-length output, there are infinitely many inputs that all map to the same output, so even if you could reverse the function, it would be infeasible to find an input which made sense (most of the infinite inputs that all map to the same output are insanely large blobs of garbage).

Yes, we've had one, but what about second preimage resistance? That's right. The function should also be **second preimage resistant** – given one input; it should be computationally infeasible to find a second input such that both inputs hash to the same output.

And the function should be **collision-resistant** – it should be computationally infeasible to find any two inputs that have the same hash value.

The most common uses for hash function in crypto are for password authentication (so the system never sees your password, it only sees a hash of your password, which it compares to a hash value stored in a database), and ensure message integrity (so that any unauthorized modification will be detected, as such modification could be due to tampering and would make the data unsafe to use).

Slide 3



Principles of Computer Security, Fifth Edition

Hash Utilities

Checksum information

```
Name      cybr201_05_Cryptography_04_Hashing_Functions.pptx
Size      482152 bytes (470 KiB)
CRC32     761882AA
CRC64     0E276FDB872C382F
SHA256    DA0DA6BB1BB412D89A3C881D05281F7CF9544B5B3D1121BE9150C2BF37933C33
SHA1      658F8BCE5D10258A35504D72386D74A388E8E176
BLAKE2sp  7C8D787DFDB0A125FA0C5106458S9D98308E3C2716ADEC98E2FFB338A0D780A3
```

HashCalc

```
Data Format: File    Data: P:\book\Principes - Update\Chapter 5 - crypto\Chapt
[ ] HMAC    Key Format: Text string    Key:

[✓] MD5         2121f380618bdbcf8f6f403e3d5d28a
[✓] MD4         8538b5b34053adb6399134a92a065d72
[✓] SHA1        efc741331eccb39f783781bb74cfaee5a4795b8e
[✓] SHA256      cd7c30600f3b97c8ee512f78dc759a980db58af0f2a658a
[✓] SHA384      c6f088e6e4aef5ba3df9cbc0fb1edd656497bee129b62e4
[✓] SHA512      8ab0b7dcc8d4142fd104f3be47513b2a36ab708cf00f015
[✓] RIPEMD160   a69f149f414ea3dabc8e9304d61e16750325fe5a
[✓] PANAMA      aa6fe79658479aaf527ecf0c4e63a3e1ec177bb0c752420
[✓] TIGER       4ab0fcce672d27623a65fede9bd98d25c23495a74ebe39
[✓] MD2         78943ad28d82eaa9041f85517c7051fa
[✓] ADLER32     fc1487a6
[✓] CRC32       5ac491e1

[✓] eDonkey/eMule  8538b5b34053adb6399134a92a065d72
```

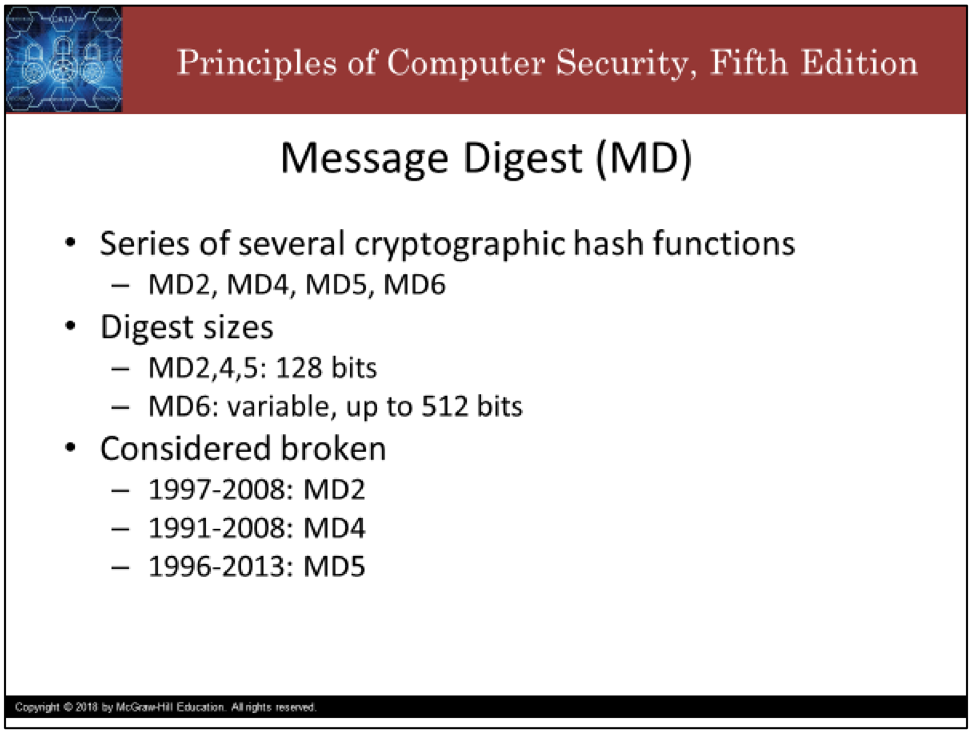
Copyright © 2018 by McGraw-Hill Education. All rights reserved.

Several programs are available that will accept an input and produce a hash value, letting you independently verify the integrity of downloaded content.

7-zip is a program whose primary function is file compression (I use it because Windows' own compression utilities only support zip compression, which is just not enough for me). But, it also has a nice checksum utility that it will add to the context menu so I can easily compute checksums and hashes. The menu entry is named CRC SHA, and the * option will compute all hashes.

HashCalc by SlavaSoft is another free utility that includes a wide range of hash functions.

Slide 4



Principles of Computer Security, Fifth Edition

Message Digest (MD)

- Series of several cryptographic hash functions
  - MD2, MD4, MD5, MD6
- Digest sizes
  - MD2,4,5: 128 bits
  - MD6: variable, up to 512 bits
- Considered broken
  - 1997-2008: MD2
  - 1991-2008: MD4
  - 1996-2013: MD5

M.D. is the generic name for a series of cryptographic hash functions.

MD2, 4, and 5 all create 128-bit hashes

MD6 can create up to 512-bit hashes

MD2 was published in 1989 and was optimized for 8-bit machines.

MD4 and 5 were published in 1990 and 1992, respectively, and were optimized for 32-bit machines.

MD6 was published in 2008. I am unsure whether MD6 is optimized for 64-bit machines, but I do know it is designed to take advantage of multi-core processors.

MD2, 4, and 5 are all considered broken.

In 1997, collisions in a component of MD2 were described. In 2004, MD2 was shown to be vulnerable to a preimage attack in 2^104 time. In 2008, that was lowered to 2^73 time.

In 1991, weaknesses in MD4 were demonstrated. The first collision attack was demonstrated in 1995. In 2004 and 2007, MD4 was completely destroyed by an attack which finds collisions almost as fast as it takes to verify them. The attack runs in just 2 or 3 operations. MD4's preimage resistance was broken in 2008 with an attack that runs int 2^102 time. In 2011 it was officially declared obsolete.

In 1996, a flaw in the design of MD5 was discovered. In 2004, it was shown that MD5 is not collision-resistant. In 2013 the best collision attack runs in time 2^18. The best attack on MD5's preimage resistance was published in 2009 and runs in time 2^123.

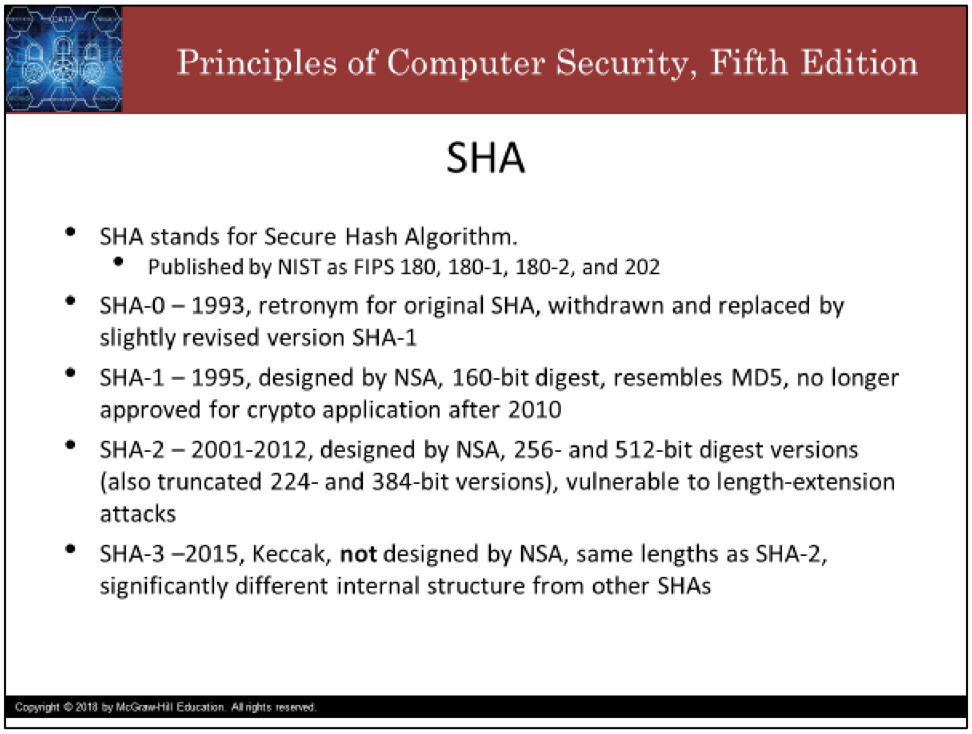
MD2 is, surprisingly, still around in some legacy systems. It has been disabled in SSL/TLS since 2009.

MD4, as I said, has been obsolete since 2011. MD4's design influenced later designs of MD5, SHA-1, and RIPEMD, all of which are now considered insecure.

Despite its well-documented weaknesses and deprecation by security experts, MD5 refuses to die and is still widely used. As of 2008, the CMU Software Engineering Institute considers MD5 essentially "cryptographically broken and unsuitable for further use. "

MD6 was a candidate for SHA-3 but was eliminated in the 1st round. I don't know of any commercial uses of MD6.

Slide 5



Principles of Computer Security, Fifth Edition

SHA

- SHA stands for Secure Hash Algorithm.
  - Published by NIST as FIPS 180, 180-1, 180-2, and 202
- SHA-0 – 1993, retronym for original SHA, withdrawn and replaced by slightly revised version SHA-1
- SHA-1 – 1995, designed by NSA, 160-bit digest, resembles MD5, no longer approved for crypto application after 2010
- SHA-2 – 2001-2012, designed by NSA, 256- and 512-bit digest versions (also truncated 224- and 384-bit versions), vulnerable to length-extension attacks
- SHA-3 –2015, Keccak, **not** designed by NSA, same lengths as SHA-2, significantly different internal structure from other SHAs

The Secure Hash Algorithms are a family of cryptographic hash functions published by the National Institute of Standards and Technology (NIST) as a U.S. Federal Information Processing Standard (FIPS).

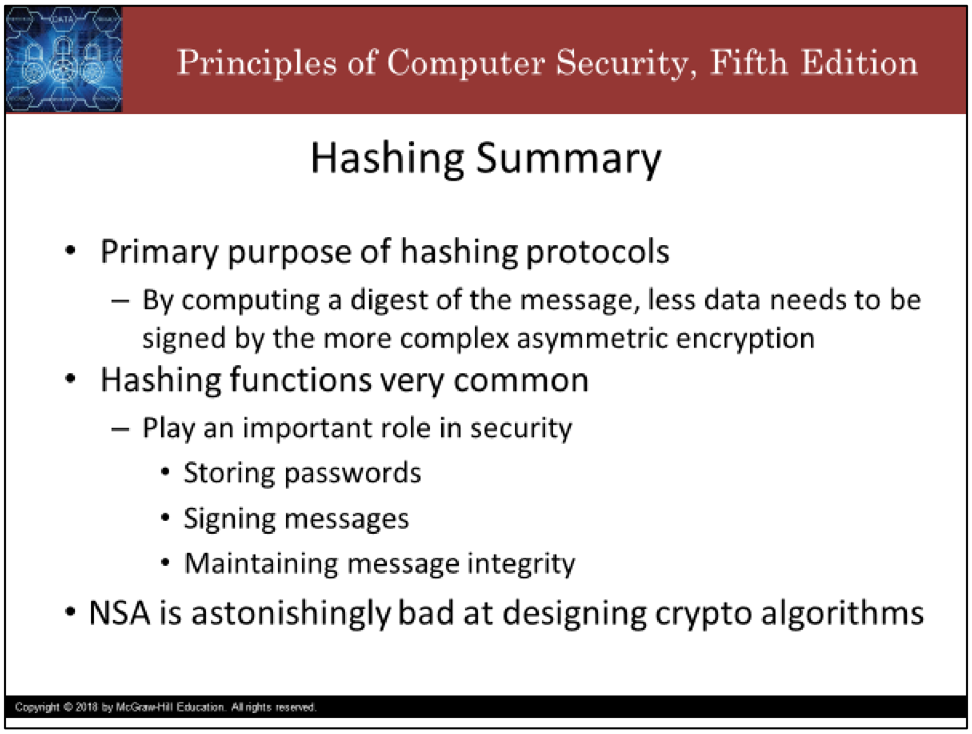
SHA-0 was published in 1993 but was quickly withdrawn due to an undisclosed significant flaw. It was replaced in 1995 by SHA-1.

SHA-1 was designed by NSA to be part of the Digital Signature Algorithm. But, it is now broken.

SHA-2 is actually 2 has functions with different block sizes: SHA-256 and SHA-512. SHA-256 uses 32-bit words, SHA-512 uses 64-bit words, which matters for the processor, which must compute them. SHA-2 was also designed by the NSA and, while not considered broken, some versions are vulnerable to length extension attacks, which allow an attacker to compute the correct hash value of the concatenation of the original input and a new input without knowing the original input.

SHA-3 is the newest member of the family, selected in 2015 through a public competition of non-NSA designers. The algorithm which was selected is called Keccak. It supports the same output lengths as SHA-2, but the internal structure is significantly different. Keccak uses an internal state of 1600 bits (compared to SHA-2's 256 or 512). The best public cryptanalysis of SHA-3 requires an obscene 2^511.5 time and 2^508 memory for an attack against just the first 8 rounds (out of a total of 24, so it only works if the implementation only does 1/3rd the work it is supposed to).

Slide 6



Principles of Computer Security, Fifth Edition

Hashing Summary

- Primary purpose of hashing protocols
  - By computing a digest of the message, less data needs to be signed by the more complex asymmetric encryption
- Hashing functions very common
  - Play an important role in security
    - Storing passwords
    - Signing messages
    - Maintaining message integrity
- NSA is astonishingly bad at designing crypto algorithms

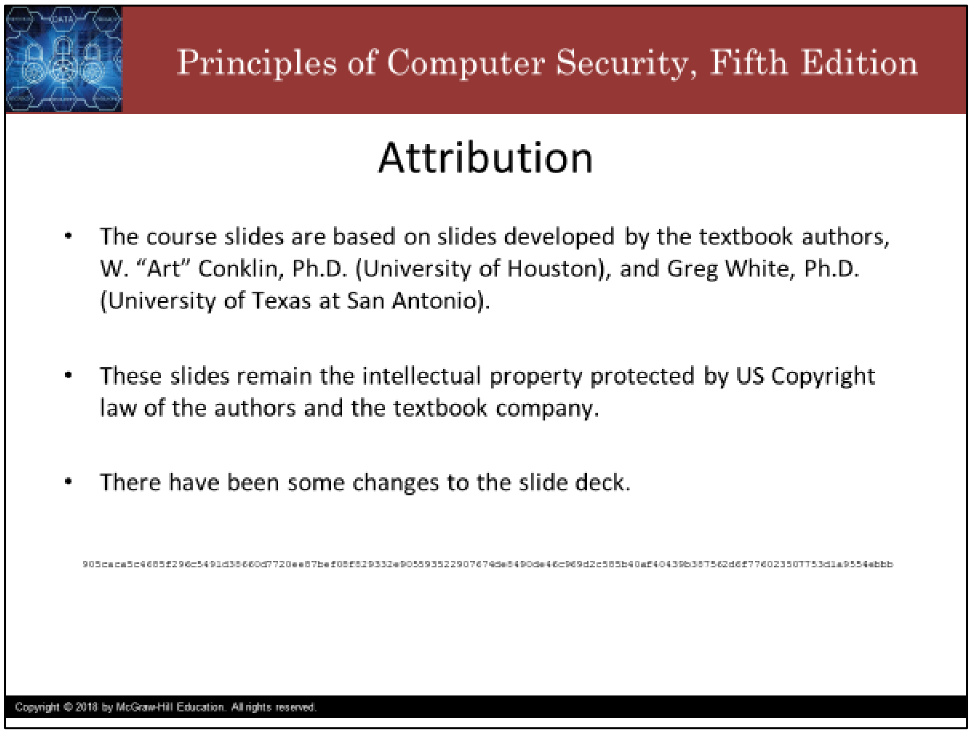
The primary purpose of hash functions is to reduce the size of the data. This is of particular utility for digital signatures, which use expensive asymmetric cryptosystems. The smaller the data, the faster it is to sign.

Hash functions are also used for storing passwords and detecting tampering.

Finally, do not use MD5, SHA-1, or SHA-2 for security-critical applications. Use SHA-3.

If you just want to have some fun, MD5 and SHA-1 and 2 are great for practicing.

Slide 7



Principles of Computer Security, Fifth Edition

Attribution

- The course slides are based on slides developed by the textbook authors, W. “Art” Conklin, Ph.D. (University of Houston), and Greg White, Ph.D. (University of Texas at San Antonio).
- These slides remain the intellectual property protected by US Copyright law of the authors and the textbook company.
- There have been some changes to the slide deck.

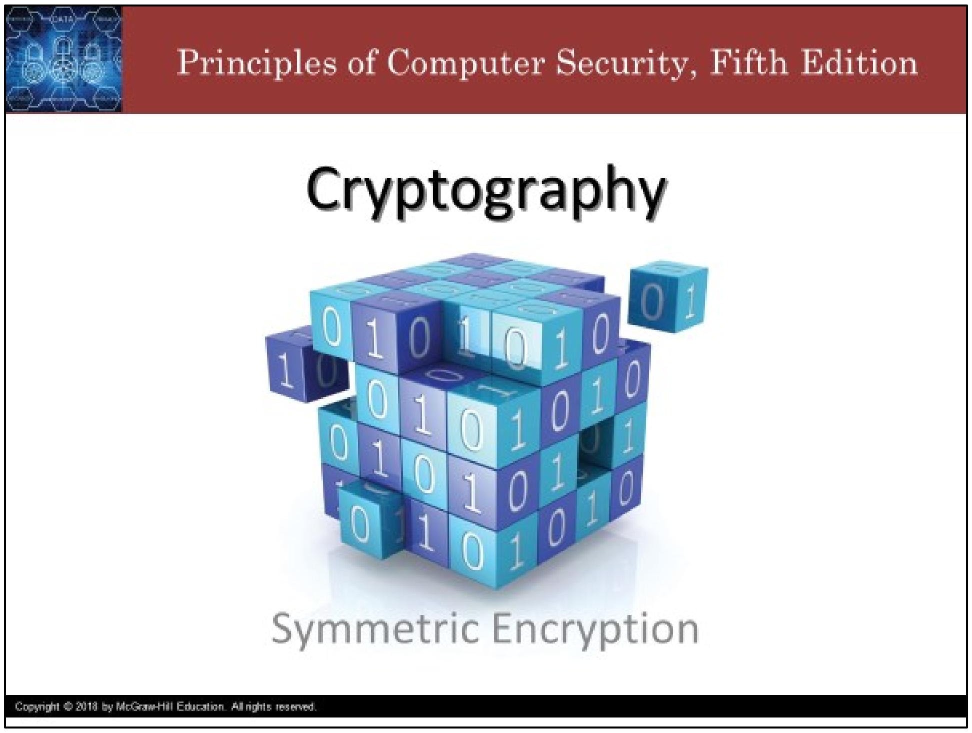
905caca5c4685f296c5491d35660d7720ee87bef05f529332e905593522907674de5490de46c969d2c585b40af40439b387562d6f776023507753d1a9554ebbb

Thank you and take care.

(Can you crack the hash?)

Cryptography: Symmetric Encryption

Slide 1



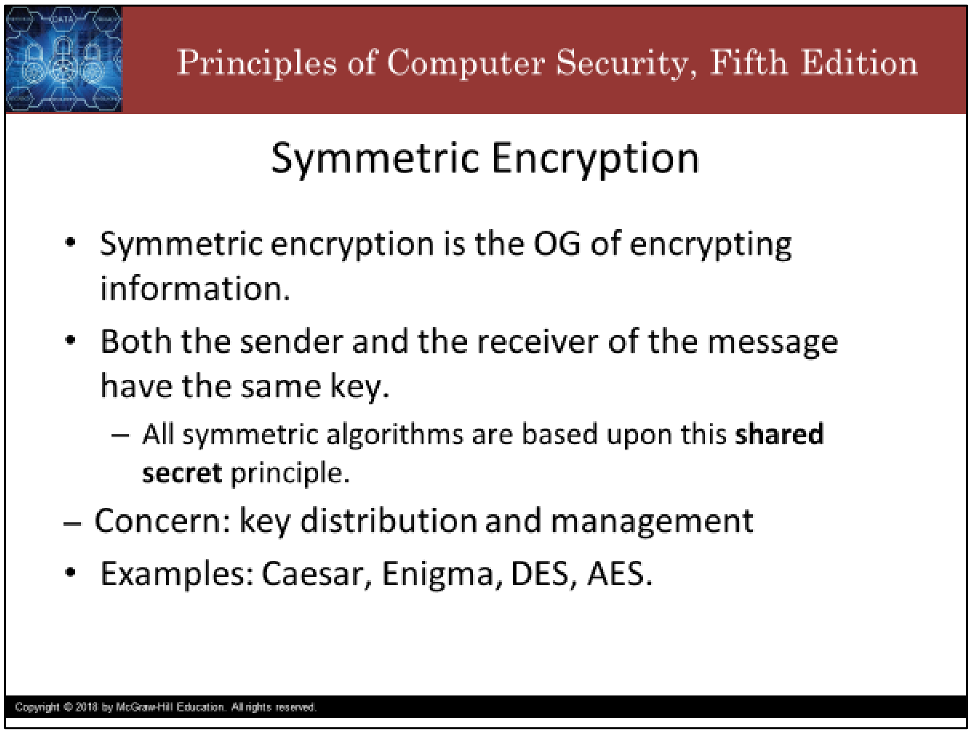
Principles of Computer Security, Fifth Edition

Cryptography

Symmetric Encryption

Howdy! In this video, we discuss symmetric encryption.

Slide 2


Principles of Computer Security, Fifth Edition

Symmetric Encryption

- Symmetric encryption is the OG of encrypting information.
- Both the sender and the receiver of the message have the same key.
  - All symmetric algorithms are based upon this **shared secret** principle.
- Concern: key distribution and management
- Examples: Caesar, Enigma, DES, AES.

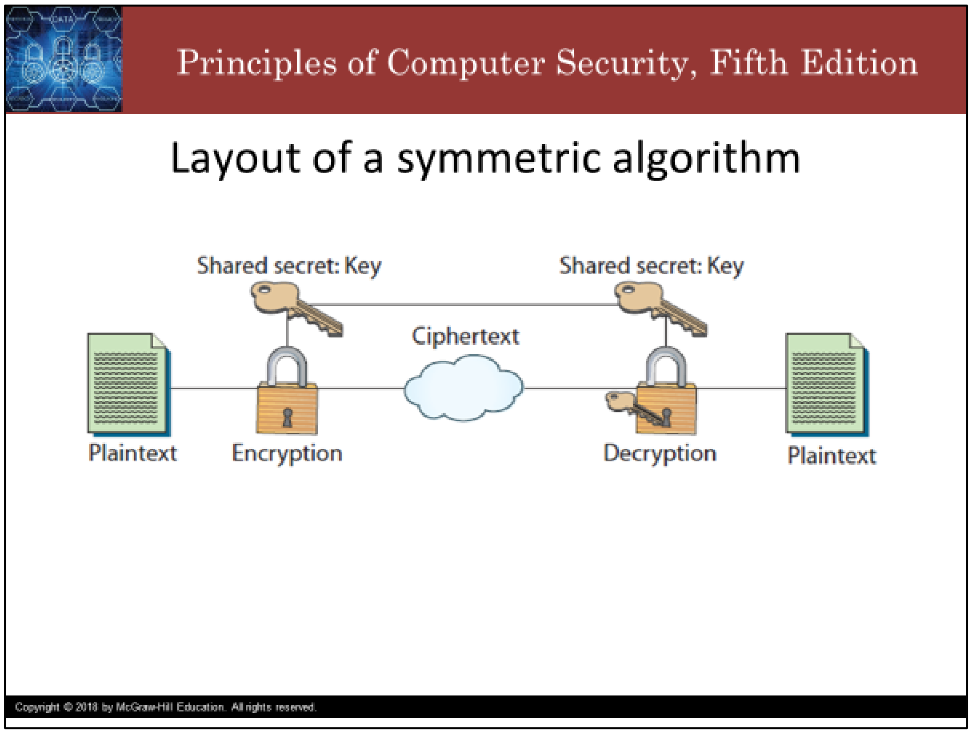
Symmetric encryption is the oldest method of encryption, going back more than 2500 years to at least 500 BCE. All of the so-called classical cryptosystems were symmetric.

All symmetric algorithms are based upon a shared secret principle: the sender and the receiver each know a secret key, which is used for both encryption and decryption.

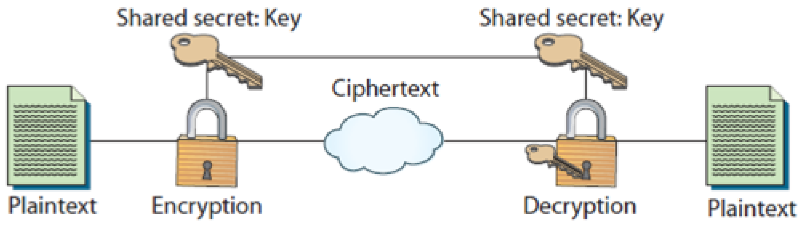
Since both ends need to have the same key in order to communicate, there is a problem that must be overcome, which is how to share the key. One option is to meet in person. But, if that is not possible, then how can it be done? Without a secure channel, anything we send could be read by the adversary, including the key. And we need the key to be shared in order to create the secure channel. It seems like there is a chicken and egg problem: making a secure channel requires sharing the key, sharing the key requires a secure channel. This is a problem which is not solvable by symmetric cryptosystems. But, there is a solution. Until that solution was invented, this problem of key distribution was one of the major thorns in the application of symmetric cryptosystems. A valid and successful attack on a symmetric cryptosystem will always be to intercept or steal the secret key. For that reason, once the key is shared, it is of the utmost importance to keep it secret. And, like passwords, using the same key for many communication channels is insecure, so we'll have many keys to keep track of. This is where key management comes in. Keep the keys secure, organized, and up to date is priority number 1.

Famous examples of symmetric cryptosystems include the Caesar cipher, the Enigma machine, and both DES and AES.

Slide 3


Principles of Computer Security, Fifth Edition

Layout of a symmetric algorithm

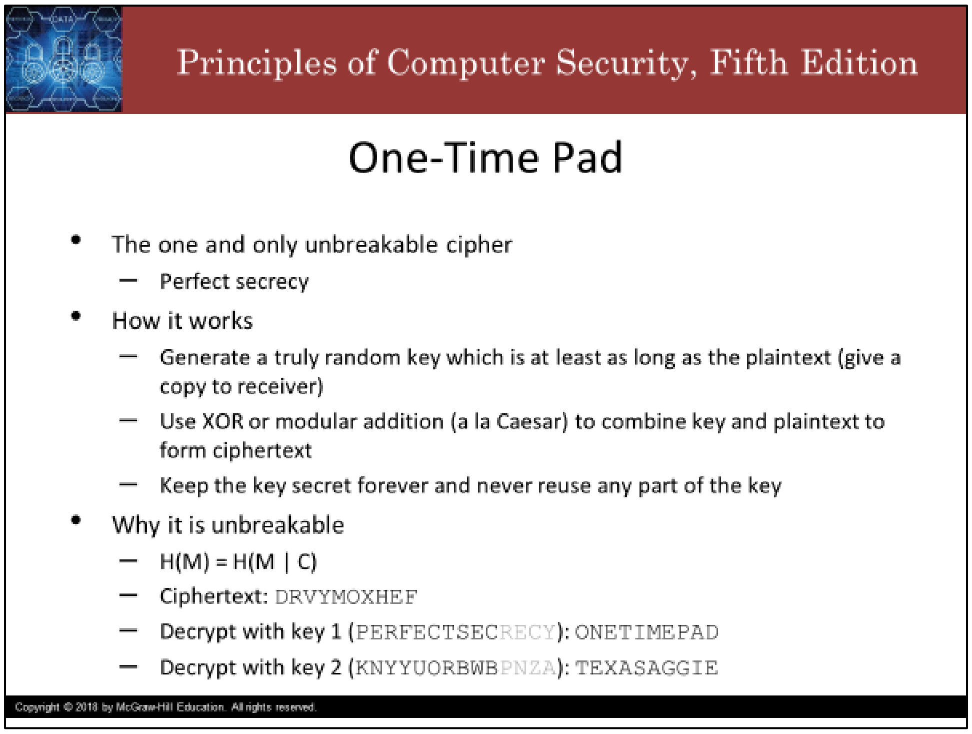


This is a simple diagram showing the process that a symmetric cryptosystem goes through to protect the data from sender to receiver.

The plaintext is encrypted using the key to form the ciphertext.

This ciphertext is transmitted to the recipient, who decrypts the ciphertext using the same key that was used for encryption.

Slide 4



Principles of Computer Security, Fifth Edition

One-Time Pad

- The one and only unbreakable cipher
  - Perfect secrecy
- How it works
  - Generate a truly random key which is at least as long as the plaintext (give a copy to receiver)
  - Use XOR or modular addition (a la Caesar) to combine key and plaintext to form ciphertext
  - Keep the key secret forever and never reuse any part of the key
- Why it is unbreakable
  - H(M) = H(M | C)
  - Ciphertext: DRVYMOXHEF
  - Decrypt with key 1 (PERFECTSECRECY): ONETIMEPAD
  - Decrypt with key 2 (KNYYUORBWBPNZA): TEXASAGGIE

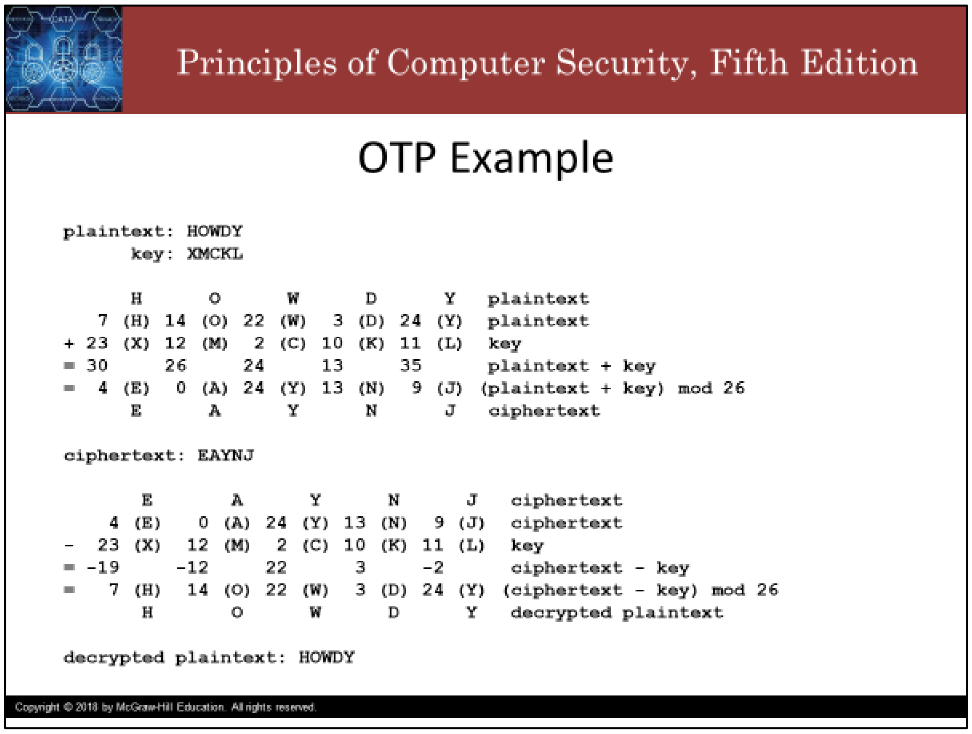
The one-time pad is a very special cipher. Depending on how pedantic about mathematical interpretations you want to be, the one-time pad is also a unique cipher with respect to the level of security it provides because the one-time pad is the _only_ unbreakable cipher. It has the property of perfect secrecy, also called information-theoretic security.

The way it works is simple in theory: pick a long and truly random sequence of values to the key. The key material must be at least as long as the plaintext message. Give a copy of the key to the receiver and send them off. When you need to send a message, add it or XOR it with the key (truncating after you run out of plaintext) in an operation similar to a Caesar cipher: just shift each symbol of plaintext by the amount specified by the corresponding key symbol. This will produce the ciphertext. Destroy the key so you can never reuse any part of it, and no one will ever find it. Now the ciphertext is unbreakable, and only the one who has the other copy of the key can decrypt it. Send the ciphertext by some reliable means of transport (skywriting, billboard, classified ad, courier service, TCP/IP, USPS, whatever). When the receiver gets it, they perform a similar process, but in reverse, to recover the message. They should immediately destroy the key material so it cannot be compromised and, once they have read the message, they should destroy that, too.

While this is simple in theory, there are several practically hard parts: generating truly random values is hard, securely distributing the key material is hard, each key can only be used once, so probably lots of key material will be shared at the start, which makes key management harder, messages may arrive out of order, and so many keys will need to be tried before the right one is found, and violating the prohibition against reuse of the key completely voids the security of the cipher.

But, if you can manage to do all of those things correctly, then the cipher really is unbreakable. The reason for this is that the ciphertext gives no information about the plaintext (besides possibly the length, but padding can hide that). Since the key is random, all keys are equally likely, as are all ciphertexts. Since each key corresponds to a unique decryption, there are as many possible decryptions as there are keys, which, because the key is as long as the plaintext, means there are as many possible decryptions as there are possible plaintexts. That is, the plaintext space and the ciphertext space are the same sizes (the keyspace can actually be bigger, so there could be several keys that bind any two plaintext and ciphertext pairs). Thus, a given ciphertext could decrypt with equal probability to any plaintext. The example on the slide is contrived, and one of the keys is not random. If the key is not random, then deciding which plaintext is the right plaintext is easier.

Slide 5


Principles of Computer Security, Fifth Edition

OTP Example

```
plaintext: HOWDY
      key: XMCKL

      H       O       W       D       Y    plaintext
   7 (H)  14 (O)  22 (W)   3 (D)  24 (Y)   plaintext
+ 23 (X)  12 (M)   2 (C)  10 (K)  11 (L)   key
= 30      26      24      13      35       plaintext + key
=  4 (E)   0 (A)  24 (Y)  13 (N)   9 (J)  (plaintext + key) mod 26
      E       A       Y       N       J    ciphertext

ciphertext: EAYNJ

       E       A       Y       N       J    ciphertext
    4 (E)    0 (A)  24 (Y)  13 (N)   9 (J)  ciphertext
-  23 (X)   12 (M)   2 (C)  10 (K)  11 (L)  key
= -19      -12      22       3      -2      ciphertext - key
=   7 (H)   14 (O)  22 (W)   3 (D)  24 (Y)  (ciphertext - key) mod 26
       H       O       W       D       Y    decrypted plaintext

decrypted plaintext: HOWDY
```

This is a worked example of the application of a one-time pad.

The plaintext is HOWDY

The key is XMCKL.

The ciphertext is obtained by adding the plaintext and the key mod 26.

I am a computer scientist, so I start counting at 0. A is 0, B is 1, and so on.

H is letter number 7, X is letter number 23. 7 + 23 is 30, which leaves a remainder of 4 after division by 26.

O is 14, M is 12, which sum to 26, which is 0 mod 26, which is the letter A.

And so on until every symbol of the plaintext has been processed.

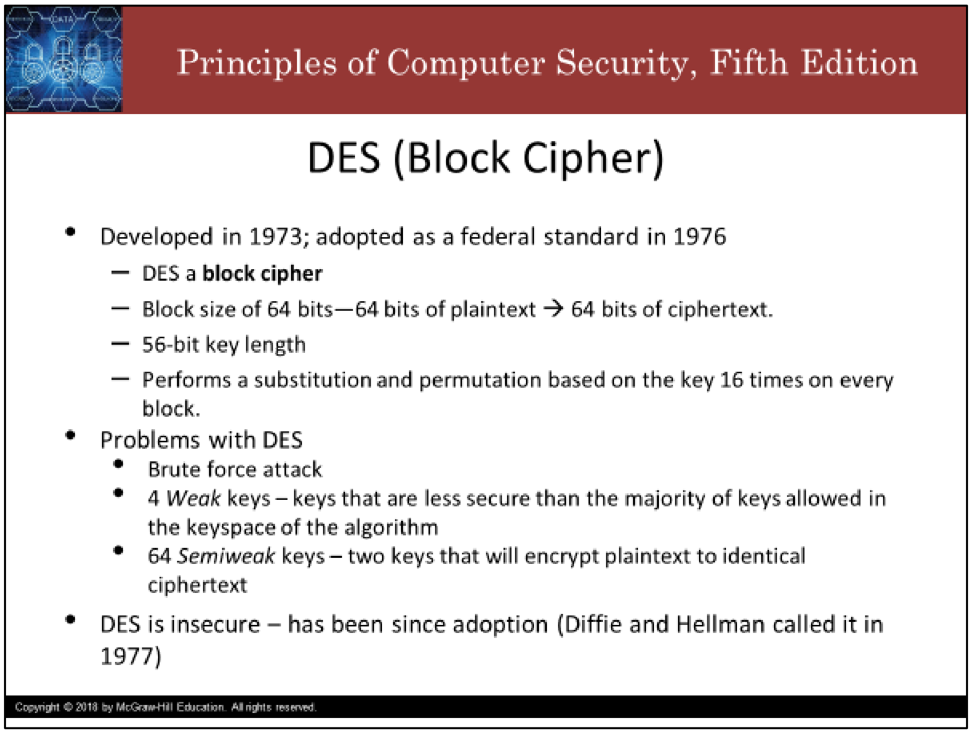
The resulting ciphertext is EAYNJ.

To decrypt, the process goes in reverse.

E is 4, X is 23, 4 – 23 is -19, which is 7 mod 26, which gives H.

Doing that same operation for the rest of the ciphertext and the key yields the decrypted plaintext, which is HOWDY.

Slide 6



Principles of Computer Security, Fifth Edition

DES (Block Cipher)

- Developed in 1973; adopted as a federal standard in 1976
  - DES a **block cipher**
  - Block size of 64 bits—64 bits of plaintext → 64 bits of ciphertext.
  - 56-bit key length
  - Performs a substitution and permutation based on the key 16 times on every block.
- Problems with DES
  - Brute force attack
  - 4 *Weak* keys – keys that are less secure than the majority of keys allowed in the keyspace of the algorithm
  - 64 *Semiweak* keys – two keys that will encrypt plaintext to identical ciphertext
- DES is insecure – has been since adoption (Diffie and Hellman called it in 1977)

In response to a call for proposal for a cryptographic algorithm to become a standard, The National Bureau of Standards (now called NIST) received an algorithm called Lucifer, which had been developed by IBM.

The NBS and the NSA worked together to analyze and augment the algorithm's security. The NSA's changes strengthened the algorithm against differential cryptanalysis, but they wanted to reduce the key size from 128 to 48 to weaken it against brute-force attacks. IBM agreed to 56 bits (reducing the amount of work required to brute force the key by a factor of 2^72). DES was adopted as a federal standard in 1976.

DES is a block cipher, which means it processes the plaintext into ciphertext in blocks. DES uses a block size of 64 bits.

The 56-bit key is chopped, rotated, and permuted to form 48-bit subkeys for each of the 16 rounds of confusion and diffusion that DES performs.

The internal structure of DES is what is known as a Feistel network, named after Howard Feistel, the inventor of the algorithm which became DES.

A neat property of DES is that decryption runs in the same direction as encryption but with the keys in reverse order.

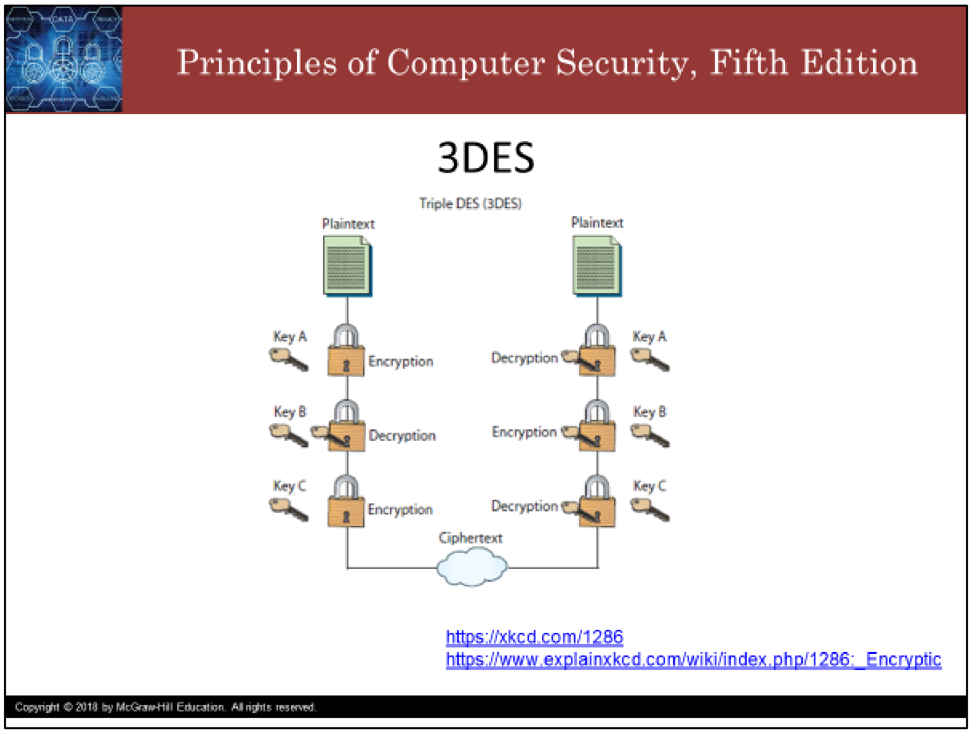
The main problem with DES is that the keyspace is too small, which makes it vulnerable to a brute force attack. This type of attack requires a lot of work, and brute-forcing even a 56-bit key in a reasonable amount of time was out of reach of most organizations for about the first 10 - 15 years of DES's run, mainly due to cost/benefit analysis, and not the actual dollar cost. Breaking keys requires a lot of compute power, and back then, no one knew what to do with so much power, and breaking DES keys is neat but not actually commercially valuable. Remember, the Internet was still in its infancy back then. eCommerce and the dot com boom were still 20 years away.

If not for the tiny keyspace, DES is actually a very strong cryptosystem. After many years of cryptanalysis, some weak keys were found, which are too insecure, and some semiweak keys were found, which allow 2 different semiweak keys to decrypt the same ciphertext. The best known public cryptanalysis, which is faster than brute force, is a linear cryptanalysis attack which requires 2^43 known plaintexts and runs in time 2^39 – 2^43. That amount of computational resources is possible in a desktop computer today.

DES has been highly influential in advancing cryptography. Some of the ideas that were developed in DES are still used in modern cryptosystems today, including in AES, the successor to DES.

DES insecure from the start: Diffie, Whitfield; Hellman, Martin E. (June 1977). "Exhaustive Cryptanalysis of the NBS Data Encryption Standard" (PDF). Computer. 10 (6): 74–84. doi:10.1109/C-M.1977.217750

Slide 7


Principles of Computer Security, Fifth Edition

3DES

Triple DES (3DES)
Plaintext
Plaintext
Key A
Encryption
Decryption
Key A
Key B
Decryption
Encryption
Key B
Key C
Encryption
Decryption
Key C
Ciphertext

https://xkcd.com/1286
https://www.explainxkcd.com/wiki/index.php/1286:_Encryptic

Triple DES (also called 3DES) is a variant of DES.

Depending on the variant, it uses either two or three keys. Either way, the effective key length is 112 bits (or two DES keys).

3DES uses Multiple encryption. The algorithm just runs normal DES 3 times, but with a twist.

Multiple encryption can be performed in several different ways.

The simplest method of multiple encryption is just to stack algorithms on top of each other: Taking plaintext, encrypting it with DES with one key, then encrypting the first ciphertext with a different key, and finally encrypting the second ciphertext with a third key.

This technique actually does not increase security as much as you would hope, though. Instead of 3 times the key length, I think the effective key length is increased only by half, which puts it at 84 or 85 bits.

What 3DES does is encrypt with one key, then decrypt with a second, and then encrypt with a third.

This gives 3DES an effective key length of 112 bits, twice that of single DES.

This greatly increases the number of attempts needed to retrieve the key and is a significant enhancement of security against brute force attack.

The additional security comes with a price, however. It can take up to three times longer to compute 3DES than to compute single DES.

While this is the case, advances in memory and processing power in today's electronics make this problem irrelevant in most devices.

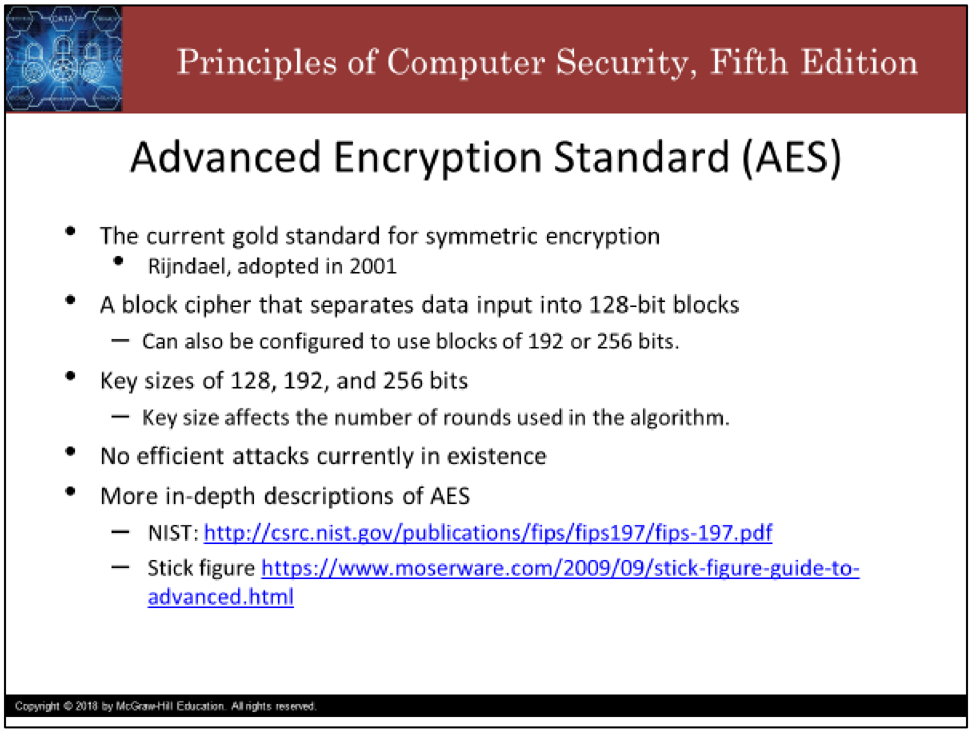
So, 3DES is stronger than DES, but it still has similar weakness.

The longer key length makes it more resistant to brute force attacks, and for that reason, 3DES is still used today for limited applications. However, as of 2016, it is considered a weak cipher.

AES has taken over as the symmetric encryption standard.

Several years ago, in I think it was 2013, hackers breached Adobe's user database and dumped 153 million user emails, encrypted passwords, and password hints. And, unfortunately for the users, Adobe stored the hint in plaintext and misused the cryptosystem, which was none other than block mode 3DES. So, people had a good time guessing the passwords using the hints and the broken crypto. Links to the relevant xkcd and explain xkcd article are on the slide for you to check out.

Slide 8



Principles of Computer Security, Fifth Edition

Advanced Encryption Standard (AES)

- The current gold standard for symmetric encryption
  - Rijndael, adopted in 2001
- A block cipher that separates data input into 128-bit blocks
  - Can also be configured to use blocks of 192 or 256 bits.
- Key sizes of 128, 192, and 256 bits
  - Key size affects the number of rounds used in the algorithm.
- No efficient attacks currently in existence
- More in-depth descriptions of AES
  - NIST: http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
  - Stick figure https://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html

AES is the current US standard for encryption. It replaced DES in 2001 after an intense 5-year standardization process which included a competition between 15 different ciphers. The eventual winner was an algorithm named Rijndael.

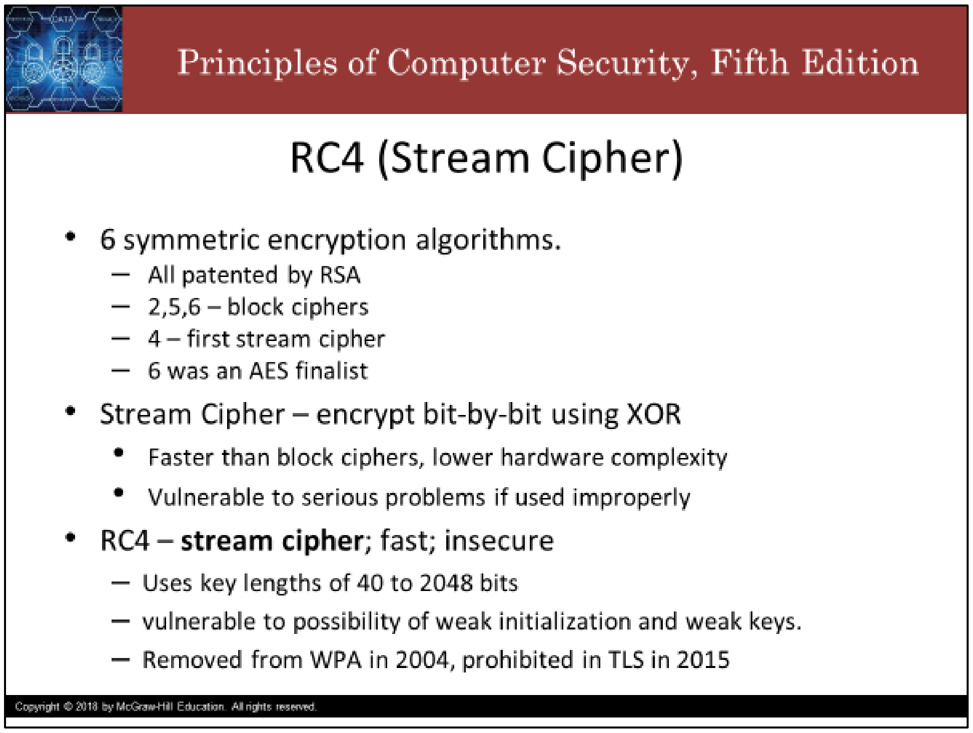
Three versions of Rijndael were selected, with key lengths 128-bits, 192-bits, and 256-bits. All use 128-bit blocks. Other block sizes can be configured, but only the 128-bit block size is part of AES.

Like DES, AES uses rounds to increase the work factor and achieve thorough confusion and diffusion. The number of rounds depends on the key size: 10 rounds for 128-bit keys, 12 for 192-bit keys, and 14 for 256-bit keys.

Attacks have been published that are computationally faster than a full brute-force attack, though none as of 2013 are computationally feasible. The best public cryptanalysis of AES reduces the work only by a factor of about 4 (or 2 bits) using a variant of the meet-in-the-middle attack.

If you want to try your hand (and brain) at grokking AES, I put two links on the slide, one to the FIPS specification and one to a very well-made stick figure guide to AES.

Slide 9



Principles of Computer Security, Fifth Edition

RC4 (Stream Cipher)

- 6 symmetric encryption algorithms.
  - All patented by RSA
  - 2,5,6 – block ciphers
  - 4 – first stream cipher
  - 6 was an AES finalist
- Stream Cipher – encrypt bit-by-bit using XOR
  - Faster than block ciphers, lower hardware complexity
  - Vulnerable to serious problems if used improperly
- RC4 – **stream cipher**; fast; insecure
  - Uses key lengths of 40 to 2048 bits
  - vulnerable to possibility of weak initialization and weak keys.
  - Removed from WPA in 2004, prohibited in TLS in 2015

The RC algorithms are a set of symmetric-key encryption algorithms invented by Ron Rivest (he of MD and RSA fame). The "RC" stands for Rivest's Cipher.

There have been six RC algorithms so far:

RC1 was never published.

RC2 was a 64-bit block cipher developed in 1987.

RC3 was broken before ever being used.

RC4 is a stream cipher developed in 1987.

RC5 is a 32/64/128-bit block cipher developed in 1994.

RC6, a 128-bit block cipher based heavily on RC5, was an AES finalist developed in 1997.

RC4 was actually the first stream cipher.

A stream cipher works by enciphering the plaintext in a stream, usually bit by bit, usually using XOR.

In a sense, a stream cipher is trying to be a one-time pad.

Stream ciphers are faster than block ciphers and have lower hardware complexity, but if used improperly (such as reusing a key), they are vulnerable to serious security problems.

RC4 was developed in 1987 and remained a trade secret of RSA until it was posted to the Internet in 1994.

The key is used to initialize a 256-byte state table.

This table is used to generate the pseudo-random stream that is XORed with the plaintext to generate the ciphertext.

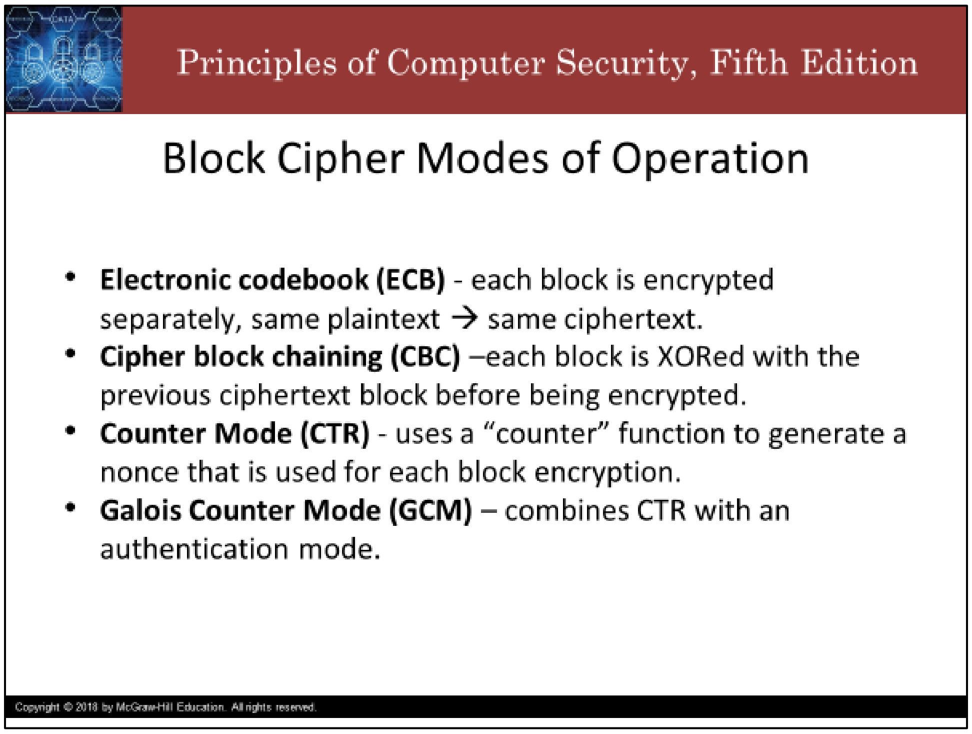
For decryption, the stream is XORed with the ciphertext to produce the plaintext.

Like a good stream cipher, the algorithm is fast and remarkably so given that it runs in software.

However, it is not a good stream cipher because it is insecure, especially so when nonrandom or related keys are used or when it is not properly initialized.

At one time, RC4 was quite popular and widely used. Now, given what we know about how insecure it is and how easy it is to mess up its use, it is blacklisted. If you are old enough to remember the WEP wireless network security standard that is now defunct due to its staggering insecurity, RC4 was a big part of the reason for that insecurity. The WPA standard, which replaced WEP, used RC4 as part of TKIP for packet encryption. WPA2 replaced WPA in 2004 and removed RC4, and replaced it with AES. WPA3 is now available in devices manufactured since July 1, 2020. It still uses AES.

Slide 10



Principles of Computer Security, Fifth Edition

Block Cipher Modes of Operation

- **Electronic codebook (ECB)** - each block is encrypted separately, same plaintext → same ciphertext.
- **Cipher block chaining (CBC)** –each block is XORed with the previous ciphertext block before being encrypted.
- **Counter Mode (CTR)** - uses a "counter" function to generate a nonce that is used for each block encryption.
- **Galois Counter Mode (GCM)** – combines CTR with an authentication mode.

Copyright © 2018 by McGraw-Hill Education. All rights reserved.

Electronic codebook (ECB) - is the simplest mode operation of all. The message to be encrypted is divided into blocks, and each block is encrypted separately.

ECB is nice because it makes encryption and decryption parallelizable and allows random read access into the ciphertext.

However, because the plaintext and the ciphertext have a one-to-one relationship, ECB mode can leak patterns from the plaintext. For example, images encrypted with ECB mode are still recognizable.

Cipher block chaining (CBC) – a block mode where each block is XORed with the previous ciphertext block before being encrypted.

CBC supports random read access and allows decryption to be parallelized, but encryption can not be parallelized due to reliance on previous blocks of ciphertext.

CBC is the most commonly used mode of operation. It does not suffer from the same problem the ECB has where patterns can shine through the ciphertext since, in CBC mode, the same plaintext block in different locations in the plaintext will almost certainly encrypt to different ciphertext blocks.

Counter Mode (CTR) - uses a "counter" function to generate a nonce that is used for each block encryption.

Like ECB, CTR mode lets encryption and decryption be parallelized and allows random read access

Unlike ECB, it does not let patterns in the plaintext show in the ciphertext.

CTR mode turns a block cipher into a stream cipher.

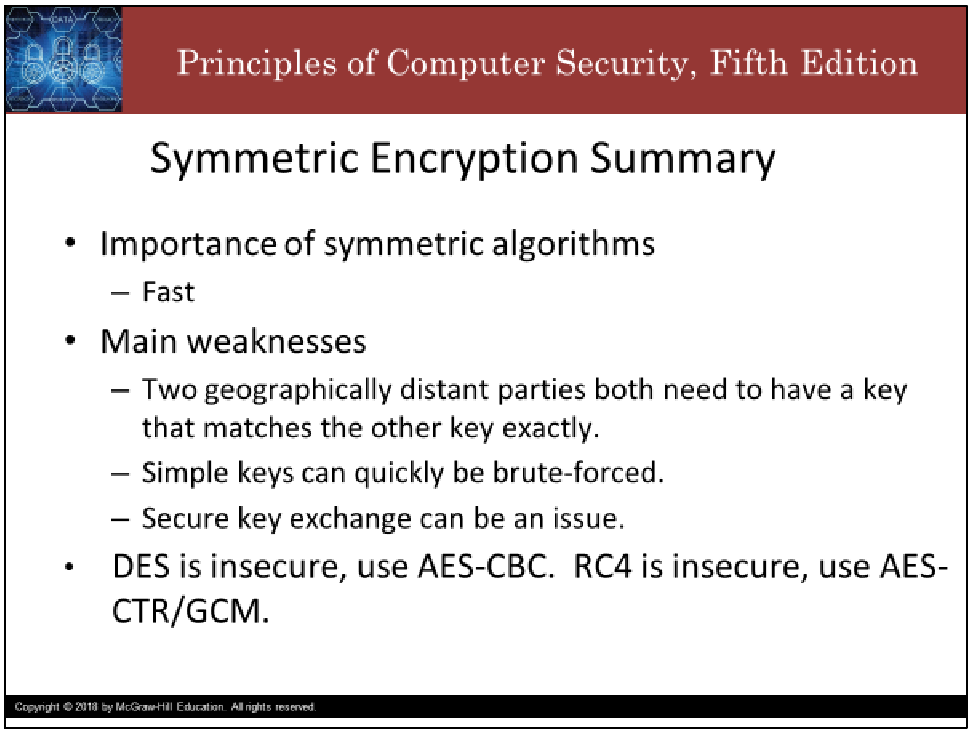
Along with CBC (the most popular block cipher mode), CTR is the other most popular mode.

Galois Counter Mode (GCM) – is an extension of CTR in that it includes the addition of a Galois mode of authentication.

GCM has all the same properties of CTR mode, plus authentication.

GCM is a form of authenticated encryption, which means it supports both the confidentiality and integrity (in the form of authenticity) of the data.

Slide 11


Principles of Computer Security, Fifth Edition

Symmetric Encryption Summary

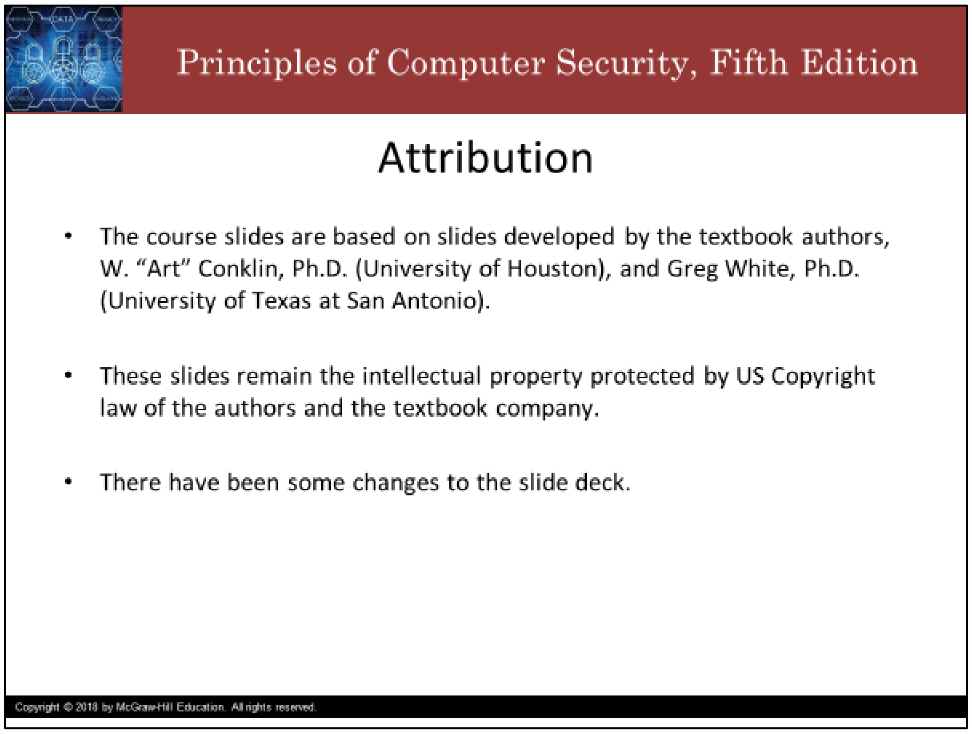
- Importance of symmetric algorithms
  - Fast
- Main weaknesses
  - Two geographically distant parties both need to have a key that matches the other key exactly.
  - Simple keys can quickly be brute-forced.
  - Secure key exchange can be an issue.
- DES is insecure, use AES-CBC. RC4 is insecure, use AES-CTR/GCM.

Symmetric encryption algorithms are historically important and are still important today. They are wicked fast due to their construction. Many symmetric ciphers can be implemented in hardware. Modern CPUs now have AES-specific instructions, which make encryption using AES very fast indeed.

The main problems with symmetric cryptosystems are related to the keys: key distribution and key management. Weak cryptosystems like DES and RC4 are vulnerable to cryptanalytic attacks due to small keyspace, weak keys, or improper use. To be fair, improper use of crypto is a deadly sin of software design, no matter what magic crypto fairy dust you use. Another problem with symmetric cryptosystems has to do with authentication. Just because you can decrypt it doesn't mean it actually came from someone with the key.

If you ever find yourself at the table when discussing options for encryption algorithms to use or support, advocate against weak crypto-like DES and RC4 and push for strong crypto like AES. If you need a block cipher, use CBC mode. If you need a stream cipher, use CTR or GCM mode.

Slide 12



Principles of Computer Security, Fifth Edition

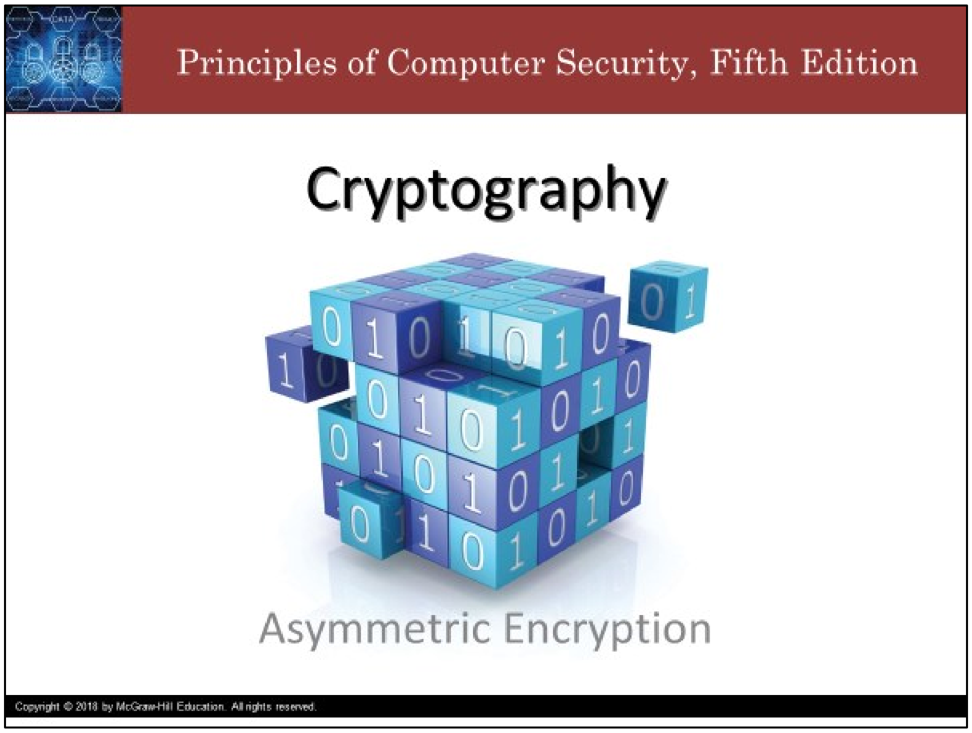
Attribution

- The course slides are based on slides developed by the textbook authors, W. “Art” Conklin, Ph.D. (University of Houston), and Greg White, Ph.D. (University of Texas at San Antonio).
- These slides remain the intellectual property protected by US Copyright law of the authors and the textbook company.
- There have been some changes to the slide deck.

Thank you and take care.

Cryptography: Asymmetric Encryption

Slide 1



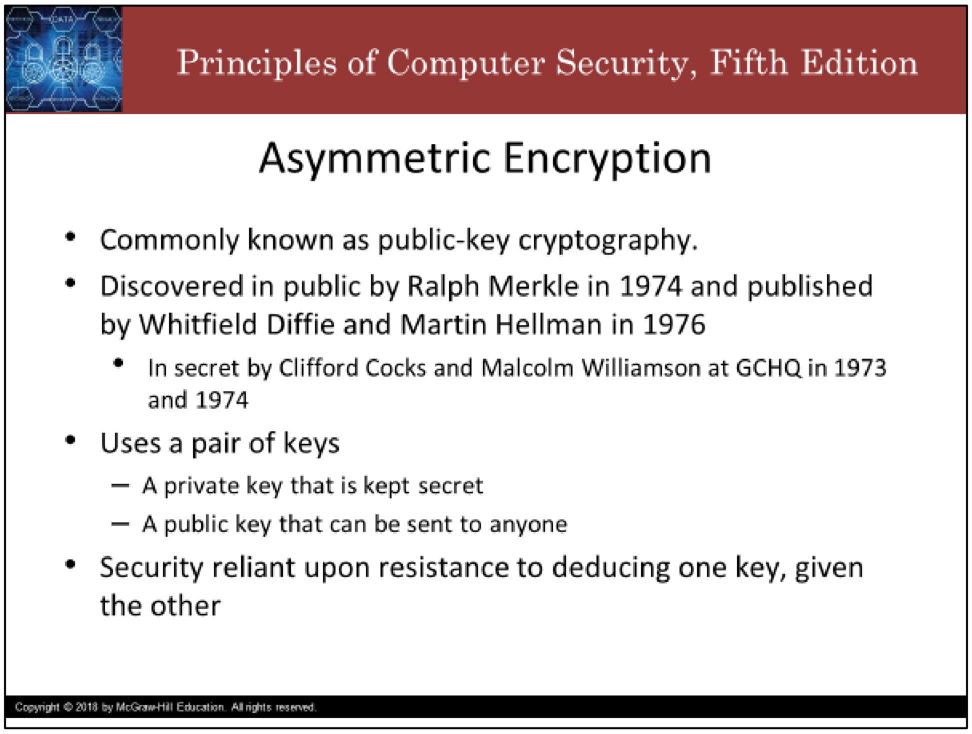
Principles of Computer Security, Fifth Edition

Cryptography

Asymmetric Encryption

Howdy! In this video, we discuss asymmetric encryption.

Slide 2



Principles of Computer Security, Fifth Edition

Asymmetric Encryption

- Commonly known as public-key cryptography.
- Discovered in public by Ralph Merkle in 1974 and published by Whitfield Diffie and Martin Hellman in 1976
  - In secret by Clifford Cocks and Malcolm Williamson at GCHQ in 1973 and 1974
- Uses a pair of keys
  - A private key that is kept secret
  - A public key that can be sent to anyone
- Security reliant upon resistance to deducing one key, given the other

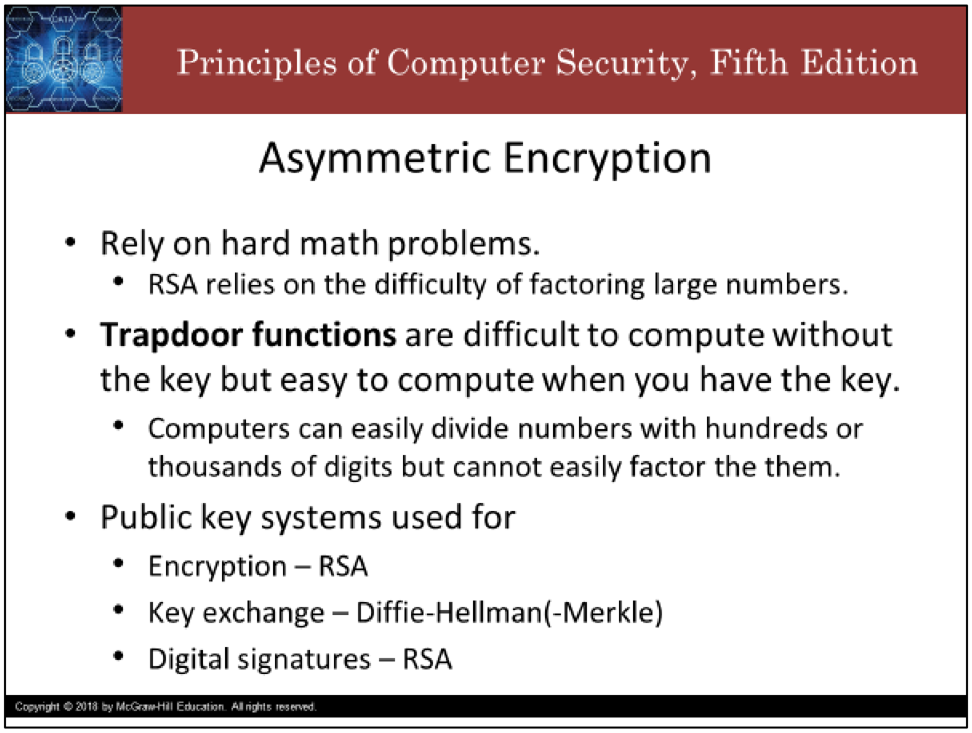
Asymmetric cryptography answers the question of how to securely communicate with someone you've never met and with whom you do not share a key. The chief hurdle in symmetric cryptography was how to share the secret key. With asymmetric cryptography, it is possible to perform the critical key agreement step over an open and insecure channel and end up with a shared secret that only the two endpoints know. It also enables things like digital signatures, which are foundational to eCommerce and support non-repudiation.

The reason it's called asymmetric cryptography is that the endpoints have asymmetric information (rather than the symmetric information of a shared secret). The more common name is public-key cryptography. The asymmetry is that users have a public key, which is public and anyone can know it, and a private key, which is tied to their identity, and only they should ever know it.

Asymmetric cryptography was invented or discovered (depending on your particular philosophy) in the 1970s. In public, it was first discovered by Ralph Merkle in 1976 as part of a class project while he was an undergraduate student. Whitfield Diffie and Martin Hellman built on Merkle's original idea to create a method of public-key agreement which allows the users to establish a shared secret-key over an authenticated (but not confidential) communications channel without using a prior shared secret. It was revealed in 1997 that cryptographers at GCHQ (the NSA of the UK) actually discovered asymmetric crypto first (but not by much).

Like all forms of cryptography, the security of the system rests in the key. Since public-key crypto uses a pair of keys, one of which is public, the security of public-key crypto rests in not just the secrecy or complexity of the private key but also on the computational infeasibility of compromising the private key given the public key.

Slide 3



Principles of Computer Security, Fifth Edition

Asymmetric Encryption

- Rely on hard math problems.
  - RSA relies on the difficulty of factoring large numbers.
- **Trapdoor functions** are difficult to compute without the key but easy to compute when you have the key.
  - Computers can easily divide numbers with hundreds or thousands of digits but cannot easily factor the them.
- Public key systems used for
  - Encryption – RSA
  - Key exchange – Diffie-Hellman(-Merkle)
  - Digital signatures – RSA

Asymmetric cryptosystems rely on the assumption that some math problems are really hard. One example is factoring products of primes.

For example, given a prime number, say 293, and another prime, say 307, it is an easy function to multiply them together to get 89,951.

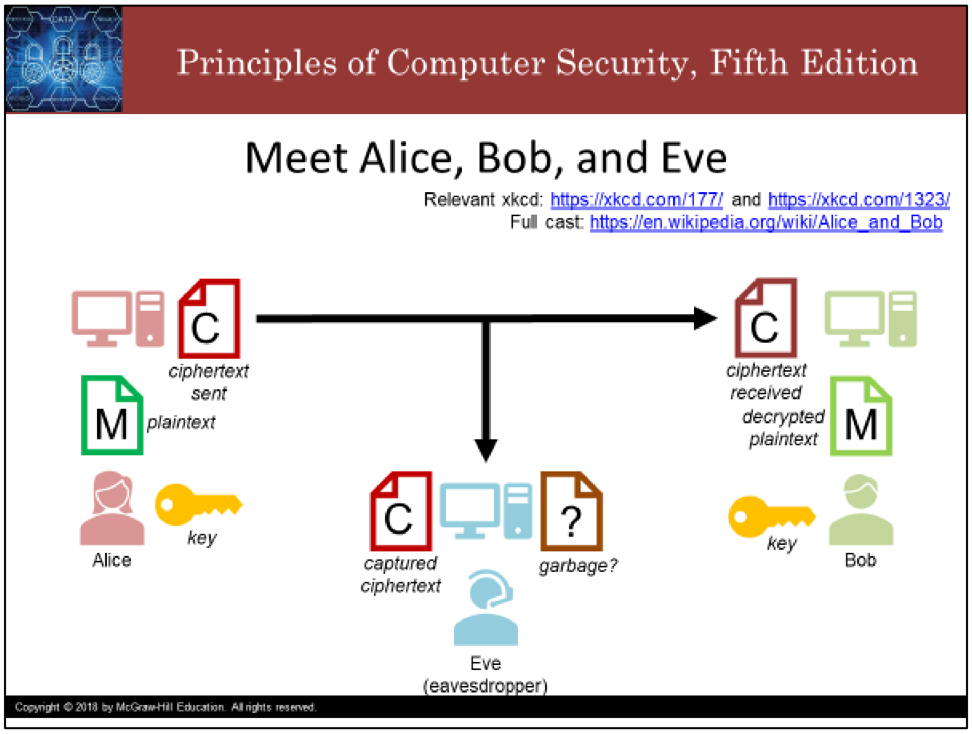
However, given 89,951, it is not so simple to find the factors 293 and 307 unless you know one of them already.

When the numbers are quite big, on the order of hundreds or thousands of bits, as they usually are for asymmetric crypto, the problem of factoring becomes intractable.

This is an example of a trapdoor function. **Trapdoor functions** are easy to compute when you have the key but difficult to compute without it.

Public key systems are used for encryption, key exchange, and digital signatures. RSA, Diffie-Hellman, elliptic curve cryptography, and ElGamal are all popular asymmetric protocols. In this video, we will look at Diffie-Hellman and RSA.

Slide 4



Principles of Computer Security, Fifth Edition

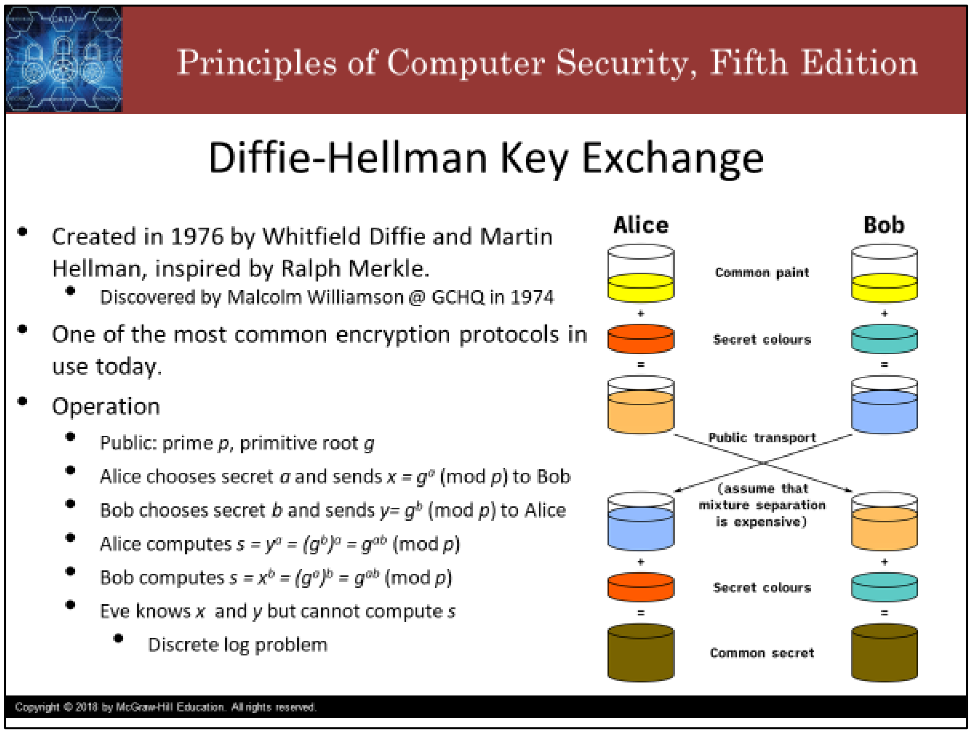
Meet Alice, Bob, and Eve

Relevant xkcd: https://xkcd.com/177/ and https://xkcd.com/1323/
Full cast: https://en.wikipedia.org/wiki/Alice_and_Bob

When cryptographers talk about cryptosystems, they talk a lot about Alice and Bob. Alice and Bob are fictional characters who serve as placeholders in a protocol to aid in discussion. It saves a lot of time and potential confusion compared to saying somewhat awkward things, "the subject which is sending" or "Person A." Alice and Bob don't have to be humans; they can be computers or processes. There are other characters, too, like Eve, the eavesdropper. Alice and Bob want to communicate privately; Eve wants to listen in. Eve knows everything about the system and can see all the ciphertext, but she does not have access to the key (which could be symmetric or asymmetric). Ideally, we want Eve to be able to learn nothing about the plaintext by observing, or, even, interacting with, the ciphertext that Alice sends to Bob.

The cast also includes popular characters like Trent, the trusted entity, Mallory, the malicious user, Trudy, the intruder, and more! Links to the relevant xkcd's and Wikipedia article are on the slide. [xkcd: https://xkcd.com/177/ and https://xkcd.com/1323/ Full cast: https://en.wikipedia.org/wiki/Alice_and_Bob]

Slide 5



Principles of Computer Security, Fifth Edition

Diffie-Hellman Key Exchange

- Created in 1976 by Whitfield Diffie and Martin Hellman, inspired by Ralph Merkle.
  - Discovered by Malcolm Williamson @ GCHQ in 1974
- One of the most common encryption protocols in use today.
- Operation
  - Public: prime *p*, primitive root *g*
  - Alice chooses secret *a* and sends $x = g^a \pmod p$ to Bob
  - Bob chooses secret *b* and sends $y = g^b \pmod p$ to Alice
  - Alice computes $s = y^a = (g^b)^a = g^{ab} \pmod p$
  - Bob computes $s = x^b = (g^a)^b = g^{ab} \pmod p$
  - Eve knows *x* and *y* but cannot compute *s*
    - Discrete log problem

Diffie-Hellman key exchange, which should be called Diffie-Hellmen-Merkle, is one of my favorite algorithms and is one of, if not _the_ one, most used public-key algorithm.

Here's how it works:

Alice and Bob agree on a particular pair of integers, p a really big prime and g a small integer which is a primitive root modulo p (all that means is that exponentiation g mod p generates all values from 1 to p-1). NIST has published 6 of these pairs, with p ranging from 1536 bits up to 8192 bits. g is the prime number 2 for all of them.

Then, both Alice and Bob choose large secret random integers and raise g to the power of their secret random integer, mod p

Then, they send the result to each other. It's OK if Eve, the eavesdropper, observes these messages.

Once they receive the message from the other, Alice and Bob raise the value they received to the power of their secret integer, mod p.

By the power of…. powers…. They have mathemagically computed the same number! Tada!

They promptly destroy their secret random integers they choose at the start so that the new shared secret will not be compromised.

They can now use this shared secret as the secret key for a symmetric cryptosystem.

Eve cannot do this, though. She has g^a mod p and g^b mod p, but there's no way for her to combine them to get g^ab mod p, like Alice and Bob.

Eve's only hope is to find out what Alice and Bob's secret random integers were. In order to do this, she must solve the discrete log problem, which is believed to be computationally infeasible. The best Eve can do here is to use brute force, but that will get her nowhere because Alice and Bob used numbers which have thousands of bits.

Diffie-Hellman is still in wide use. It remains very effective because of the nature of what it is protecting—a temporary automatically generated secret key that is good only for a single communication session.

2048-bit MODP Group

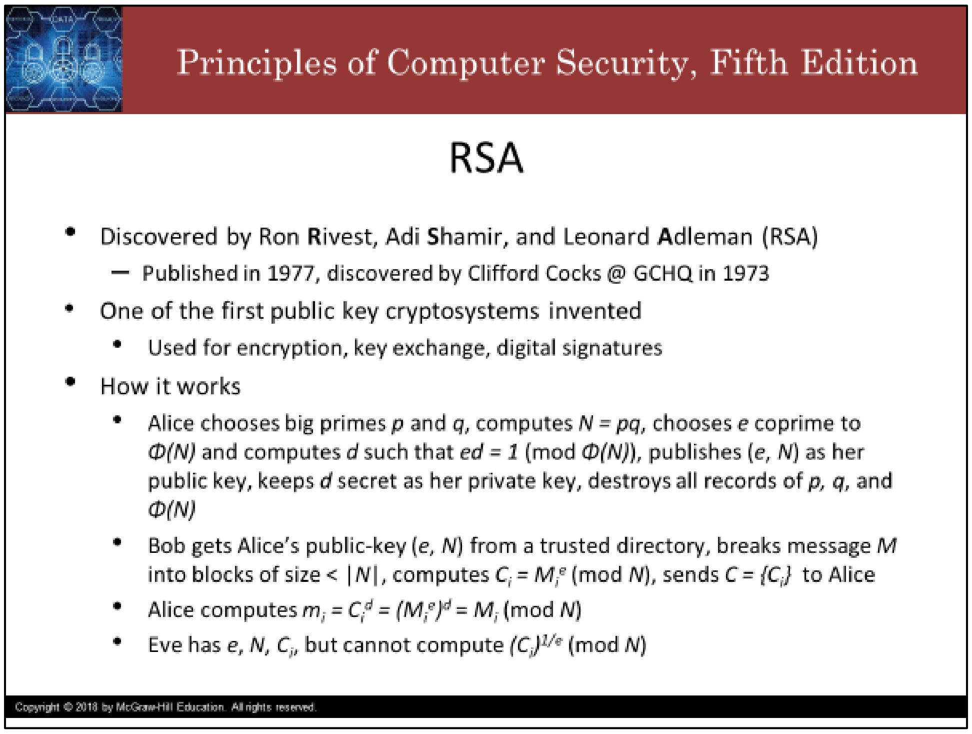
This prime is: 2^2048 - 2^1984 - 1 + 2^64 * { [2^1918 pi] + 124476 }

Its hexadecimal value is: FFFFFFFF FFFFFFFF C90FDAA2 2168C234 C4C6628B 80DC1CD1 29024E08 8A67CC74 020BBEA6 3B139B22 514A0879 8E3404DD EF9519B3 CD3A431B 302B0A6D F25F1437 4FE1356D 6D51C245 E485B576 625E7EC6 F44C42E9 A637ED6B 0BFF5CB6 F406B7ED EE386BFB 5A899FA5 AE9F2411 7C4B1FE6 49286651 ECE45B3D C2007CB8 A163BF05 98DA4836 1C55D39A 69163FA8 FD24CF5F 83655D23 DCA3AD96 1C62F356 208552BB 9ED52907 7096966D 670C354E 4ABC9804 F1746C08 CA18217C 32905E46 2E36CE3B E39E772C 180E8603 9B2783A2 EC07A28F B5C55DF0 6F4C52C9 DE2BCBF6 95581718 3995497C EA956AE5 15D22618 98FA0510 15728E5A 8AACAA68 FFFFFFFF FFFFFFFF

The generator is: 2

(https://datatracker.ietf.org/doc/rfc3526/?include_text=1)

Slide 6



Principles of Computer Security, Fifth Edition

RSA

- Discovered by Ron **R**ivest, Adi **S**hamir, and Leonard **A**dleman (RSA)
  - Published in 1977, discovered by Clifford Cocks @ GCHQ in 1973
- One of the first public key cryptosystems invented
  - Used for encryption, key exchange, digital signatures
- How it works
  - Alice chooses big primes p and q, computes $N = pq$, chooses e coprime to $\Phi(N)$ and computes d such that $ed = 1$ (mod $\Phi(N)$), publishes (e, N) as her public key, keeps d secret as her private key, destroys all records of p, q, and $\Phi(N)$
  - Bob gets Alice's public-key (e, N) from a trusted directory, breaks message M into blocks of size < $|N|$, computes $C_i = M_i^e$ (mod N), sends $C = \{C_i\}$ to Alice
  - Alice computes $m_i = C_i^d = (M_i^e)^d = M_i$ (mod N)
  - Eve has e, N, C_i, but cannot compute $(C_i)^{1/e}$ (mod N)

RSA is a public-key cryptosystem that is widely used. It was also one of the first. It was described in public in 1977 and patented by MIT in 1983. But, actually, it was described in public by someone else first. Martin Gardner wrote an article in Scientific American about the RSA algorithm (named after its creators) in August 1977. The patent for RSA was filed in December 1977. The paper authored by Rivest, Shamir, and Adleman, which described RSA, was published in the Communications of the ACM in February 1978, which is like the Nature or Science of computing, the toppest tier venue for us. But a decade before the patent was granted and four years before RSA was described by the Americans, Clifford Cocks at GCHQ had already discovered it. Had that fact been know, the patent on RSA would not have been granted since R, S, and A were not the first to discover or describe the algorithm which now bears their names. But, the patent was granted because GCHQ didn't tell anybody (except possibly the NSA) about what they already knew. 17 years later, when the patent was about to expire, RSA Security (a company founded by the American "inventors' of RSA, which later sold to EMC for 2.1 billion dollars… crypto is big business, y'all) released the algorithm into the public domain. So, now it is ours. And we love it.

RSA can do encryption, key exchange, and digital signatures, which are the three main functions required by public-key cryptography, which is why we say RSA is a cryptosystem rather than just one algorithm.

RSA uses the same kinds of math as Diffie-Hellman. The main operation, which is used for all the features, is modular exponentiation, the same as in Diffie-Hellman. But first, Alice needs to generate her key pair. This is done by picking two very large and distinct prime numbers. The standard for RSA right now is to pick primes that are at least 1024 bits. Alice then computes the value N as the product of the primes p and q. N is one-half of Alice's public key. Due to the hardness of factoring integers, finding p and q given N is intractable. The next step in key generation is to pick an integer e, which is the other half of the public key and is named e because it is the encrypting key, and compute the integer d, which is the modular inverse of e mod the totient function of N (were this a math class, we'd go deeper, but suffice it to say this is an easy thing to do given p and q). d is the private key and must be kept secret. It is named d because it is the decryption key. Typically, e is chosen to be a small number or a number with lots of consecutive binary 0s in the middle, which makes encryption a faster process. This typically has the effect of making d an obnoxiously big number, which slows down decryption, but the tradeoff is worth it since brute-forcing an obnoxious number is much harder than brute-forcing an easy or predictable number. Whatever d turns out to be is what it is, and it must be kept secret. At this point, Alice can publish her public key consisting of the values for the encryption key e and the modulus N. She also destroys all records of p, q, and phi(N) since they are the keys to the trapdoor and their compromise would compromise the whole system. So into the void they go. To send a message to Alice, Bob gets Alice's public key from some trusted source and breaks his message into blocks whose size is less than the size of the modulus N (so, Bob will use RSA as a block cipher in ECB mode). For each block of plaintext, Bob treats it as a number and raises it to the power of Alice's public key e mod N. The sequence of ciphertext block obtained in this way are sent to Alice. Alice can decrypt Bob's message by taking each ciphertext block and raising it to the power of her private key d. Since the ciphertext is M to the e, raising it to the d-th power is like raising M to power of e times d. This is why we computed d to be the modular inverse of e. E times d is equivalent to 1, so the whole exponent will fall away and leave only the plaintext behind. Alice is the only one who can do this operation. Not even Bob, who knows what the plaint text is, can perform this operation without Alice's secret private key.

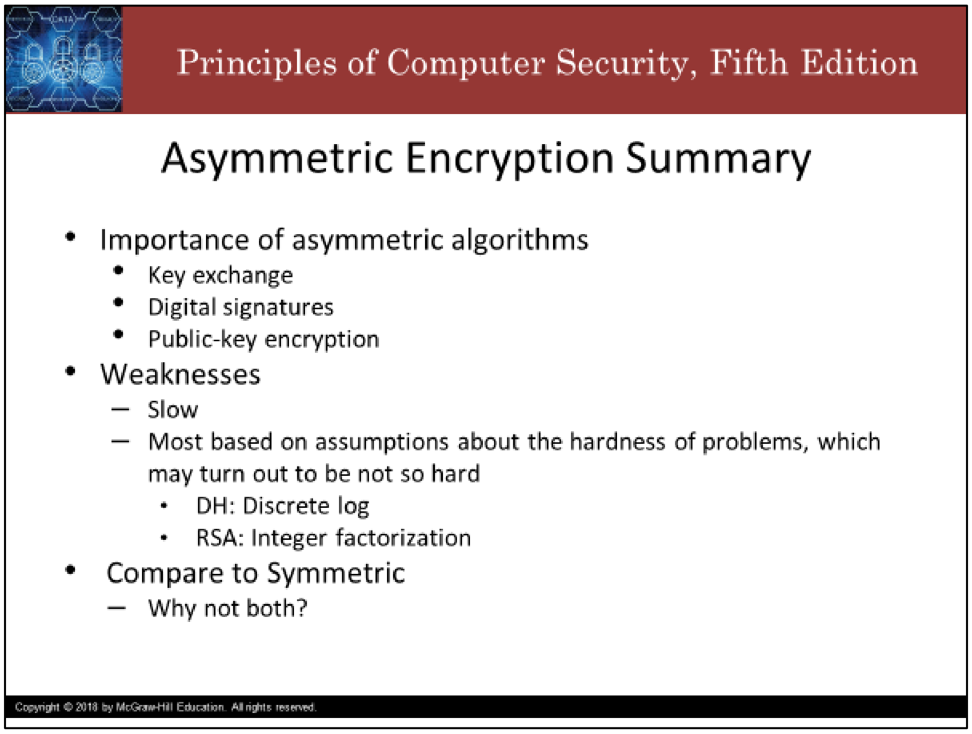
For this reason, we say that Alice's key-pair is bound to her identity. This property, that Alice is the only one who can decrypt message encrypted with her public key, is useful for digital signatures. Alice can sign a document by encrypting it with her private key. Usually, she will do this to a hash of the document since hashes are small and encrypting with RSA is expensive. Anyone can verify Alice's signature by decrypting the ciphertext with Alice's public key to recover the original document or hash. Note that decrypting in RSA is the exact same operation as encrypting, just with a different key. Going back to encryption, Eve has Alice's public key and Bob's ciphertext, but she would need to compute the e-th root of the ciphertext in order to decrypt it without Alice's private key. This problem is as at least as hard as factoring the modulus N, which is believed to be computationally infeasible for large enough N. This is why RSA keys are so big: we need to make Eve's work factor as high as possible.

Asymmetric methods, like RSA, are significantly slower than symmetric methods and thus are typically not suitable for encryption long conversation, such as would be required for a secure HTTP connection.

Instead, asymmetric encryption is used to exchange symmetric keys. Bob can encrypt an AES key using Alice's public key and send it to her in less than a single block (AES keys are 128 – 256 bits, and RSA blocks with a 2048-bit key are 2048 bits long). Once Alice and Bob have a shared key, they can use the much faster symmetric encryption algorithm to secure their communication.

The best public cryptanalysis of RSA is still impractical on a classical system. The fastest factoring algorithm just isn't fast enough yet. The largest key which has been broken is 829 bits. This is why the recommended key size for RSA is at least 2048 bits. As algorithms can better, and computational power grows, larger RSA keys will fall. But, every bit of key increases the work factor by up to a factor of 2, so the 1200 bits of distance between what is possible now and a 2048-bit key gives plenty of security. But, there's more to the story. Breaking RSA on a classical computer is infeasible, but quantum computers are a different beast. There is a quantum factoring algorithm named Shor's algorithm for its discoverer Peter Shor who discovered it in 1994 and published it in 1997, and Shor's algorithm runs in polynomial-time. Without getting too much into the wild world of computational complexity, what this means is that a quantum computer can crack RSA in the blink of an eye. This is a reality that we will have to face in the coming years as quantum computing technology continues to improve. For that reason, cryptographers have been developing post-quantum crypto algorithms for the last 20 or so years. So, there's nothing to fear, but it'll be a bittersweet day when RSA and Diffie-Hellman finally retire. But, when they do, like so many other cryptosystems before them, they will be well-loved by students of cryptology for a very long time.

Slide 7



Principles of Computer Security, Fifth Edition

Asymmetric Encryption Summary

- Importance of asymmetric algorithms
  - Key exchange
  - Digital signatures
  - Public-key encryption
- Weaknesses
  - Slow
  - Most based on assumptions about the hardness of problems, which may turn out to be not so hard
    - DH: Discrete log
    - RSA: Integer factorization
- Compare to Symmetric
  - Why not both?

Ever since its discovery in the 1970s, asymmetric algorithms have rocked the world of cryptography. Diffie and Hellman opened their earth-shattering 1976 paper, which introduced the first asymmetric algorithm to be publicly described, “We stand today on the brink of a revolution in cryptography.” So right, they were.

Asymmetric algorithms solve one of the main problems, if not _the_ main problem, with symmetric cryptography: key exchange. With asymmetric algorithms, Alice and Bob can securely agree on a shared secret over an insecure channel. Shortly after the Diffie-Hellman key exchange algorithm was published, the RSA cryptosystem was described, which included encryption and digital signatures. Digital signatures is a very useful cryptographic primitive made possible by asymmetric algorithms, allowing users to authenticate and verify identity with a high level of security. Asymmetric encryption, or public-key encryption, enables two people who have never met and have never communicated before to engage in secure communication without having to negotiate a key, i.e., Bob can encrypt a message that only Alice and can read and the time Alice ever learns of Bob’s existence is when she receives a message from him that is encrypted just for her.

Both symmetric and asymmetric cryptosystems have their advantages and disadvantages.

Symmetric encryption tends to be faster, is less computationally involved, and is better for large or lengthy transfers.

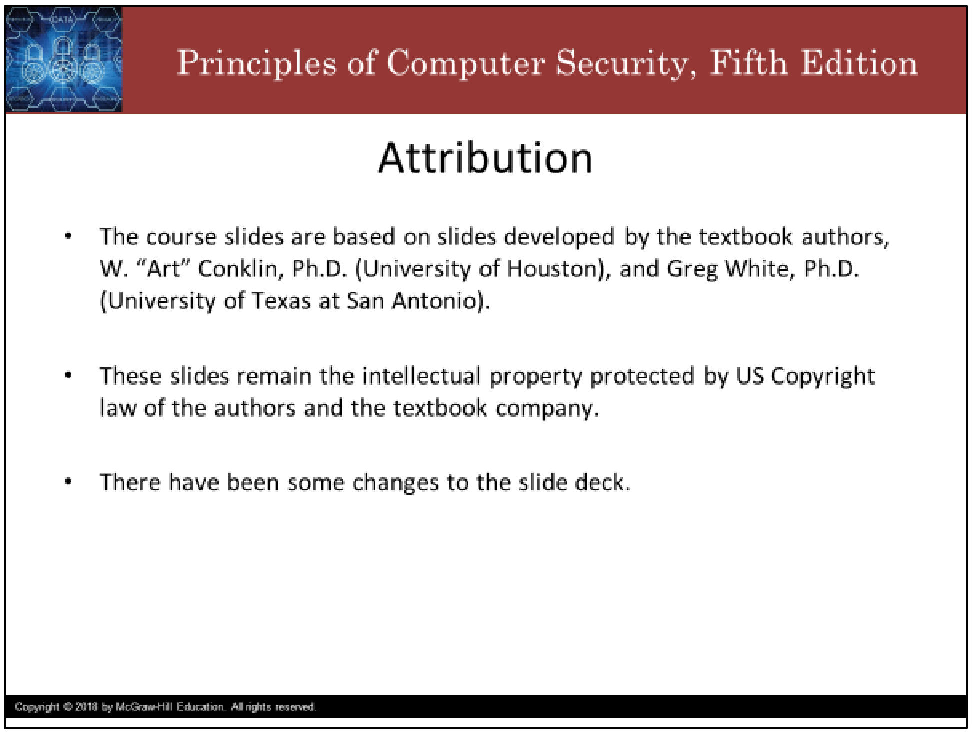
But, it suffers from the key exchange and key management problems in that keys must be protected from unauthorized parties, and every pair of people needs to have their own shared key, which makes for a lot of keys.

Asymmetric methods resolve the key exchange and key management issues with key exchange protocols and public keys.

But, they have significant computational complexity that makes them impractical for large or length transfers.

The best choice is to use both: Use asymmetric encryption to pass a shared symmetric key. By adding in ephemeral key exchange, you can achieve perfect forward secrecy. And, digital signatures, which are not practical without asymmetric methods are useful to add a layer of integrity on top of the confidentiality of the encryption.

Slide 8



Principles of Computer Security, Fifth Edition

Attribution

- The course slides are based on slides developed by the textbook authors, W. "Art" Conklin, Ph.D. (University of Houston), and Greg White, Ph.D. (University of Texas at San Antonio).
- These slides remain the intellectual property protected by US Copyright law of the authors and the textbook company.
- There have been some changes to the slide deck.

Copyright © 2018 by McGraw-Hill Education. All rights reserved.

Thank you and take care.