APPLIED CRYPTOGRAPHY

CO 487/687

©*Alfred Menezes*

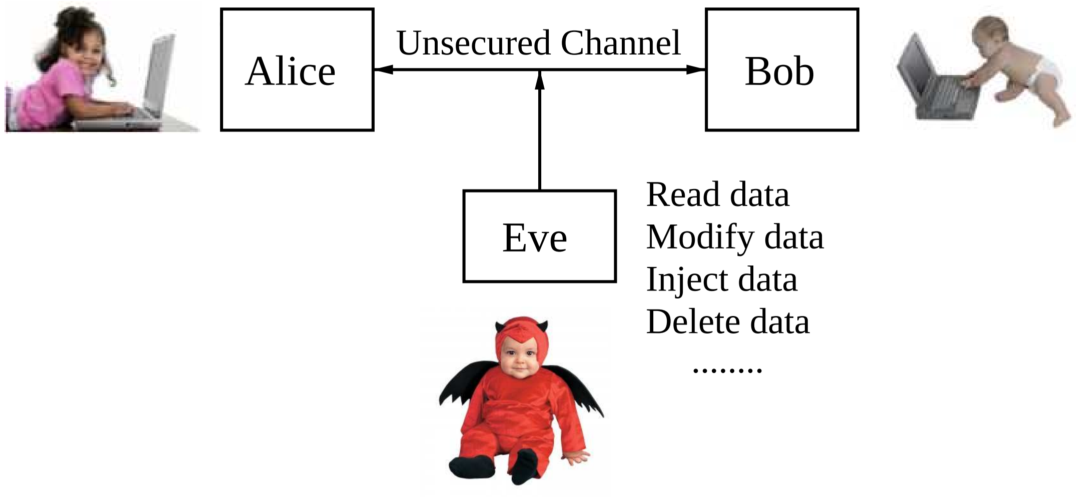
University of Waterloo

Winter 2017

COURSE PREVIEW

What is Cryptography?

Cryptography is about securing communications in the presence of *malicious* adversaries.



Alice

Unsecured Channel

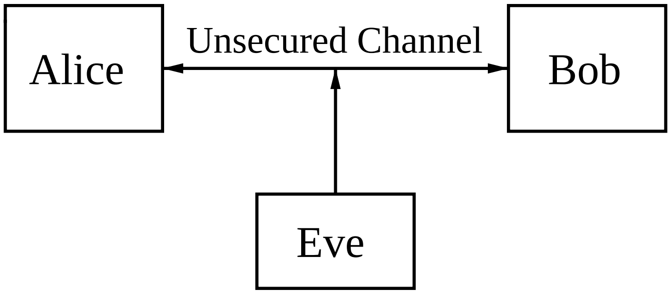
Bob

Eve

Read data
Modify data
Inject data
Delete data
....

Fundamental Goals of Cryptography

- *Confidentiality*: Keeping data secret from all but those authorized to see it.
- *Data integrity*: Ensuring data has not been altered by unauthorized means.
- *Data origin authentication*: Corroborating the source of data.
- *Non-repudiation*: Preventing an entity from denying previous commitments or actions.



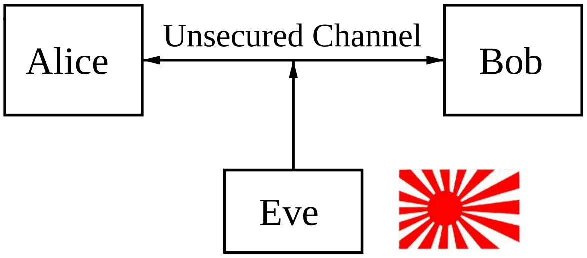
Alice

Unsecured Channel

Bob

Eve

WWII: Navajo Code Talkers

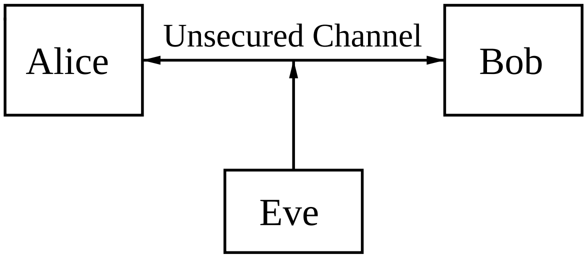


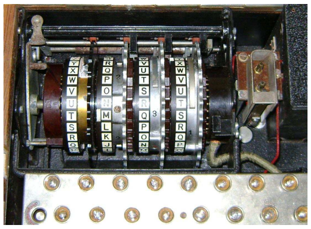
Native Americans from the Navajo tribe were employed to radio messages between the battlefield and command central.
The Navajo language belongs to the Na-Dene family of languages, which has no link with any Asian or European language. The language is also unintelligible to other Native American tribes.

- "... weird succession of guttural, nasal, tongue-twisting sounds... we couldn't even transcribe it, much less crack it".

Fighter plane	Hummingbird	Da-he-tih-hi
Observation plane	Owl	Tas-chizzie
Bombs	Eggs	A-ye-shi

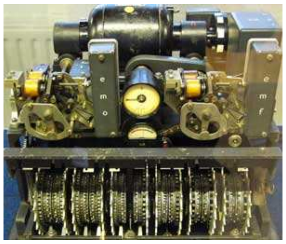
WWII: Enigma Machine

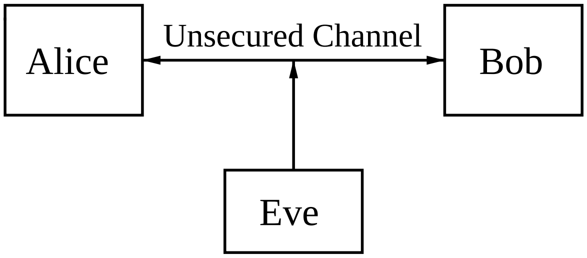


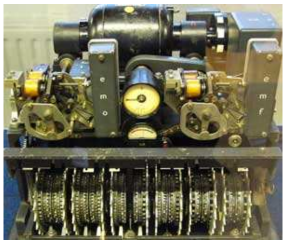


Alan Turing

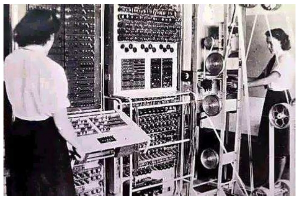
WWII: Lorenz Machine







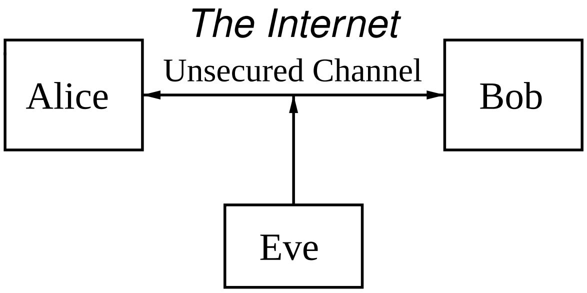
Bill Tutte



Colossus

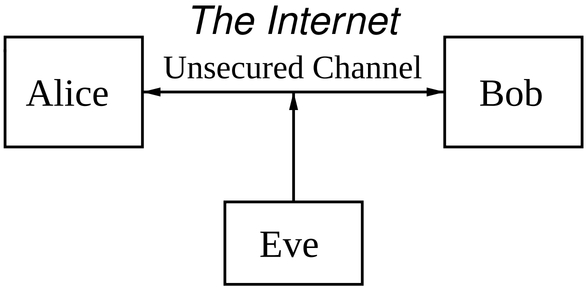
See: `http://billtuttememorial.org.uk/`

Online Shopping



amazon

Automatic Software Upgrades



The Internet

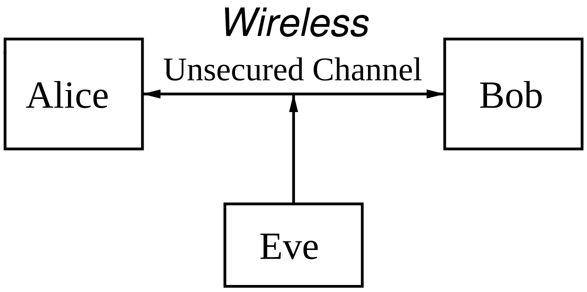
Unsecured Channel

Alice

Bob

Eve

Cell Phone Service



Wireless

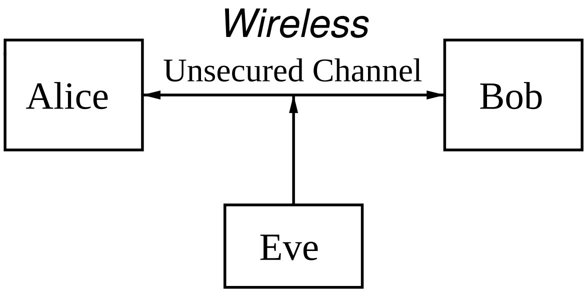
Unsecured Channel

Alice

Bob

Eve

Smart Meters



Wireless

Alice

Unsecured Channel

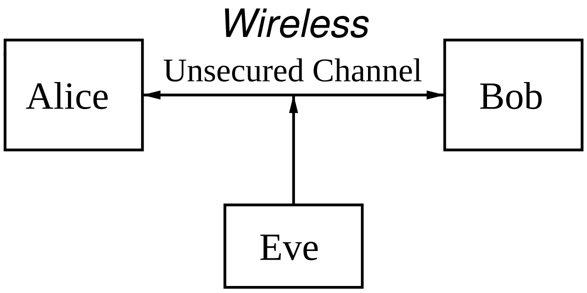
Bob

Eve

Waterloo North Hydro Inc.

Wireless Pacemakers

Pacemaker



Wireless

Alice

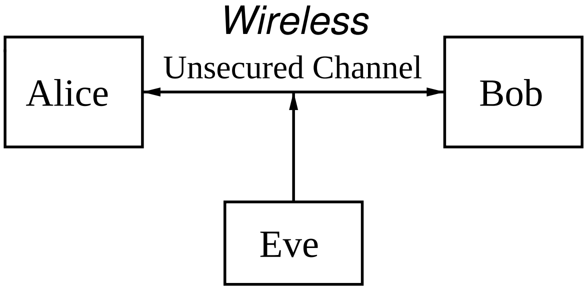
Unsecured Channel

Bob

Eve

Doctor

Vehicle-to-Vehicle Communications



Wireless

Unsecured Channel

Alice

Bob

Eve

Communicating Parties

Alice	*Bob*	*Communications channel*
person	person	telephone cable
person	person	cellular network
person	web site	internet
BlackBerry	service provider	cellular network
smart card	bank machine	financial network
music player	iTunes	internet
smart meter	energy provider	wireless
your car brakes	another car	wireless
military commander	satellite	space

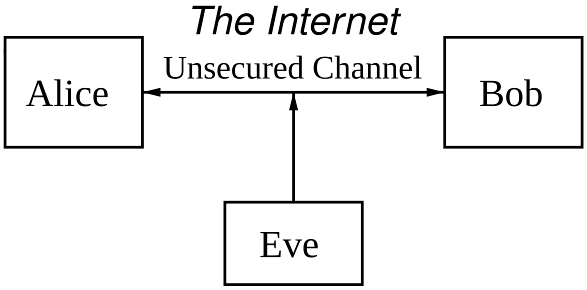
Adversaries

National Security Agency

Edward Snowden

The Adversary is Malicious

Secure Web Transactions

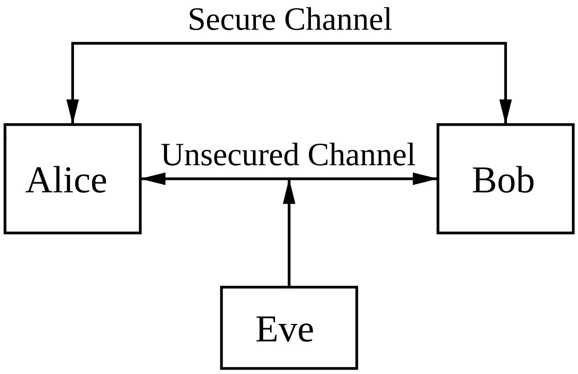


Secure Sockets Layer (SSL): The cryptographic protocol used by web browers for secure web transactions such as credit card payments; also for secure access to gmail, hotmail, facebook etc.

SSL is used to assure an individual user (called a *client*) of the authenticity of the web site (called the *server*) he or she is visiting, and to establish a secure communications channel for the remainder of the session.

Symmetric-Key Cryptography

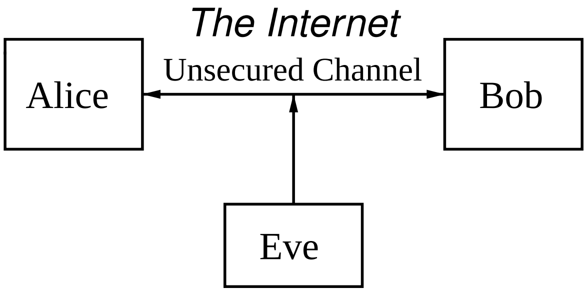
Symmetric-key cryptography: Communicating parties a priori share some *secret* information k, called a *key*.



Confidentiality: Use an *encryption scheme* (such as *AES*).

Authentication (i.e., data origin authentication and data integrity): Use a *MAC scheme* (such as *HMAC*).

Secure Web Transactions (2)

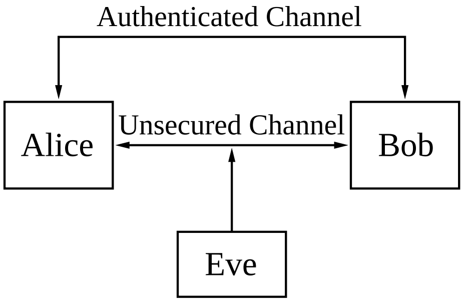


The *client* and the *server* can engage in secure communications by encrypting their messages with *AES* and authenticating the resulting ciphertexts with *HMAC*.

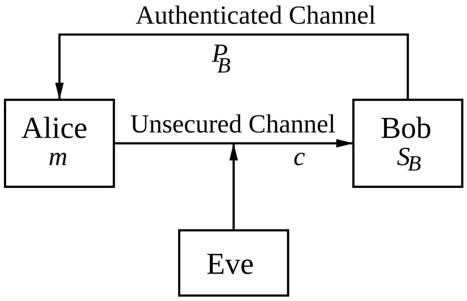
But how do they select their shared secret key k?

Public-Key Cryptography

Public-key cryptography: Communicating parties a priori share some *authenticated* (but non-secret) information.

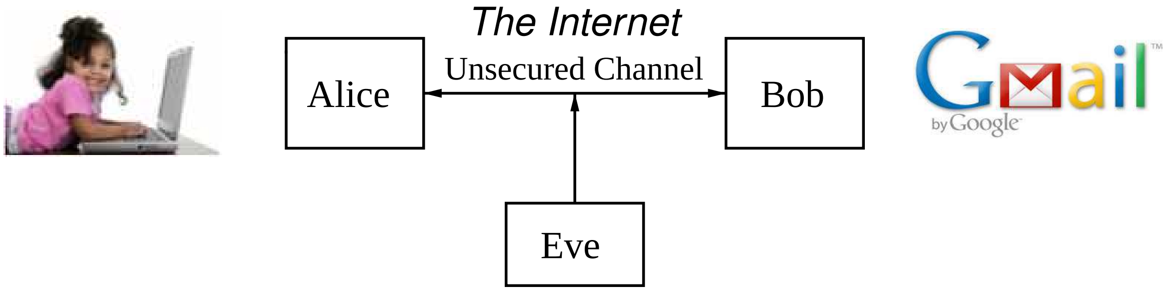


Public-Key Encryption (For Confidentiality)



- To *encrypt* a secret message m for Bob, Alice does:
  1. Obtain an authentic copy of Bob’s public key P_B.
  2. Compute $c = E(P_B, m)$; E is the encryption function.
  3. Send c to Bob.
- To *decrypt* c, Bob does:
  1. Compute $m = D(S_B, c)$, where D is the decryption function and S_B is Bob’s private key.

Secure Web Transactions (3)

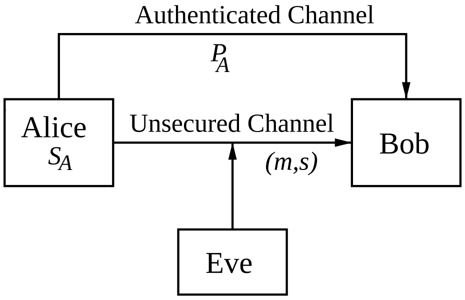


So, to share a secret key, the client selects the secret *session key* k, and encrypts it with the server’s *RSA* public key. Then only the server can decrypt the resulting ciphertext to recover the session key.

But how does the client obtain an *authentic* copy of the server’s RSA public key?

Digital Signatures

(for data integrity, data origin authentication, non-repudiation)



- To *sign* a message m, Alice does:
  1. Compute $s =$ Sign(S_A, m), where S_A is Alice's private key.
  2. Send m and s to Bob.
- To *verify* Alice's signature s on m, Bob does:
  1. Obtain an authentic copy of Alice's public key P_A.
  2. Accept if Verify$(P_A, m, s) =$ Accept.

The SSL Protocol

1. When a client first visits a secured web page, the server transmits its *certificate* to the client.
   - The certificate contains the server's identifying information (e.g., web site name and URL) and RSA public key, and the RSA signature of a *certifying authority*.
   - It is assumed that the certifying authority has carefully verified the server's identity before issuing the certificate.
   - *Verisign* (`www.verisign.com`) has issued more than 1,000,000 certificates for web servers.

The SSL Protocol (2)

2. Upon receipt of the certificate, the client *verifies* the signature using the certifying authority's public key, which is pre-installed in the browser. A successful verification confirms the *authenticity* of the server and of its RSA public key.
3. The client selects a random *session key* k, *encrypts* it with the server's RSA public key, and transmits the resulting ciphertext to the server.
4. The server *decrypts* the session key, which is then used with symmetric-key schemes to *encrypt* (usually with AES) and *authenticate* (usually with HMAC) all sensitive data exchanged for the remainder of the session.

The SSL Protocol (3)

5. The establishment of a secure link is indicated by a *closed padlock* in the Internet Explorer and Chrome browsers. Clicking on this icon reveals the server's certificate and information about the certifying authority.

6. Examples of secured web sites are

```
https://www.gmail.com
https://www.facebook.com
https://www.amazon.com
https://www.cibconline.cibc.com
```

The SSL Protocol (4)

SSL is one of the most successful security technologies every deployed.

Why? Because users do not have to take any special action:

- to install it.
- to turn it on.
- to be secure.

But is SSL *really* secure?

The SSL Protocol (5)

There are many potential security vulnerabilities:

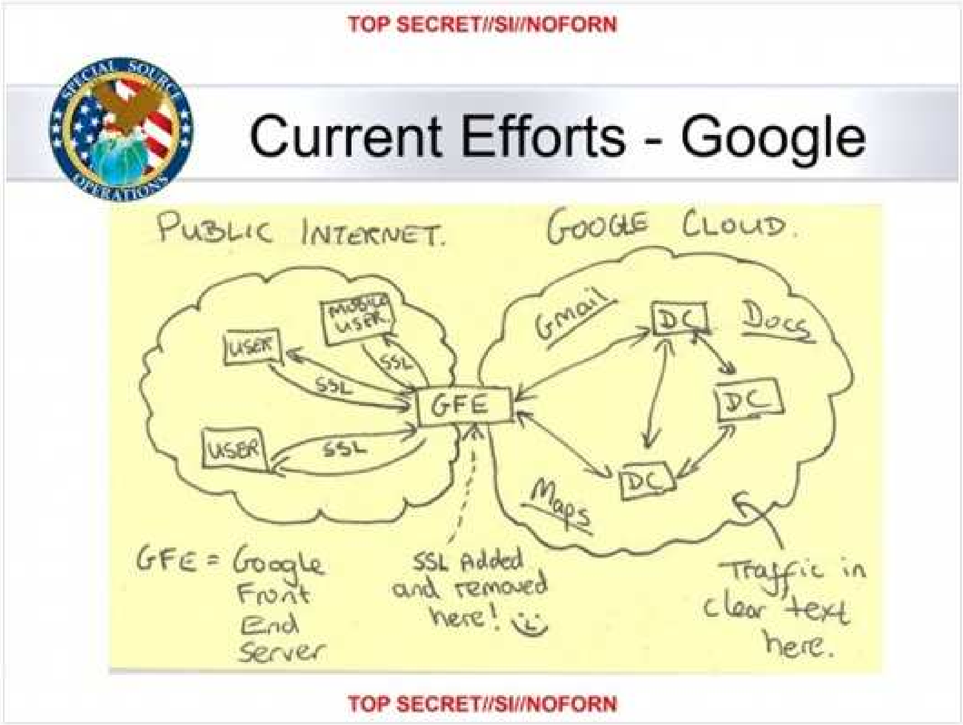
1. The crypto is weak (e.g., AES, RSA, HMAC, RC4)
2. *Quantum attacks* on the underlying cryptography.
3. Weak random number generation
   - Well-publicized finding in 1995 by Goldberg and Wagner that the random number generator in Netscape was flawed.
   - The National Security Agency's Dual EC DRBG (backdoored pseudorandom bit generator).
4. Issuance of fraudulent certificates
   - In 2001, Verisign erroneously issued two Class 3 code-signing certificates to someone masquerading as a Microsoft representative.
   - Mistake due to human error.

The SSL Protocol (6)

5. Software bugs (see: *Heartbleed*).
6. Malicious code inserted in web browser software by disgruntled programmers.
   - ► A browser is a complicated piece of software. How can assurances be provided that software does what it is supposed to?
7. *Phishing* attacks.
8. Most users do not verify the security information by clicking on the padlock.
9. SSL only protects data during transit. It does *not* protect your data when it is collected at the server.
   - ► Many servers store large amounts of credit card data and other personal information.

The SSL Protocol (7)

10. The National Security Agency.



TOP SECRET//SI//NOFORN

Current Efforts - Google

PUBLIC INTERNET.
GOOGLE CLOUD.
USER
MOBILE USER
USER
SSL
SSL
SSL
GFE
Gmail
DC
Docs
DC
DC
Maps
GFE = Google Front End Server
SSL Added and removed here! ☺
Traffic in clear text here.

TOP SECRET//SI//NOFORN

Cryptography in Context

Information security is comprised of the concepts, technical measures, and administrative measures used to protect information assetss from deliberate or inadvertent unauthorized acquisition, damage, disclosure, manipulation, modification, loss, or use.

The real challenge is an engineering one: building *high confidence systems*.

Information Security Includes the Study of:

Computer security

- Security models and policies
- Secure operating systems
- Virus protection
- Auditing mechanisms
- Risk analysis
- Risk management

Network security

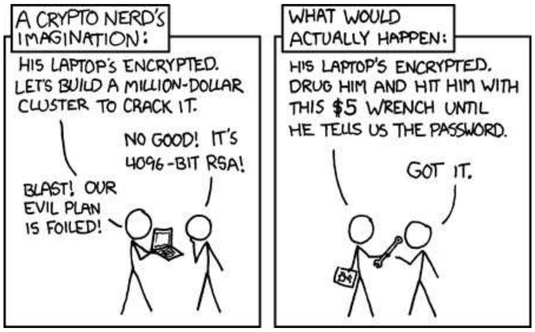
- Internet protocols and their security
- Viruses and worms
- Denial-of-service (DoS) attacks
- Firewalls
- Intrusion detection systems
- Wireless communications

Software security

- Detecting and preventing buffer overflows
- Programming languages and compilers
- Specifying and enforcing security policies
- Digital rights management
- Code obfuscation
- Software tamper resistance
- Trusted computing

Cryptography $\neq$ Security

- Cryptography provides some mathematical tools that can assist with the provision of information security services. It is a *small*, albeit an *essential*, part of a complete security solution.
- This course will focus on cryptography.
- *Security is a chain*
  - Weak links become targets; one flaw is all it takes
  - Cryptography is usually not the weakest link. However, when the crypto fails the damage can be catastrophic.


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

(taken from xkcd.com)

Course Outline

Crypto primitives

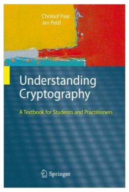
- Symmetric-key encryption
- Hash functions
- Message authentication
- Public-key encryption
- Signature schemes
- Key establishment
- Pseudorandom bit generation

Crypto deployments

- IEEE 802.11
- Electronic payment systems (Bitcoin)
- Key management (PKIs)
- Web security protocols (SSL)
- IPsec (VPNs)
- Anonymity on the Internet

About the Course

- Coverage will favour breadth at the expense of depth
  - For depth, try the recommended and optional readings
  - See also: *CO 485* (Mathematics of Public-Key Crypto)
  - See also: *CS 458* (Computer Security and Privacy)
- This course is not a traditional textbook course!
  - *Attendance* is strongly recommended
  - Exam questions are not very predictable
  - *Your job*: Identify and understand the important (technical and non-technical) *concepts* presented in class.
- Optional text: Understanding Cryptography
  - Available for free download from `http://tinyurl.com/PaarPelzl`
  - Recommended readings are posted on the course web site


Christof Paar
Jan Pelzl
Understanding Cryptography
A Textbook for Students and Practitioners
Springer

About the Course (2)

- Web site: `http://learn.uwaterloo.ca`
  - Slides, assignments & solutions, handouts, sample exams, ...
  - Required, recommended and optional readings
- Evaluation: 5 assignments (20%), Midterm test (30%), Exam (50%)
- Assignments: submitted via Crowdmark
- Midterm test: *March 8* (Wednesday), 7-9 pm
- Policies (*read the course outline*):
  - Class etiquette
  - Office hours, Email queries
  - Collaboration, Deadlines, Grade appeals

SYMMETRIC-KEY ENCRYPTION

Outline

1. Basic Concepts
2. The One-Time Pad
3. Stream Ciphers
4. The RC4 Stream Cipher
5. Block Ciphers
6. The Data Encryption Standard (DES)
7. Multiple Encryption
8. Modes of Operation
9. The Advanced Encryption Standard (AES)

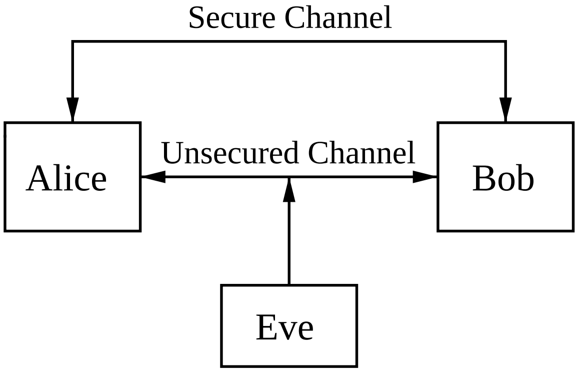
Basic Concepts

Definition: A *symmetric-key encryption scheme (SKES)* consists of:

- M – the plaintext space,
- C – the ciphertext space,
- K – the key space,
- a family of encryption functions, $E_k : M \to C$, $\forall k \in K$,
- a family of decryption functions, $D_k : C \to M$, $\forall k \in K$,

such that $D_k(E_k(m)) = m$ for all $m \in M$, $k \in K$.

Using a SKES to Achieve Confidentiality



1. Alice and Bob agree on a *secret key* $k \in K$ by communicating over the *secure channel*.
2. Alice computes $c = E_k(m)$ and sends the ciphertext c to Bob over the *unsecured channel*.
3. Bob retrieves the plaintext by computing $m = D_k(c)$.

The Simple Substitution Cipher

- M = all English messages.
- C = all encrypted messages.
- K = all permutations of the English alphabet.
- $E_k(m)$: Apply permutation k to m, one letter at a time.
- $D_k(c)$: Apply inverse permutation k^{-1} to c, one letter at a time.

Example:

k =

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
D	N	X	E	S	K	O	J	T	A	F	P	Y	I	Q	U	B	R	Z	G	V	C	H	M	W	L

$m =$ the big dog, $c = E_k($the big dog$) =$ GJS NTO EQO.

Question: Is the simple substitution cipher a secure SKES?

What Does it Mean for a SKES to be Secure?

1. What is the adversary's goal?
2. What are the computational powers of the adversary?
3. How does the adversary interact with the two communicating parties?

- *Security model*: Defines the computational abilities of the adversary, and how she interacts with the communicating parties.
- *Basic assumption*: The adversary knows everything about the SKES, except the particular key k chosen by Alice and Bob. (Avoid security by obscurity!!)

Adversary’s Interaction

- ► Passive attacks:
  - *Ciphertext-only attack*.
  - *Known-plaintext attack*: The adversary also knows some plaintext and the corresponding ciphertext.
- ► Active attacks:
  - *Chosen-plaintext attack*: The adversary can also choose some plaintext and obtains the corresponding ciphertext.
- ► Other attacks (not considered in this course):
  - *Clandestine attacks*: bribery, blackmail, etc.
  - *Side-channel attacks*: monitor the encryption and decryption equipment (timing attacks, power analysis attacks, electromagnetic-radiation analysis, etc.)

Computational Power of the Adversary

- ► *Information-theoretic security*: Eve has infinite computational resources.
- ► *Complexity-theoretic security*: Eve is a ‘polynomial-time Turing machine’.
- ► *Computational security*: Eve has 10,000 Intel Xeon E5-2697 Cores at her disposal. (Eve is “computationally bounded”)

Adversary's Goal

1. Recover the secret key.
2. Systematically recover plaintext from ciphertext (without necessarily learning the secret key).
3. Learn *some* partial information about the plaintext from the ciphertext (other than its length).

- ► If the adversary can achieve 1 or 2, the SKES is said to be *totally insecure* (or *totally broken*).
- ► If the adversary cannot learn any partial information about the plaintext from the ciphertext (except possibly its length), the SKES is said to be *semantically secure*.

Note: Hiding length information is very hard in practice. This topic falls under the heading of *traffic analysis*.

Definition of a Secure SKES

Definition: A symmetric-key encryption scheme is said to be *secure* if it is semantically secure against chosen-plaintext attack by a computationally bounded adversary.

To *break* a symmetric-key encryption scheme, the adversary has to accomplish the following:

1. The adversary is given a challenge ciphertext c (generated by Alice or Bob using their secret key k).
2. During its computation, the adversary can select plaintext and obtains (from Alice or Bob) the corresponding ciphertext.
3. After a feasible amount of computation, the adversary obtains some information about the plaintext m corresponding to c (other than the length of m).

Desirable Properties of a SKES

1. Efficient algorithms should be known for computing E_k and D_k (i.e. for encryption and decryption).
2. The key should be small (but large enough to render exhaustive key search infeasible).
3. The scheme should be secure.
4. The scheme should be secure even against the designer of the system.

Security of the Simple Substitution Cipher

Totally insecure against a chosen-plaintext attack. [Why?]

What about security under a ciphertext-only attack?

Is *exhaustive key search* possible?

- ► Given sufficient amounts of ciphertext c, decrypt c using each possible key until c decrypts to a plaintext message which "makes sense".
- ► In principle, 30 characters of ciphertext are sufficient on average to yield a unique plaintext that is a sensible English message. In practice, a few hundred characters are needed.

Security of the Simple Substitution Cipher (2)

Exhaustive search:

- Number of keys to try is $26! \approx 4 \times 10^{26} \approx 2^{88}$.
- If the adversary uses 10^6 computers, each capable of trying 10^9 keys per second, then exhaustive key search takes about 10^4 years.
- So, exhaustive key search is infeasible.

Work Factor

In this course:

- 2^{40} operations is considered *very easy*.
- 2^{56} operations is considered *easy*.
- 2^{64} operations is considered *feasible*.
- 2^{80} operations is considered *barely feasible*.
- 2^{128} operations is considered *infeasible*.

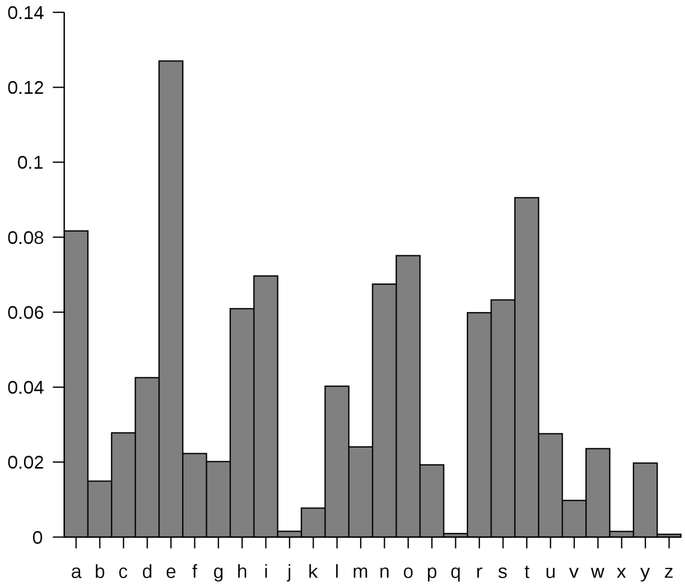
The bitcoin network is presently computing hashes at the rate of 2^{61} per second (or 2^{86} per year).

The “Landauer limit” from thermodynamics suggests that exhaustively trying 2^{128} symmetric keys would require $\gg 30$ gigawatts of power for one year (which is > 1% of the world’s energy production).

`en.wikipedia.org/wiki/Brute-force_attack`

Security of the Simple Substitution Cipher (3)

Of course, simple frequency analysis of ciphertext letters can be used to recover the key. Hence, the simple substitution cipher is totally insecure even against a ciphertext-only attack.



Frequencies of letters in normal english text.

Polyalphabetic Ciphers

Basic idea: Use several permutations, so a plaintext letter is encrypted to one of several possible ciphertext letters.

Example: *Vigenère cipher*:

- *Key* is an English word having no repeated letters e.g. $k =$ CRYPTO.
- Example of *encryption*:

$$\begin{array}{rcccccccccccccc} m = & \text{t} & \text{h} & \text{i} & \text{s} & \text{i} & \text{s} & \text{a} & \text{m} & \text{e} & \text{s} & \text{s} & \text{a} & \text{g} & \text{e} \\ +k = & \text{C} & \text{R} & \text{Y} & \text{P} & \text{T} & \text{O} & \text{C} & \text{R} & \text{Y} & \text{P} & \text{T} & \text{O} & \text{C} & \text{R} \\ \hline c = & \text{V} & \text{Y} & \text{G} & \text{H} & \text{B} & \text{G} & \text{C} & \text{D} & \text{C} & \text{H} & \text{L} & \text{O} & \text{I} & \text{V} \end{array}$$

- Here, A=0, B=1, $\ldots$, Z=25; addition of letters is mod 26.
- *Decryption* is subtraction modulo 26.
- Frequency distribution of ciphertext letters is flatter (than for a simple substitution cipher).

Security of the Vigenère Cipher

- ▶ The Vigenère cipher is totally insecure against a chosen-plaintext attack. [Why?]
- ▶ What about security under a *ciphertext-only attack*?
- ▶ The key length ℓ can be found as follows:
  - • Make a guess for ℓ and check your guess as follows.
    - ♢ Divide the ciphertext letters into ℓ groups, $G_0, G_1, \ldots, G_{\ell-1}$, where the ith ciphertext letter is placed in group $G_{i \bmod \ell}$.
    - ♢ Examine the frequency distributions of letters in each group. If each distribution "looks like" the expected distribution of letters from normal English text, then the key length guess is probably correct.

Security of the Vigenère Cipher (2)

The following are the letters of the English alphabet, grouped by letters whose frequencies are approximately equal. The letters in each group are listed in order of decreasing frequency.

Group 1: E
Group 2: T A O I N S H R
Group 3: D L
Group 4: C U M W F G Y P B
Group 5: V K J X Q Z

The crucial observation is that the letters V, K, J, X, Q and Z occur quite infrequently in sensible English text.

Security of the Vigenère Cipher (3)

Once the key length ℓ is determined, then:

- Notice that the ciphertext letters in each group G_i were obtained by applying a permutation that is a cyclic shift of the alphabet to the corresponding plaintext letters.
- Use the frequency counts of ciphertext letters in G_i to make guesses for the ith letter of the key word.
- Use the guesses for the key letters to guess the key word (keeping in mind that it is an English word).
- Check your guess for the key word by decrypting the ciphertext.

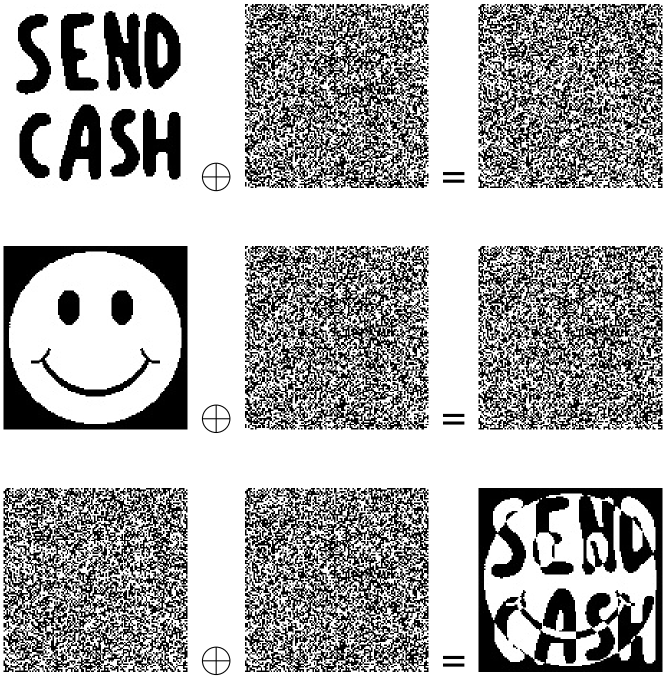
The One-Time Pad

- Invented by Vernam in 1917 for the telegraph system.
- Key is a *random* string of letters.
- Example of encryption:

$m =$	t	h	i	s	i	s	a	m	e	s	s	a	g	e
$+\,k =$	Z	F	K	W	O	G	P	S	M	F	J	D	L	G
$c =$	S	M	S	P	W	Y	P	F	Q	X	C	D	R	K

- Note: The key is as long as the plaintext.
- The key should not be re-used:
  - If $c_1 = m_1{+}k$ and $c_2 = m_2{+}k$, then $c_1{-}c_2 = m_1{-}m_2$.
  - $c_1{-}c_2$ depends only on the plaintext (and not on the key) and hence can leak information about the plaintext.
  - If m_1 is known, then m_2 can be easily computed.

Example: Reusing a Key in the One-Time Pad



$(m_1 \oplus k = c_1)$

$(m_2 \oplus k = c_2)$

$(c_1 \oplus c_2 = m_1 \oplus m_2)$

Notation: $\oplus$ is bitwise exclusive-or (i.e., bitwise addition modulo 2).

[Source: `cryptosmith.com/2008/05/31/stream-reuse/`]

Security of the One-Time Pad

- *Perfect secrecy*: The one-time pad is semantically secure against ciphertext-only attack by an adversary with infinite computational resources.
- This can be proven formally using concepts from information theory [Shannon 1949].
- The bad news: Shannon (1949) proved that if plaintexts are m-bit strings, then any symmetric-key encryption scheme with perfect secrecy must have $|K| \geq 2^m$.
- So, perfect secrecy (and the one-time pad) is fairly useless in practice.
- All is not lost: *stream ciphers*.

Stream Ciphers

- Basic idea: Instead of using a random key in the one-time pad, use a “pseudorandom” key.
- A *pseudorandom bit generator* (PRBG) is a deterministic algorithm that takes as input a (random) *seed*, and outputs a longer “pseudorandom” sequence called the *keystream*.
- Convention: From now on, unless otherwise stated, messages and keys will be assumed to be bit strings.

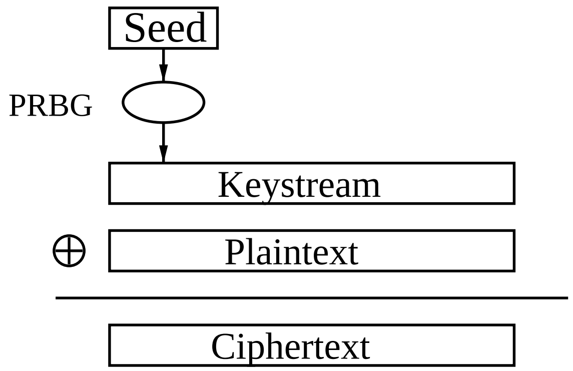
Notation: $\oplus$ is bitwise exclusive-or (i.e., bitwise addition modulo 2).
For example:

1011001010 $\oplus$ 1001001001 = 0010000011

Note that $x \oplus x = 0$ and $x \oplus y = y \oplus x$.
Hence if $x = y \oplus z$, then $x \oplus y = z$.

Stream Ciphers (2)

- Using a PRBG for encryption (a stream cipher):
  - The *seed* is the *secret key* shared by Alice and Bob.



- No more perfect secrecy – security depends on the quality of the PRBG.

Security Requirements for the PRBG

- The keystream should be "indistinguishable" from a random sequence (the *indistinguishability requirement*).

- If an adversary knows a portion c_1 of ciphertext and the corresponding plaintext m_1, then she can easily find the corresponding portion $k_1 = c_1 \oplus m_1$ of the keystream. Thus, given portions of the keystream, it should be infeasible to learn any information about the rest of the keystream (the *unpredictability requirement*).
- Aside: Don't use UNIX random number generators (rand and srand) for cryptography!
  ($X_0 =$ seed, $X_{i+1} = aX_i + b \bmod n$, $i \geq 0$.)
- We will study PRBGs in more detail later in the course.

The RC4 Stream Cipher

Designed by Ron Rivest in 1987.

- Widely used in commercial products: SSL/TLS, Lotus Notes, Windows password encryption, Adobe Acrobat, Oracle secure SQL, etc.
- Pros: Extremely simple; extremely fast; variable key sizes. No catastrophic weakness has been found.
- Cons: Design criteria are proprietary (voodoo magic); not much public scrutiny until the year 2001.
- RC4 has two components: A *key scheduling algorithm*, and a *keystream generator*.

RC4 Key Scheduling Algorithm

In the following, $K[i]$, $\overline{K}[i]$ and $S[i]$ are 8-bit integers.

Input: Secret key $K[0], K[1], \ldots, K[d-1]$. (Keysize is $8d$ bits.)
Output: 256-long array: $S[0], S[1], \ldots, S[255]$.

```
For $i$ from 0 to 255 do:
        $S[i] \leftarrow i$
        $\overline{K}[i] \leftarrow K[i \bmod d]$
$j \leftarrow 0$
For $i$ from 0 to 255 do:
        $j \leftarrow (\overline{K}[i] + S[i] + j) \bmod 256$
        Swap($S[i], S[j]$)
```

[*Idea*: S is a “random-looking” permutation of $\{0, 1, 2, \ldots, 255\}$ that is generated from the secret key]

RC4 Keystream Generator

Input: 256-long byte array: $S[0], S[1], \ldots, S[255]$.
Output: Keystream.

```
$i \leftarrow 0$; $j \leftarrow 0$
While keystream bytes are required do:
        $i \leftarrow (i + 1) \bmod 256$
        $j \leftarrow (S[i] + j) \bmod 256$
        Swap($S[i], S[j]$)
        $t \leftarrow (S[i] + S[j]) \bmod 256$
        Output($S[t]$)
```

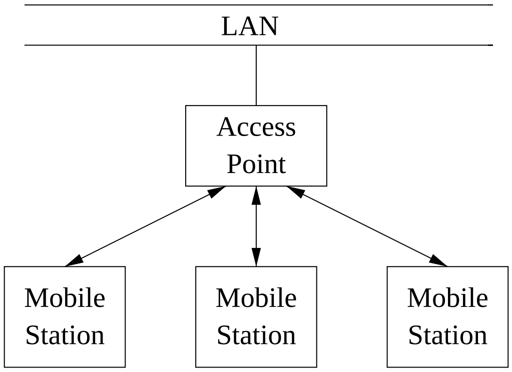
[The keystream bytes are xored with the plaintext bytes]

Wireless Security

- Wireless networks have become prevalent.
- Popular standards for wireless networks:
  - IEEE 802.11 (longer range, higher speeds, commonly used for wireless LANs).
  - Bluetooth (short range, low speed).
- New security concerns:
  - More attack opportunities (no need for physical access).
  - Attack from a distance ($>$ 1 km with good antennae).
  - No physical evidence of attack.

Case Study: IEEE 802.11 Security

- IEEE 802.11 standard for wireless LAN communications includes a protocol called *Wired Equivalent Privacy* (WEP).
- Ratified in September 1999.
- Multiple amendments: 802.11a (1999), 802.11b (1999), 802.11g (2003), 802.11i (2004), 802.11n (2009).
- IEEE 802.11b and WEP are still used today.
- WEP's goal is (only) to protect link-level data during wireless transmission between mobile stations and access points.



Main Security Goals of WEP

1. *Confidentiality*: Prevent casual eavesdropping.
   - RC4 is used for encryption.
2. *Data Integrity*: Prevent tampering with transmitted messages.
   - An ‘integrity checksum’ is used.
3. *Access Control*: Protect access to a wireless network infrastructure.
   - Discard all packets that are not properly encrypted using WEP.

Description of WEP Protocol

- Mobile station shares a secret key k with access point.
  - k is either 40 bits or 104 bits in length.
  - The standard does not specify how the key is to be distributed.
  - In practice, one shared key per LAN is common; this key is manually injected into each access point and mobile station; the key is not changed very frequently.
- Messages are divided into *packets* of some fixed length (e.g. 1500 bytes).
- WEP uses a *per-packet 24-bit IV* v to process each packet. WEP does not specify how the IVs are managed. In practice:
  - A random IV is generated for each packet; or
  - The IV is set to 0 and incremented by 1 for each use.

Description of WEP Protocol (2)

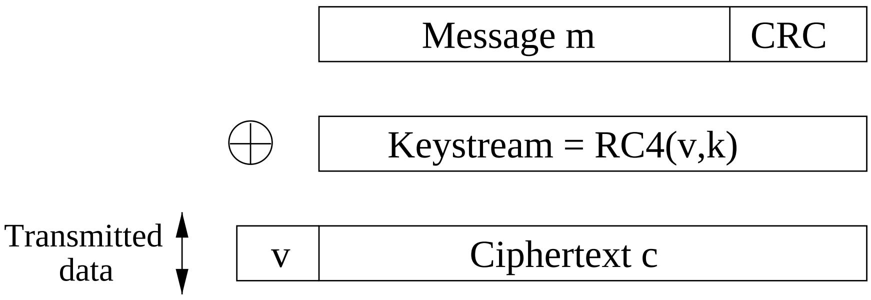
To send a packet m, an entity does the following:

1. Select a 24-bit IV v.
2. Compute a 32-bit *checksum*: $S =$ CRC(m).
   - 802.11 specifies that a CRC-32 checksum be used. CRC-32 is *linear*. That is, for any two messages m_1 and m_2 of the same bitlength,

     $$\text{CRC}(m_1 \oplus m_2) = \text{CRC}(m_1) \oplus \text{CRC}(m_2).$$

     (The details of CRC-32 are not important to us)
3. Compute $c = (m \| S) \oplus$ RC4$(v \, \| \, k)$.
   - $\|$ denotes concatenation.
   - $(v \, \| \, k)$ is the key used in the RC4 stream cipher.
4. Send (v, c) over the wireless channel.

Description of WEP Protocol (3)



The receiver of (v, c) does the following:

1. Compute $(m \, \| \, S) = c \oplus$ RC4$(v \, \| \, k)$.
2. Compute $S' =$ CRC(m); reject the packet if $S' \neq S$.

Question: Are confidentiality, data integrity, and access control achieved?
Answer: NO! [Borisov, Goldberg & Wagner; 2001]

Problem: IV Collision

- Suppose that two packets (v, c) and (v, c') use the same IV. Let m, m' be the corresponding plaintexts. Then $c \oplus c' = m \oplus m'$.
- If m is known, then m' is immediately available.
- If m is not known, then one may be able to use the expected distribution of m and m' to discover information about them. (Much of network traffic contents is predictable.)

Finding IV Collisions

- Since there are only 2^{24} choices for the IV, collisions are guaranteed after enough time — a few days on a busy network (5 Mbps).
- If IVs are randomly selected, then one can expect a collision after about 2^{12} packets.
  - Birthday paradox: Suppose that an urn contains n numbered balls. Suppose that balls are drawn from the urn, one at a time, with replacement. The expected number of draws before a ball is selected for a second time (called a *collision*) is approximately $\sqrt{\pi n/2}$.
- Collisions are more likely if keys k are long-lived and the same key is used for multiple mobile stations in a network.
- Thus, WEP does not provide a high degree of confidentiality.

Problem: Checksum is Linear

- CRC-32 is used to check integrity. This is fine for random errors, but not for deliberate ones.
- It is easy to make controlled changes to (encrypted) packets:
  - Suppose (v, c) is an encrypted packet.
  - Let $c = \text{RC4}(v\|k) \oplus (m\|S)$, where k, m, S are unknown.
  - Let $m' = m \oplus \Delta$, where Δ is a bit string.
    (The 1's in Δ correspond to the bits of m an attacker wishes to change.)
  - Let $c' = c \oplus (\Delta\|\text{CRC}(\Delta))$.
  - Then (v, c') is a valid encrypted packet for m'.
    [Exercise: Prove this]
- Thus, WEP does not provide data integrity.

Problem: Integrity Function is Unkeyed

- Suppose that an attacker learns the plaintext m corresponding to a single encrypted packet (v, c).
- Then, the attacker can compute the RC4 keystream $\text{RC4}(v\|k) = c \oplus (m\|\text{CRC}(m))$.
- Henceforth, the attacker can compute a valid encrypted packet for *any* plaintext m' of her choice: (v, c'), where $c' = \text{RC4}(v\|k) \oplus (m'\|\text{CRC}(m'))$.
- Thus, WEP does not provide access control.

Recommended Reading: Sections 1, 2, 3, 4.1, 4.2, 6 of “Intercepting mobile communications: The insecurity of 802.11", by N. Borisov, I. Goldberg and D. Wagner.

A More Devastating Attack

- Fluhrer, Mantin & Shamir, 2001.
- Assumptions:
  1. The same 104-bit key k is used for a long period of time. [Most products do this.]
  2. The IV is incremented for each packet, or a random IV is selected for each packet. [Most products do this.]
  3. The first plaintext byte of each packet (i.e. the first byte of each m) is known to the attacker.
     [Most wireless protocols prepend the plaintext with some header bytes which are non-secret.]
- *Attack*: A *passive* adversary who can collect about 5,000,000 encrypted packets can very easily recover k (and thus totally break the system).
  [Details not covered in this course.]

Implementing the Fluhrer-Mantin-Shamir Attack

- The attack can be easily mounted in practice:
  - Can buy a $100 wireless card and hack drivers to capture (encrypted) packets.
  - On a busy wireless network (5Mbps), 5 million packets can be captured in a few hours, and then k can be immediately computed.
- Implementation details: A. Stubbefield, J. Ionnidis, A. Rubin, “Using the Fluhrer, Mantin and Shamir attack to break WEP”, AT&T Technical Report, August 2001.
- Script kiddies:
  - *Aircrack-ng*: `www.aircrack-ng.org/doku.php`
  - *WEPCrack*: `sourceforge.net/projects/wepcrack`
- *aircrack-ptw*: Breaks WEP in under 60 seconds (only about 40,000 packets are needed).

Implications of WEP Insecurity

- WEP was blamed for the 2007 theft of 45 million credit-card numbers from T.J. Maxx.
  (T.J. Maxx is an American department store chain)
- A subsequent class action lawsuit settled for $40,900,000.
- See: tinyurl.com/WEP-TJMaxx

IEEE 802.11 Update

`http://en.wikipedia.org/wiki/Wi-Fi_Protected_Access`

- Wi-fi Protected Access (*WPA*) [temporary replacement]
  - Uses a master key, from which 128-bit session keys are periodically generated.
  - The 128-bit session key is combined with a 48-bit IV to form the RC4 key.
  - A Message Authentication Code (MAC) algorithm is used instead of CRC.
  - WPA is compatible with many (but not all) WEP access points.
- *WPA2*
  - Implements the new *IEEE 802.11i* standard (2004).
  - Uses the AES block cipher instead of RC4.

RC4 Update

- The Fluhrer-Mantin-Shamir attack exploits known biases in the first few bytes of the keystream. The attack can be defeated by discarding the first few bytes (e.g., 768 bytes) of the keystream. [Details omitted]
- The Fluhrer-Mantin-Shamir attack is not effective on RC4-based SSL/TLS. This are two reasons for this:
  - (i) SSL generates RC4 encryption keys by hashing the master secret key and some nonces, so that different sessions have unrelated keys.
  - (ii) SSL does not re-key RC4 for each packet in a session.
- As of 2013, approximately 50% of all SSL traffic was secured using RC4.
- *2013-2015*: Because of several new weaknesses discovered, RC4 is no longer used in applications such as SSL.

Lessons Learned

1. *The devil is in the details*: RC4 is a pretty good (secure) stream cipher. However, it was improperly used in WEP and the result was a highly insecure communications protocol. Do not assume that “obvious” ways of using cryptographic functions are secure.

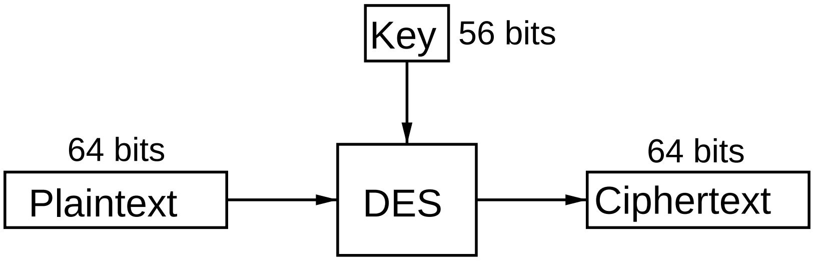
2. *Attacks only get better; they never get worse*.
   - *Moore’s law* (1965): computers get twice as fast every two years.
   - Known attacks are constantly being tweaked and improved.
   - New attacks are constantly being invented.

Lessons Learned (2)

3. *Designing security is hard*:
   - Designing cryptographic protocols is complicated and difficult.
   - Clearly state your security objectives.
   - Hire *cryptography* experts to design your security. Programming or engineering experts are not good enough.
   - Make your protocols available for public scrutiny.

Block Ciphers

- A *block cipher* is a SKES that breaks up the plaintext into blocks of a fixed length (e.g. 128 bits), and encrypts the blocks one at a time.
- In contrast, a *stream cipher* encrypts the plaintext one character (usually a bit) at a time.
- Canonical example of a block cipher: The *Data Encryption Standard (DES)*



Key size: 56 bits; Size of key space: 2^{56}; Block size: 64 bits.

Brief History of Block Ciphers

- Late 1960's: Feistel network and LUCIFER designed at IBM.
- 1972: NBS (now *NIST*: National Institute of Standards and Technology) solicits proposals for encryption algorithms for the protection of computer data.
- 1974: *IBM* develops DES.
- 1975: *NSA* (National Security Agency) "fixes" DES
  - Reduces the key size from 64 bits to 56 bits.
- 1977: DES adopted as US Federal Information Processing Standard (FIPS 46).
- 1981: DES adopted as a US banking standard (ANSI X3.92).

Brief History of Block Ciphers (2)

- 1997: NIST begins the *AES* (Advanced Encryption Standard) competition.
- 1999: 5 finalists for AES announced.
- 2001: *Rijndael* adopted for AES (FIPS 197).
  - AES has three key sizes: *128*, *192* and *256* bits.
- 2012:
  - No significant weaknesses found with AES (as yet).
  - AES uptake is rapidly increasing.
  - However, DES (and Triple-DES) is still deployed.

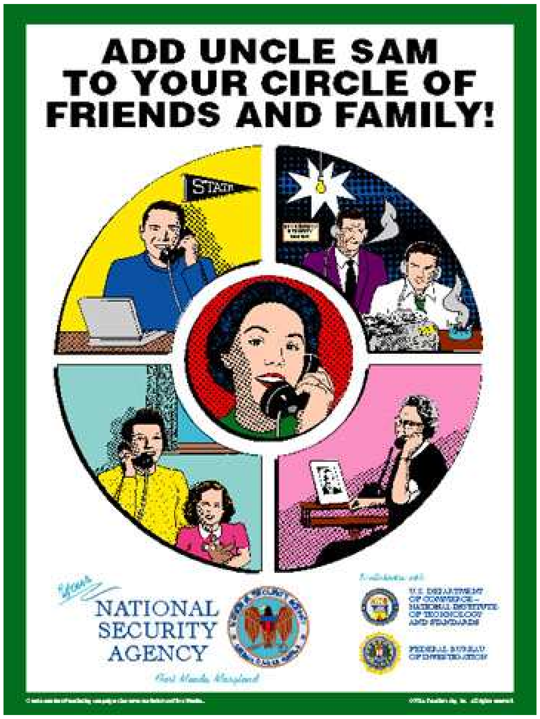
The National Security Agency (NSA)

The National Security Agency

- `www.nsa.gov`
- Founded in 1952.
- Budget: *Classified* (estimated: $10.8 billion/year).
- Number of employees: *Classified* (30,000–40,000).
- Signals Intelligence (*SIGINT*): produce foreign intelligence information.
- Information Assurance (*IA*): protects all classified and sensitive information that is stored or sent through US government equipment.
- Very influential in setting US government export policy for cryptographic products (especially encryption).
- Canadian counterpart: Communications Security Establishment (*CSE*) `www.cse-cst.gc.ca`

The National Security Agency

A GOOD MARKETING ORGANIZATION LISTENS TO ITS CUSTOMERS...
WE HEAR YOU.
Your NATIONAL SECURITY AGENCY


ADD UNCLE SAM TO YOUR CIRCLE OF FRIENDS AND FAMILY!
STATE
Your NATIONAL SECURITY AGENCY

NSA Shenanigans

Edward Snowden

- CIA/NSA contractor who disclosed in 2013 up to 200,000 NSA classified documents to the press.
- Currently in Russia under temporary asylum.
- Documents revealed the enormous capabilities of NSA and its partners (including CSE and GCHQ):
  - Groundbreaking cryptanalytic capabilities
  - Collects large amounts of internet communications and automatically analyzes the data
  - Directly attacks routers, switches, firewalls, etc.
  - Installs viruses on individual computers
  - Breaks into individual computers
- Bruce Schneier: “The NSA is subverting the Internet and turning it into a massive surveillance tool.”

Some Desirable Properties of Block Ciphers

Design principles described by Claude Shannon in 1949:

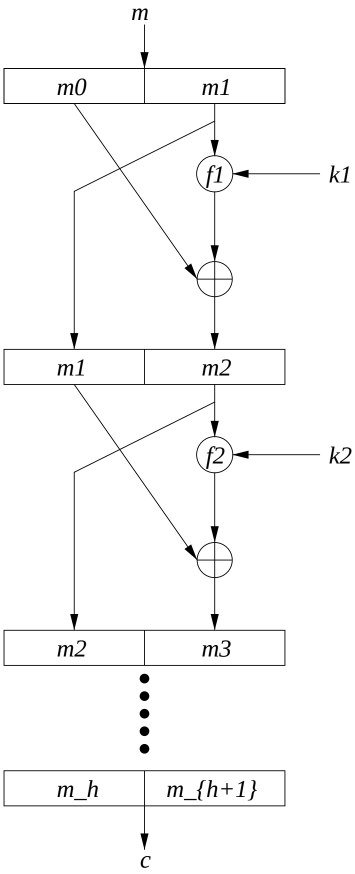
- Security:
  - *Diffusion*: each ciphertext bit should depend on all plaintext bits.
  - *Confusion*: the relationship between key and ciphertext bits should be complicated.
  - *Key size*: should be small, but large enough to preclude exhaustive key search.
- Efficiency:
  - High encryption and decryption rate.
  - Simplicity (easier to implement and analyze).
  - Suitability for hardware or software.

Feistel Ciphers: A Class of Block Ciphers

Components of a Feistel cipher:

- Parameters: n (half the block size), h (number of rounds), ℓ (key size).
- $M = \{0,1\}^{2n}$, $C = \{0,1\}^{2n}$, $K = \{0,1\}^{\ell}$.
- A *key scheduling algorithm* which determines *subkeys* $k_1, k_2, \ldots, k_h$ from a key k.
- Each subkey k_i defines a *component function* $f_i : \{0,1\}^n \to \{0,1\}^n$.

The Feistel Ladder



Components of a Feistel Cipher

Encryption takes h rounds:

- Plaintext is $m = (m_0, m_1)$, where $m_i \in \{0,1\}^n$.
- Round 1: $(m_0, m_1) \rightarrow (m_1, m_2)$, where $m_2 = m_0 \oplus f_1(m_1)$.
- Round 2: $(m_1, m_2) \rightarrow (m_2, m_3)$, where $m_3 = m_1 \oplus f_2(m_2)$.
- $\ldots\ldots$
- Round h: $(m_{h-1}, m_h) \rightarrow (m_h, m_{h+1})$, where $m_{h+1} = m_{h-1} \oplus f_h(m_h)$.
- Ciphertext is $c = (m_h, m_{h+1})$.

Decryption: Given $c = (m_h, m_{h+1})$ and k, find $m = (m_0, m_1)$:

- Compute $m_{h-1} = m_{h+1} \oplus f_h(m_h)$.
- Similarly, compute $m_{h-2}, \ldots, m_1, m_0$.

Feistel Cipher (notes)

- No restrictions need be placed on the functions f_i in order for the encryption procedure to be invertible.
- Underlying principle: Take something "simple" and use it several times; hope that the result is "complicated".
- Implementation notes:
  - Encryption: Only need to implement one round; the same code can be used for each round.
  - Decryption: Can use encryption code. (Use subkeys in reverse order.)
- We will study two examples of Feistel ciphers:
  - *New Data Seal* (also designed at IBM).
  - *DES*.

The New Data Seal (NDS)

- Feistel cipher with $n = 64$, $h = 16$.
- Key is a random function $S_k : \{0,1\}^8 \to \{0,1\}^8$ ($\ell = 2048$; exhaustive key search is infeasible).

  Note: S_k can be represented by a table.

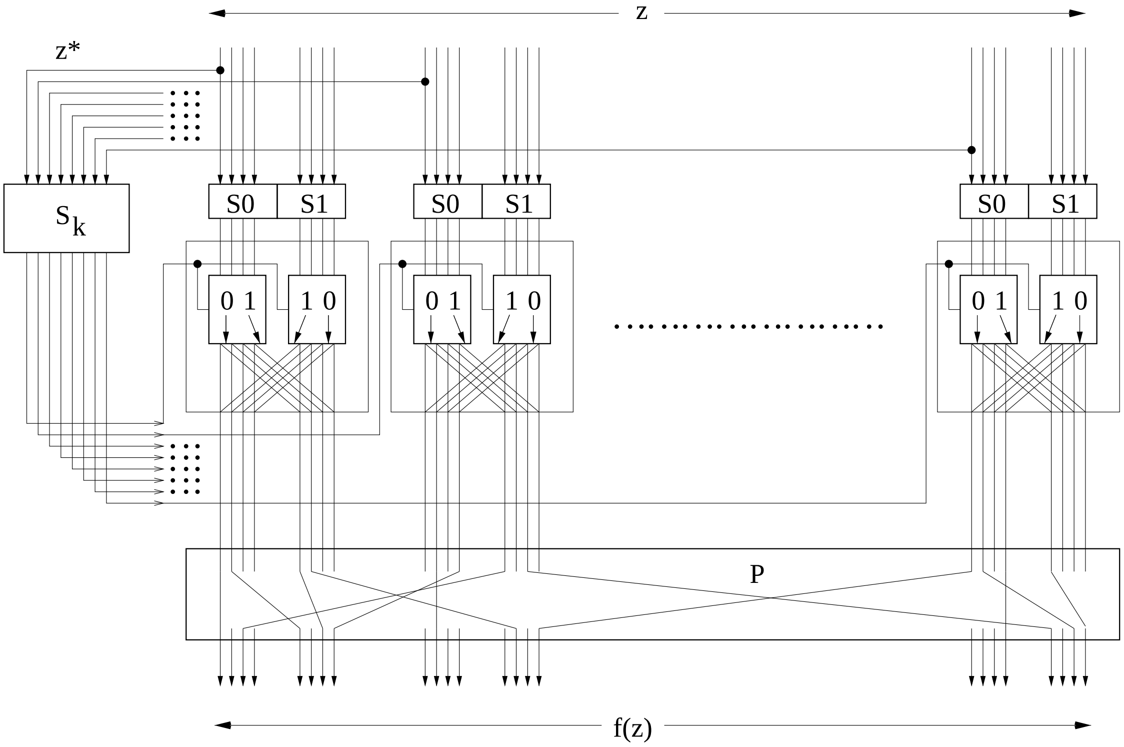
  Example:

r	0	1	2	3	4	5	$\cdots$	255
$S_k(r)$	73	106	5	7	73	119	$\cdots$	210

- Each subkey is S_k itself (so we will write f for f_i).
- The component function $f : \{0,1\}^{64} \to \{0,1\}^{64}$ is defined in the following two slides.

The NDS Component Function f

The NDS component function $f : \{0,1\}^{64} \rightarrow \{0,1\}^{64}$.



The NDS Component Function f

Input is $z \in \{0,1\}^{64}$. Output is $f(z) \in \{0,1\}^{64}$.

1. Divide z into 8 bytes: $z = (z^1, z^2, \ldots, z^8)$.
2. Divide each byte z^j into two nibbles: (n_1^j, n_2^j).
3. Apply $S_0 : \{0,1\}^4 \rightarrow \{0,1\}^4$ to n_1^j to get p_1^j $(1 \leq j \leq 8)$. Apply $S_1 : \{0,1\}^4 \rightarrow \{0,1\}^4$ to n_2^j to get p_2^j $(1 \leq j \leq 8)$. (S_0, S_1 are fixed and public knowledge)
4. Let z^* be the byte obtained by taking the first bit of each byte of z. Compute the byte $t = S_k(z^*)$.
5. If the jth bit of t is 1, then swap p_1^j and p_2^j $(1 \leq j \leq 8)$.
6. Permute (rearrange) the resulting 64 bits as determined by P. (P is fixed and public knowledge)

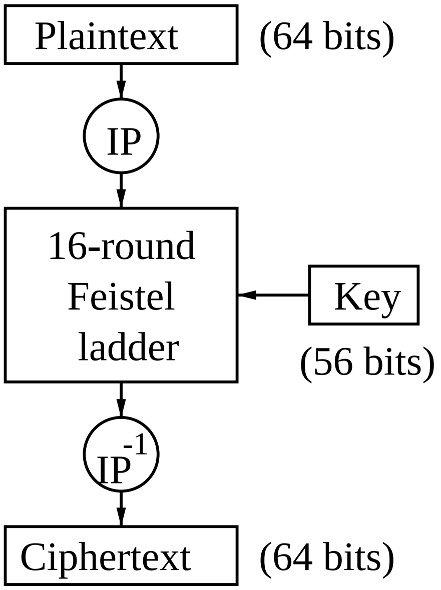
Chosen-Plaintext Attack on NDS

- An adversary can easily determine the secret key after obtaining the ciphertexts for about 32,000 carefully-chosen plaintexts.
- The attack shows that NDS is (totally) *insecure*.
- The attack also demonstrates the importance of using different subkeys in the rounds of a Feistel cipher.
- For a description of the attack see your notes from class.

The Data Encryption Standard (DES)

- Designed in 1973/74 by IBM.
- “Fixed” by NSA in 1975 (keysize reduced from 64 bits to 56 bits).
- Adopted in 1977 by NIST as a US government FIPS for encryption of unclassified data.
- Adopted in 1981 as a US banking standard.
- DES (and Triple-DES) are still widely used is practice, although their use is decreasing.
- Design principles are still classified, making analysis difficult.
- Hundreds of research papers written on security of DES.

Overview of DES



Plaintext (64 bits)

IP

16-round Feistel ladder

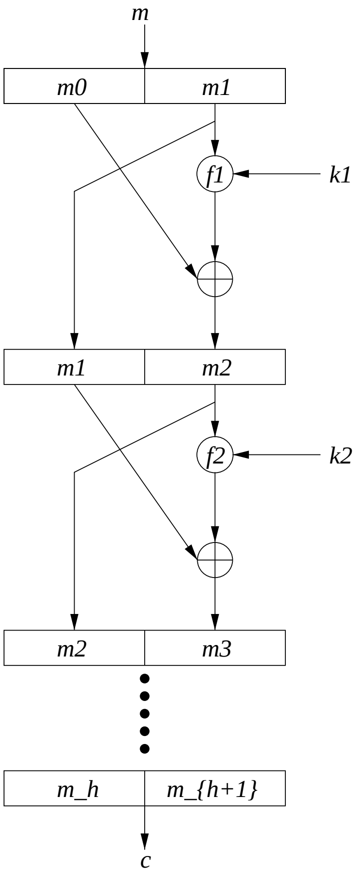
Key (56 bits)

IP^{-1}

Ciphertext (64 bits)

(IP = Initial Permutation)

The Feistel Ladder

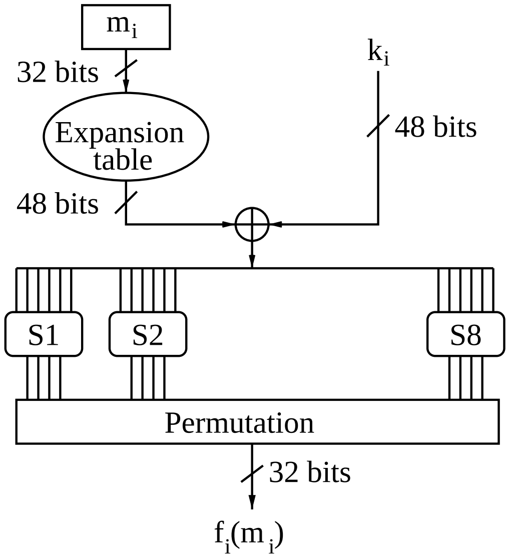


Description of DES

- Feistel cipher with $n = 32$, $h = 16$, $\ell = 56$.
- Key scheduling algorithm:
  - Selects 16 48-bit *subkeys* $k_1, \ldots, k_{16}$ from the key k.
  - Each subkey consists of a (fixed) selection of 48 bits of k.
- *S-boxes* (S1, S2, S3, S4, S5, S6, S7, S8):
  - The only components of DES that are non-linear.
  - The security of DES crucially depends on their choice.
  - DES with randomly selected S-boxes is easy to break.

Description of DES (2)

Component function $f_i : \{0,1\}^{32} \to \{0,1\}^{32}$:



Note: The IP, key scheduling algorithm, Expansion table, S-boxes, and Permutation are fixed and public knowledge.

Performance

[Wei Dai] `www.cryptopp.com/benchmarks.html`
Speed benchmarks (2009) for a software implementation on an Intel Core 2 1.83 GHz processor.

SKES	Block size (bits)	Key size (bits)	Speed
DES	64	56	32 Mbits/sec
3DES	64	112	13 Mbits/sec
AES	128	128	139 Mbits/sec
RC4	—	variable	126 Mbits/sec
SKIPJACK	80	80	10 Mbits/sec
DES	64	56	10+ Gbits/sec (ASIC chip)

DES Problem 1: Small Key Size

- Exhaustive search on key space takes 2^{56} steps and can be *easily parallelized*.
- DES challenges from RSA Security (3 known PT/CT pairs):

The unkn	own mess	age is:	????????

  - June *1997*: Broken by Internet search (3 months).
  - July *1998*: Broken in 3 days by DeepCrack machine (1800 chips; $250,000).
  - Jan *1999*: Broken in 22 hrs, 15 min (DeepCrack + `distributed.net`).
- RC5 64-bit challenge:
  - July *2002*: Broken in 1757 days.
  - Participation by 331,252 individuals.
- In progress (`distributed.net`): RC5 72-bit challenge.

DES Problem 2: Small Block Size

- If plaintext blocks are distributed “uniformly at random”, then the expected number of ciphertext blocks observed before a collision occurs is $\approx 2^{32}$ (by the birthday paradox).
    - Hence the ciphertext reveals <u>some</u> information about the plaintext.

- Small block size is also damaging to some authentication applications (more on this later).

Sophisticated Attacks on DES

Differential cryptanalysis [Biham & Shamir 1989]:

- Recovers key given 2^{47} chosen plaintext/ciphertext pairs.
- DES was designed to resist this attack.
- Differential cryptanalysis has been more effective on some other block ciphers.

Linear cryptanalysis [Matsui 1993]:

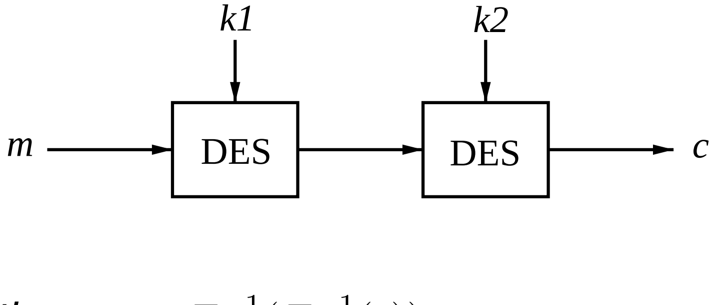
- Recovers key given 2^{43} known plaintext/ciphertext pairs.
- Storing these pairs takes 131,000 Gbytes.
- Implemented in 1993: 10 days on 12 machines.

Multiple Encryption

- Recall that the only weaknesses known in DES are the obvious ones: small key size and small block size.
- Question: How can one construct a more secure block cipher from DES? (i.e. without changing the internals of DES.)
- *Multiple encryption*: Re-encrypt the ciphertext one or more times using independent keys. Hope that this increases the effective key size.
- Multiple encryption does not always result in increased security.
  Example: If E_π denotes the simple substitution cipher with key π, then is $E_{\pi_1} \circ E_{\pi_2}$ any more secure than E_π?

Double Encryption

- *Double-DES*. Key is $k = (k_1, k_2)$, $k_1, k_2 \in_R \{0,1\}^{56}$.
  [$k \in_R K$ means that k is chosen uniformly at random from K]
- *Encryption*: $c = E_{k_2}(E_{k_1}(m))$.
  (E = DES encryption, E^{-1} = DES decryption)



- *Decryption*: $m = E_{k_1}^{-1}(E_{k_2}^{-1}(c))$.
- Key size of Double-DES is $\ell = 112$, so exhaustive key search takes 2^{112} steps (infeasible).
- Note: Block size is unchanged.

Meet-In-The-Middle Attack on Double-DES

Main idea: If $c = E_{k_2}(E_{k_1}(m))$, then $E_{k_2}^{-1}(c) = E_{k_1}(m)$.

Input: 3 known PT/CT pairs (m_1, c_1), (m_2, c_2), (m_3, c_3).
Output: The secret key (k_1, k_2).

1. For each $h_2 \in \{0,1\}^{56}$:
   (a) Compute $E_{h_2}^{-1}(c_1)$, and store $[E_{h_2}^{-1}(c_1), h_2]$ in a table sorted by first component.
2. For each $h_1 \in \{0,1\}^{56}$ do the following:
   (a) Compute $E_{h_1}(m_1)$.
   (b) Search for $E_{h_1}(m_1)$ in the table. (We say that $E_{h_1}(m_1)$ *matches* table entry $[E_{h_2}^{-1}(c_1), h_2]$ if $E_{h_1}(m_1) = E_{h_2}^{-1}(c_1)$.)
   (c) For each match $[E_{h_2}^{-1}(c_1), h_2]$ in the table, check if $E_{h_2}(E_{h_1}(m_2)) = c_2$; if so then check if $E_{h_2}(E_{h_1}(m_3)) = c_3$. If both checks pass, then output (h_1, h_2) and STOP.

Number of PT/CT Pairs Needed

Number of known plaintext/ciphertext pairs needed for unique key determination:

Question: Let E be a block cipher with key space $K = \{0,1\}^{\ell}$, and plaintext and ciphertext space $\{0,1\}^{L}$.

Let $k' \in K$ be the secret key chosen by Alice and Bob, and let (m_i, c_i), $1 \le i \le t$, be known plaintext/ciphertext pairs, where the plaintext m_i are pairwise distinct.
(Note that $c_i = E_{k'}(m_i)$ for all $1 \le i \le t$)

Then how large should t be to ensure (with probability very close to 1) that there is only one key $k \in K$ such that $E_k(m_i) = c_i$ for all $1 \le i \le t$?

Number of PT/CT Pairs Needed (2)

- For each $k \in K$, the encryption function $E_k : \{0,1\}^L \to \{0,1\}^L$ is a permutation.
- We make the *heuristic assumption* that for each $k \in K$, E_k is a random function (i.e., a randomly selected function). This assumption is certainly false since E_k is *not* random, and because a random function is almost certainly not a permutation. Nonetheless, it turns out that the assumption is good enough for our analysis.
- Now, fix $k \in K$, $k \neq k'$. Then the probability that $E_k(m_i) = c_i$ for all $1 \leq i \leq t$ is
$$\underbrace{\frac{1}{2^L} \cdot \frac{1}{2^L} \cdots \frac{1}{2^L}}_{t} = \frac{1}{2^{Lt}}.$$
- Thus the expected number of $k \in K$ (not including k') for which $E_k(m_i) = c_i$ for all $1 \leq i \leq t$ is
$$FK = \frac{2^\ell - 1}{2^{Lt}}.$$

Meet-In-The-Middle Attack on Double DES

Let E be the DES encryption function, so Double-DES encryption is $c = E_{k_2}(E_{k_1}(m))$.

1. If $\ell = 112$, $L = 64$, $t = 3$, then $FK \approx 1/2^{80}$.
   Thus if a Double-DES key (h_1, h_2) is found for which $E_{h_2}(E_{h_1}(m_i)) = c_i$ for $i = 1, 2, 3$, then with very high probability we have $(h_1, h_2) = (k_1, k_2)$.
2. If $\ell = 112$, $L = 64$, $t = 1$, then $FK \approx 2^{48}$.
   Thus the expected number of Double-DES keys (h_1, h_2) for which $E_{h_2}(E_{h_1}(m_1)) = c_1$ is 2^{48}.
3. Of the 2^{48} keys (h_1, h_2) satisfying $E_{h_2}(E_{h_1}(m_1)) = c_1$, the expected number of keys which also satisfy $E_{h_2}(E_{h_1}(m_2)) = c_2$ is $\approx 2^{48}/2^{64} = 1/2^{16}$.

Analysis of the Meet-In-The-Middle Attack

Analysis:

- Number of DES operations is $\approx 2^{56} + 2^{56} + 2 \cdot 2^{48} \approx 2^{57}$. (We are not counting the time to do the sorting and searching)
- Space requirements: $2^{56}(64 + 56)$ bits $\approx 983,040$ Tbytes.

Time-memory tradeoff. [Exercise] The attack can be modified to decrease the storage requirements at the expense of time:

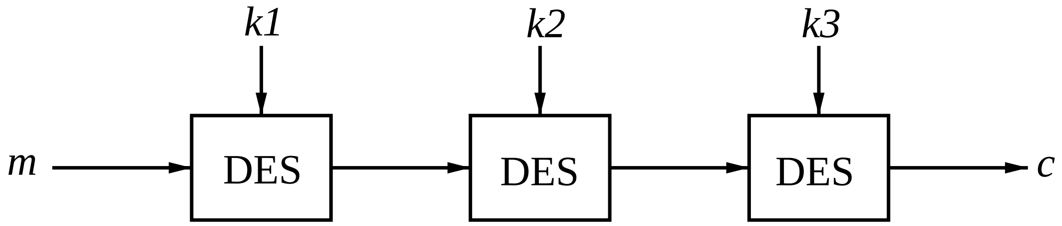
- Time: 2^{56+s} steps; memory: 2^{56-s} units, $1 \leq s \leq 55$.

Conclusions:

- Double-DES has the same effective key size as DES.
- Double-DES is not much more secure than DES.

Triple-Encryption

- *Triple-DES*. Key is $k = (k_1, k_2, k_3)$, $k_1, k_2, k_3 \in_R \{0,1\}^{56}$.
- *Encryption*: $c = E_{k_3}(E_{k_2}(E_{k_1}(m)))$. ($E$ = DES encryption, E^{-1} = DES decryption)



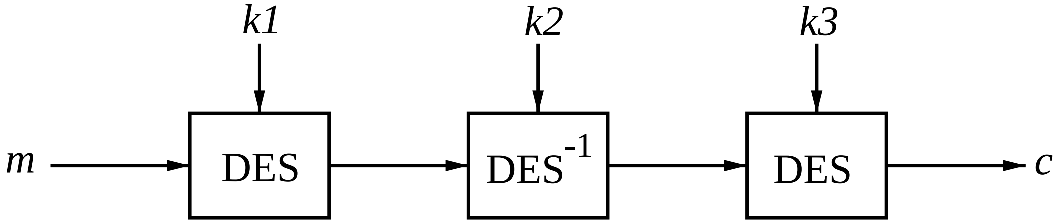
- *Decryption*: $m = E^{-1}_{k_1}(E^{-1}_{k_2}(E^{-1}_{k_3}(c)))$.
- Key size of Triple-DES is $\ell = 168$, so exhaustive key search takes 2^{168} steps (infeasible).

Triple-Encryption (2)

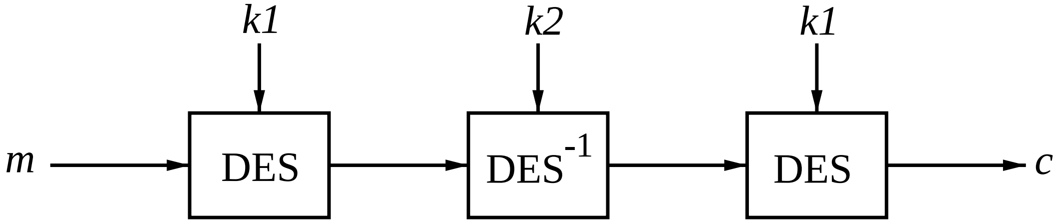
- Meet-in-the-middle attack takes $\approx 2^{112}$ steps. [Exercise]
- So, the effective key size of Triple-DES against exhaustive key search is $\leq$ 112 bits.
- No *proof* that Triple-DES is more secure than DES.
- Note: Block size is unchanged.
- Possibility of dictionary attacks.
  - Adversary stores a large table (of size $\leq 2^{64}$) of (m, c) pairs.
  - To prevent this attack: change secret keys periodically.
- Triple-DES is widely deployed.

Some Variants

EDE Triple-DES: for backward compatibility with DES



Two-key Triple-DES:



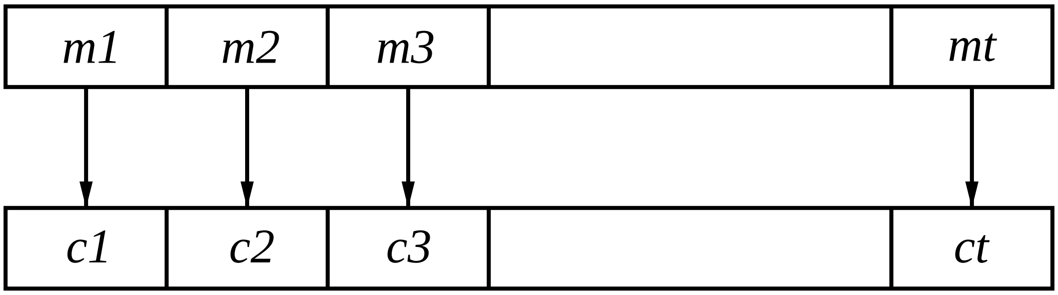
Effective key length is 56 bits (under a chosen-plaintext attack). [Hard Exercise]

Block Cipher Modes of Operation

- Plaintext message is $m = m_1, m_2, \ldots, m_t$, where each m_i is an L-bit block.
- Question: How should we use a block cipher $E_k : \{0,1\}^L \to \{0,1\}^L$ to encrypt m?

Electronic Codebook (ECB) Mode

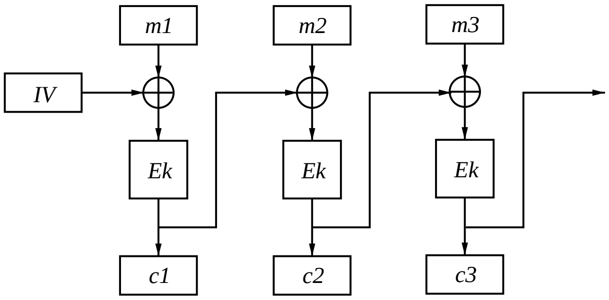
- *Encrypt* blocks independently, one at a time: $c = c_1, c_2, \ldots, c_t$, where $c_i = E_k(m_i)$.



- *Decryption*: $m_i = E_k^{-1}(c_i)$, $i = 1, 2, \ldots, t$.
- Drawback: Identical plaintexts result in identical ciphertexts (under the same key), and thus ECB encryption is not (semantically) secure against chosen-plaintext attacks. [Why?]

Cipher Block Chaining (CBC) Mode

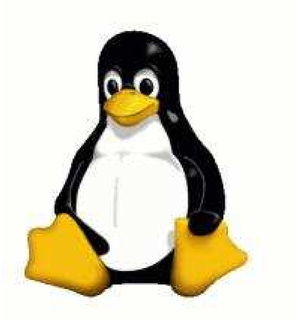
- *Encryption*: Select $c_0 \in_R \{0,1\}^L$ (c_0 is a random non-secret IV). Then compute $c_i = E_k(m_i \oplus c_{i-1})$, $i = 1, 2, \ldots, t$.



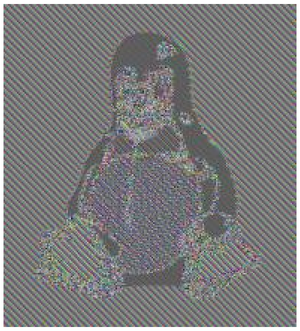
- Ciphertext is $(c_0, c_1, c_2, \ldots, c_t)$.
- *Decryption*: $m_i = E_k^{-1}(c_i) \oplus c_{i-1}$, $i = 1, 2, \ldots, t$.
- Identical plaintexts with different IVs result in different ciphertexts and for this reason CBC encryption is (semantically) secure against chosen-plaintext attacks (for a well chosen block cipher E).

ECB versus CBC encryption

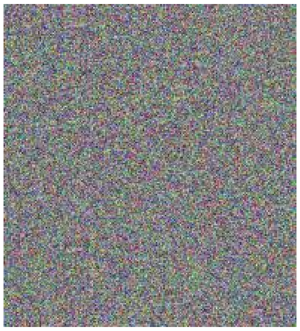
Plaintext



ECB encryption

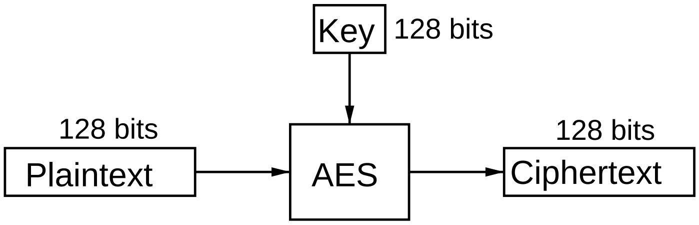


CBC encryption



The Advanced Encryption Standard (AES)

- `www.nist.gov/aes`
- September 1997: Call issued for AES candidate algorithms.
- Requirements:
  - Key sizes: *128*, *192* and *256* bits.
  - Block size: 128 bits.
  - Efficient on both hardware and software platforms.
  - Availability on a worldwide, non-exclusive, royalty-free basis.

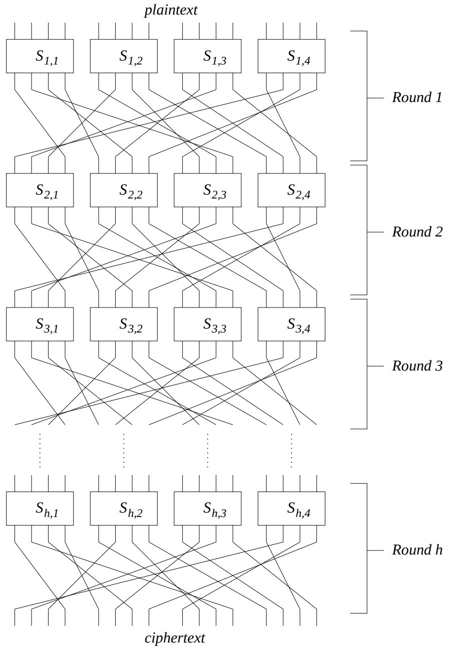


The AES Process

- August 1998: 15 submissions in Round 1.
- August 1999: 5 finalists selected by NIST:
  - MARS, RC6, Rijndael, Serpent, Twofish.
- 1999: NSA performed a hardware efficiency comparison.
- October 2 2000: *Rijndael* was selected.
- December 2001: The AES standard is officially adopted (FIPS 197).
- Rijndael is an iterated block cipher. It is not a Feistel cipher, but is an example of a substitution-permutation network.

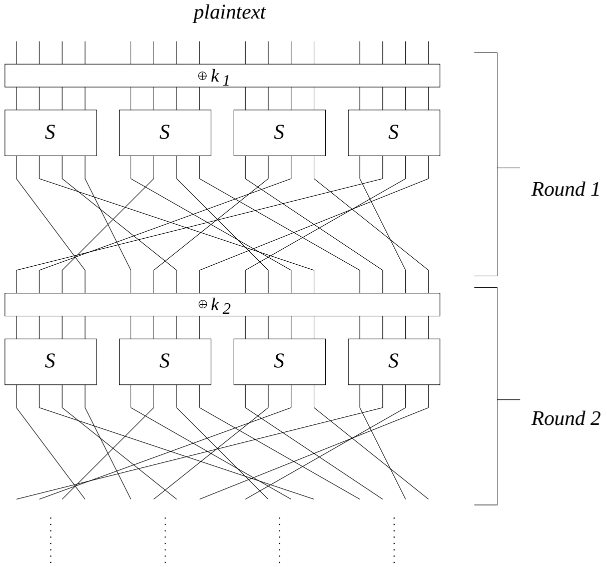
Substitution-Permutation Networks (1)

A *substitution-permutation network* (SPN) is an iterated block cipher where a round consists of a *substitution* operation followed by a *permutation* operation.



Substitution-Permutation Networks (2)

- The key k is used to influence the result of the substitution step.
- One way to do this is to XOR the S-box inputs with key bits before the S-box is applied.
- From k, derive round keys, $k_1, k_2, \ldots, k_h, k_{h+1}$. ($h$ denotes the number of rounds)



Substitution-Permutation Networks (3)

- By XOR-ing an additional round key, k_{h+1}, after the last round, this prevents an adversary from taking ciphertext and undoing the final substitution and permutation operations.
- The internals of the cipher are protected by k_1 and k_{h+1}. This is called *whitening*.
- Here is a description of *encryption*:

```
$A \leftarrow$ plaintext
for $i = 1 \ldots h$ do
    $A \leftarrow A \oplus k_i$    (XOR)
    $A \leftarrow S(A)$    (Substitution)
    $A \leftarrow P(A)$    (Permutation)
$A \leftarrow A \oplus k_{h+1}$
ciphertext $\leftarrow A$
```

- *Decryption* is just the reverse of encryption.
(The S-box must be invertible!)

AES

- AES is an SPN, where the permutation operation is replaced by two invertible linear transformations.
- All operations are *byte* oriented (e.g., S-box maps 8-bits to 8-bits). This allows AES to be efficiently implemented on various platforms.
- The block size of AES is 128 bits.
- Each round key is 128 bits.
- AES accepts three different key lengths. The number of rounds depends on the key length:

key length	h
128	10
192	12
256	14

AES Round Operations

- Each round updates a variable called State which consists of a 4×4 array of bytes (note: $4 \cdot 4 \cdot 8 = 128$, the block size). State is initialized with the plaintext:

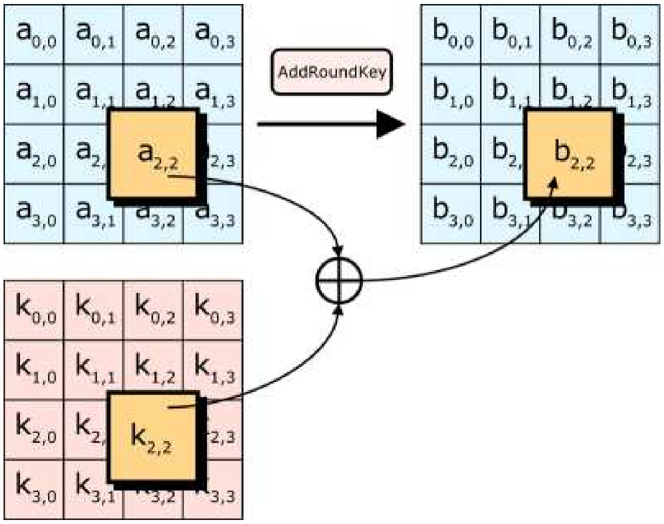
$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$\longleftarrow$ *plaintext*

- After h rounds are completed, a final round key is XOR-ed with State, the result being the ciphertext.
- The AES round function uses four operations: `AddRoundKey`, `SubBytes`, `ShiftRows`, `MixColumns`.
- Some arithmetic operations used in AES are carried out in the finite field GF(2^8) (we will not cover finite field arithmetic in this course).

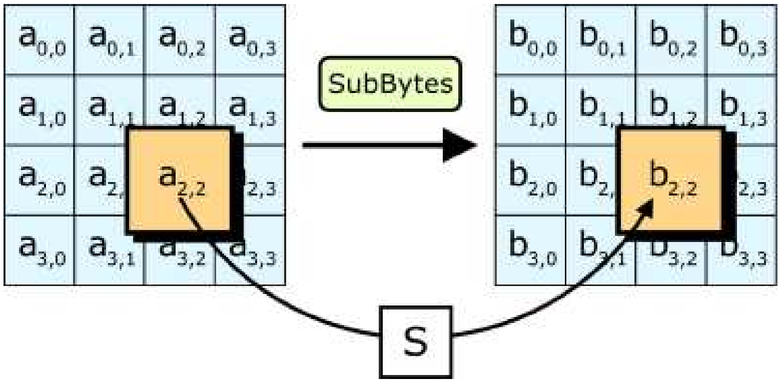
Add Round Key

XOR each byte of State with the corresponding byte of the round key.


AddRoundKey
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,2}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,2}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$
$k_{0,0}$ $k_{0,1}$ $k_{0,2}$ $k_{0,3}$
$k_{1,0}$ $k_{1,1}$ $k_{1,2}$ $k_{1,3}$
$k_{2,0}$ $k_{2,2}$
$k_{3,0}$ $k_{3,1}$

Substitute Bytes

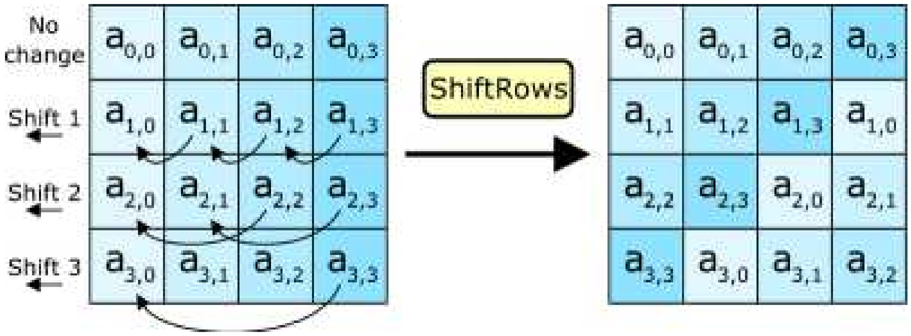
Take each byte in State and replace it with the output of the S-box.


$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
SubBytes
S
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

$S : \{0,1\} \to \{0,1\}^8$ is a fixed, public, invertible function.

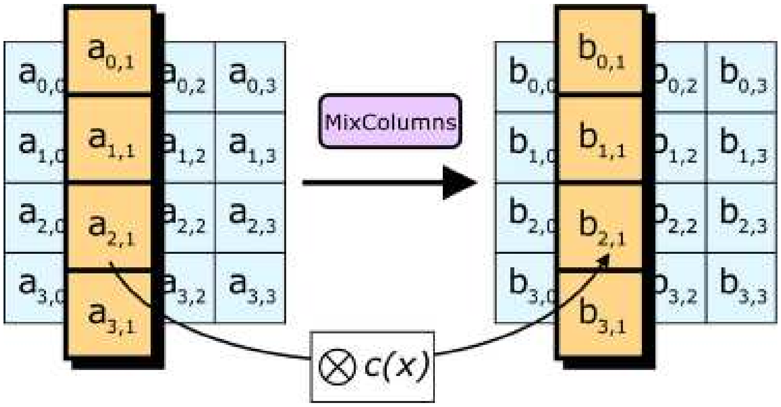
Shift Rows

Permute the bytes of State by applying a *cyclic shift* to each row.


No change
Shift 1
Shift 2
Shift 3
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
ShiftRows
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$ $a_{1,0}$
$a_{2,2}$ $a_{2,3}$ $a_{2,0}$ $a_{2,1}$
$a_{3,3}$ $a_{3,0}$ $a_{3,1}$ $a_{3,2}$

Mix Columns

- Read column i of State as a polynomial: $a_{0,i} + a_{1,i}x + a_{2,i}x^2 + a_{3,i}x^3$
- Multiply this polynomial with the constant polynomial $c(x) = 03 \cdot x^2 + 01 \cdot x^2 + 01 \cdot x + 02$ and reduce modulo $x^4 - 1$. This gives a new polynomial: $b_{0,i} + b_{1,i}x + b_{2,i}x^2 + b_{3,i}x^3$



$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,0}$ $a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,0}$ $a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
MixColumns
$\otimes c(x)$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

AES Encryption

- From the key k derive $h + 1$ round keys $k_0, k_1, \ldots, k_h$.
- Here is a description of *encryption*:

```
State ← *plaintext*
State ← State $\oplus k_0$
for $i = 1 \ldots h - 1$ do
    State ← SubBytes(State)
    State ← ShiftRows(State)
    State ← MixColumns(State)
    State ← State $\oplus k_i$
State ← SubBytes(State)
State ← ShiftRows(State)
State ← State $\oplus k_h$
*ciphertext* ← State
```

- Note that in the final round, `MixColumns` is not applied. This helps make decryption more similar to encryption, and thus is useful when implementing AES. [Details omitted]

HASH FUNCTIONS

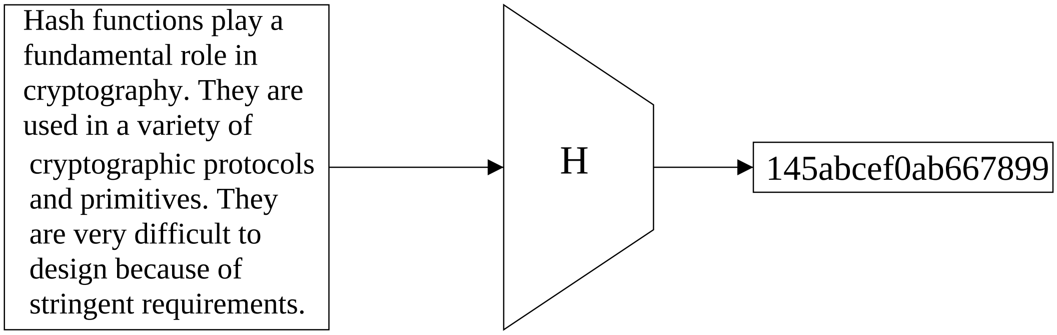
Outline

1. Definitions and Terminology
2. Applications of Hash Functions
3. Hash Functions from Block Ciphers
4. Generic Attacks
5. Iterated Hash Functions
6. MDx-Family of Hash Functions
7. How to Exploit a Single Hash Collision

Definitions and Terminology

- Hash functions play a fundamental role in cryptography.
- They are used in a variety of cryptographic primitives and protocols.
- They are very difficult to design because of very stringent security and performance requirements.
- The only serious candidates available today are: SHA-1, RIPEMD-160, SHA-224, SHA-256, SHA-384, SHA-512 and SHA-3.

What is a Hash Function?



Hash functions play a fundamental role in cryptography. They are used in a variety of cryptographic protocols and primitives. They are very difficult to design because of stringent requirements.

See:

```
http://www.xorbin.com/tools/md5-hash-calculator
http://www.xorbin.com/tools/sha1-hash-calculator
http://www.xorbin.com/tools/sha256-hash-calculator
```

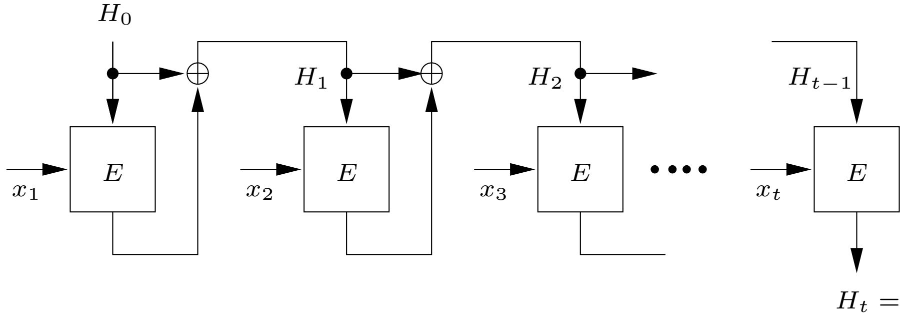
Definition of a Hash Function

- A *hash function* is a mapping H such that:
  - (i) H maps inputs of arbitrary lengths to outputs of a fixed length n: $H : \{0,1\}^* \longrightarrow \{0,1\}^n$.
    (More generally, H maps elements of a set S to a set T where $|S| > |T|$.)
  - (ii) $H(x)$ can be efficiently computed for all $x \in \{0,1\}^*$.
- H is called an n*-bit hash function*.
- $H(x)$ is called the *hash value*, *hash*, or *message digest* of x.
- Note: The description of a hash function is public. There are no secret keys.

Hash Functions from Block Ciphers

Davies-Meyer hash function.

- Let E_k be an n-bit block cipher with n-bit key k.
- Let IV be a fixed n-bit *initializing value*.
- To compute $H(x)$, do:
  - Break up $x \,\|\, 1$ into n-bit blocks: $\overline{x} = x_1, x_2, \ldots, x_t$, padding out the last block with 0 bits if necessary.
  - Define $H_0 = IV$.
  - Compute $H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$ for $i = 1, 2, \ldots, t$.
  - Define $H(x) = H_t$.



Some Applications of Hash Functions (1)

Hash functions are the Swiss Army knife of cryptography.

Hash functions are used for all kinds of applications they were not designed for.

Main reason for this widespread use of hash functions is *speed*.

Preimage Resistance

$H : \{0,1\}^* \longrightarrow \{0,1\}^n$.

Password protection on a multi-user computer system:

- Server stores $(\text{userid}, H(\text{password}))$ in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
- Requires preimage-resistance.

Preimage resistance: Given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible probability of success) any input x such that $H(x) = y$.

- x is called a *preimage* of y.

Note: $x \in_R S$ means that x is chosen independently and uniformly at random from S.

2nd Preimage Resistance

$H : \{0,1\}^* \longrightarrow \{0,1\}^n$.

Modification Detection Codes (MDCs).

- ► To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
- ► e.g. virus protection.
- ► Requires 2nd preimage resistance.

2nd preimage resistance: Given an input $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \neq x$ such that $H(x) = H(x')$.

Collision Resistance

Message digests for digital signature schemes:

- ► For reasons of efficiency, instead of signing a (long) message, the (much shorter) message digest is signed.
- ► Requires preimage-resistance, 2nd preimage resistance, and collision resistance. (More on this later)
- ► To see why collision resistance is required:
  - Suppose that the legitimate signer Alice can find two messages x_1 and x_2, with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.
  - Alice can sign x_1 and later claim to have signed x_2.

Collision resistance: It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs x, x' such that $H(x) = H(x')$.

- ► The pair (x, x') is called a *collision* for H.

Typical Cryptographic Requirements

$H : \{0,1\}^* \longrightarrow \{0,1\}^n$.

- *Preimage resistance*: Given a hash value $y \in_R \{0,1\}^n$, it is computationally infeasible to find (with non-negligible probability of success) any input x such that $H(x) = y$.
    - x is called a *preimage* of y.
- *2nd preimage resistance*: Given an input $x \in_R \{0,1\}^*$, it is computationally infeasible to find (with non-negligible probability of success) a second input $x' \neq x$ such that $H(x) = H(x')$.
- *Collision resistance*: It is computationally infeasible to find (with non-negligible probability of success) two distinct inputs x, x' such that $H(x) = H(x')$.
    - The pair (x, x') is called a *collision* for H.

Some Relationships Between Properties

1. Collision res. implies 2nd preimage resistance. [Why?]
2. 2nd preimage resistance does not guarantee collision resistance. [Why?]
3. Collision resistance does not guarantee preimage resistance. Justification:
    - Let $H : \{0,1\}^* \to \{0,1\}^n$ be a collision-resistant hash function.
    - Consider $\overline{H} : \{0,1\}^* \to \{0,1\}^{n+1}$ defined by:

$$\overline{H}(x) = \begin{cases} 1 \,\|\, x, & \text{if } x \in \{0,1\}^n \\ 0 \,\|\, H(x), & \text{if } x \notin \{0,1\}^n. \end{cases}$$

    - Then $\overline{H}$ is collision-resistant (since H is). [Why?]
    - And $\overline{H}$ is not preimage-resistant because preimages can be easily found for (at least) half of all $y \in \{0,1\}^{n+1}$.

Some Relationships Between Properties (2)

However, if H is “somewhat uniform” (i.e., all hash values have roughly the same number of preimages), then the following argument shows that collision resistance of H does indeed guarantee preimage resistance:

Suppose that H is not preimage resistant. Select arbitrary $x \in \{0,1\}^*$ and compute $y = H(x)$. Find a preimage x' of y; this is successful with some negligible probability. Then, if H is uniform, we expect that $x' \neq x$ with very high probability. Thus, we have efficiently found a collision (x, x') for H, whence H is not collision resistant.

Terminology

- A hash function that is preimage resistant is sometimes called a *one-way hash function* (OWHF).
- A hash function that is collision resistant is sometimes called a *collision-resistant hash function* (CRHF).
- A hash function that is both preimage resistance and collision resistant is called a *cryptographic hash function*.

Some Applications of Hash Functions (2)

1. *Password protection* on a multi-user computer system:
   - ► Server stores $(\text{userid}, H(\text{password}))$ in a password file. Thus, if an attacker gets a copy of the password file, she does not learn any passwords.
   - ► Requires preimage-resistance.
2. *Modification Detection Codes* (MDCs).
   - ► To ensure that a message m is not modified by unauthorized means, one computes $H(m)$ and protects $H(m)$ from unauthorized modification.
   - ► e.g. virus protection.
   - ► Requires 2nd preimage resistance.

Some Applications of Hash Functions (3)

3. *Message digests for digital signature schemes*:
   - ► For reasons of efficiency, instead of signing a (long) message, the (much shorter) message digest is signed.
   - ► Requires preimage-resistance, 2nd preimage resistance, and collision resistance. (More on this later)
   - ► To see why collision resistance is required:
     - Suppose that the legitimate signer Alice can find two messages x_1 and x_2, with $x_1 \neq x_2$ and $H(x_1) = H(x_2)$.
     - Alice can sign x_1 and later claim to have signed x_2.
4. *Message Authentication Codes* (MACs). (More on this later)
   - ► Provides data integrity & data origin authentication.

Some Applications of Hash Functions (4)

5. *Pseudorandom bit generation*:
   - Distilling random bits s from several "random" sources $x_1, x_2, \ldots, x_t$.
   - Output $s = H(x_1, x_2, \ldots, x_t)$.

6. *Key derivation function* (KDF): Deriving a cryptographic key from a shared secret. (More on this later)

Notes:

- Collision resistance is not always necessary.
- Depending on the application, other properties may be needed, for example 'near-collision resistance', 'partial preimage resistance', ...

Generic Attacks

- A *generic* attack on hash functions $H : \{0,1\}^* \to \{0,1\}^n$ does not exploit any properties a specific hash function may have.
- In the *analysis* of a generic attack, we view H as a *random function* in the sense that for each $x \in \{0,1\}^*$, the value $y = H(x)$ was chosen by selecting $y \in_R \{0,1\}^n$.
- From a security point of view, a random function is an *ideal* hash function. However, random functions are not suitable for practical applications because they cannot be compactly stored.

Generic Attack for Finding Preimages

- Given $y \in_R \{0,1\}^n$, select arbitrary $x \in \{0,1\}^*$ until $H(x) = y$.
- Expected number of steps is $\approx 2^n$.
  (Here, a 'step' is a hash function evaluation.)
- This attack is infeasible if $n \geq 80$.

Note: It has been proven that this generic attack for finding preimages is optimal, i.e., no better *generic* attack exists.

Generic Attack for Finding Collisions

- Select arbitrary $x \in \{0,1\}^*$ and store $(H(x), x)$ in a table sorted by first entry. Continue until a collision is found.
- Expected number of steps: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$ (by birthday paradox). (Here, a step is a hash function evaluation.)
- It has been proven that this generic attack for finding collisions is optimal in terms of the number of hash function evaluations.
- Expected space required: $\sqrt{\pi 2^n/2} \approx \sqrt{2^n}$.
- This attack is infeasible if $n \geq 160$.
- If $n = 128$:
  - Expected running time: 2^{64} steps. [barely feasible]
  - Expected space required: 7×10^8 Tbytes. [infeasible]

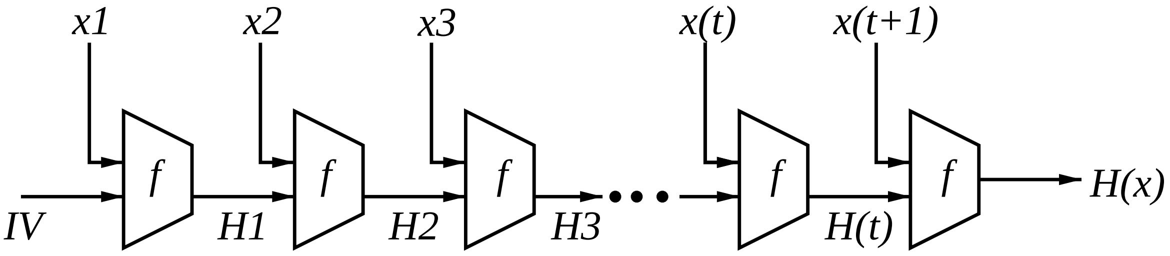
VW Parallel Collision Search

- *VW: Van Oorschot & Wiener* (1993)
- See class notes for a description of the VW search algorithm.
- Very small space requirements.
- Easy to parallelize: [see class notes]
  - m-fold speedup with m processors.
- Described a 1996 US$10 million machine which can find collisions for 128-bit hash functions in 21 days. Collision finding for 160-bit hash functions such as SHA-1 would take about 3500 years on this machine.
- The collision-finding algorithm can easily be modified to find "meaningful" collisions. [see class notes]

Iterated Hash Functions (Merkle Meta-Method)

- Components:
  - Fixed *initializing value* $IV \in \{0,1\}^n$.
  - *Compression function* $f : \{0,1\}^{n+r} \longrightarrow \{0,1\}^n$.
- To compute $H(x)$ where x has bitlength $b < 2^r$ do:
  - Break up x into r-bit blocks: $\overline{x} = x_1, x_2, \ldots, x_t$, padding out the last block with 0 bits if necessary.
  - Define x_{t+1}, the *length-block*, to hold the right-justified binary representation of b.
  - Define $H_0 = IV$.
  - Compute $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t+1$.
  - H_i's are called *chaining variables*.
  - Define $H(x) = H_{t+1}$.

Collision Resistance of Iterated Hash Functions



- Theorem (Merkle): If the compression function f is collision resistant, then the function H is also collision resistant.
  Proof: See class notes.
- Merkle's theorem reduces the problem of finding collision-resistant hash functions to that of finding collision-resistant compression functions.

MDx-Family of Hash Functions

- *MDx* is a family of iterated hash functions.
- *MD4* was proposed by Ron Rivest in 1990.
- MD4 has 128-bit outputs.
- Dobbertin (1996) found "meaningful" collisions in a *few seconds* on a PC.
- Wang et al. (2004) found collisions for MD4 *by hand*.

Prof. Xiaoyun Wang

- Leurent (2008) discovered an algorithm for finding MD4 preimages in 2^{102} steps.

MD5 Hash Function

- *MD5* is a strengthened version of MD4.
- Designed by Rivest in 1991.
- MD5 has 128-bit outputs.
- Dobbertin (1996) found collisions for the MD5 compression function.
- Wang and Yu (2004) found MD5 collisions in 2^{39} steps.
- Klima (2006) found MD5 collisions in *31 seconds* on a notebook computer.
- Sasaki & Aoki (2009) discovered a method for finding preimages for MD5 in $2^{123.4}$ steps.

MD5 Hash Function (2)

- *Summary*: MD5 should not be used if collision resistance is required, but is probably okay as a one-way hash function.
- MD5 is still used today
  More on this later....
- (2006) MD5 shows up about 850 times in Windows source code.

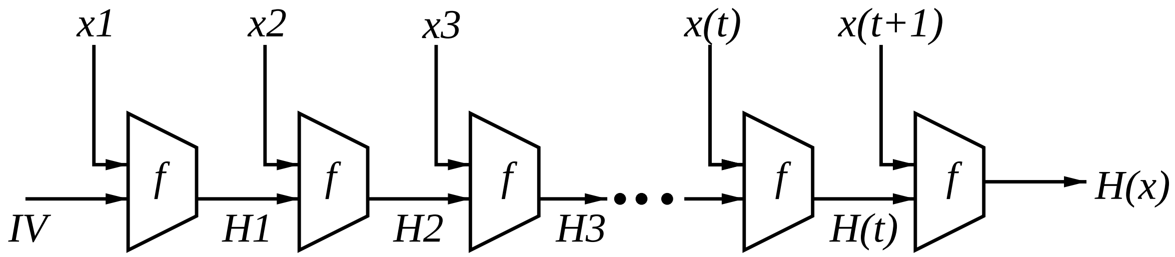
SHA-1

- Secure Hash Algorithm (*SHA*) was designed by *NSA* and published by NIST in 1993 (FIPS 180).
- *160-bit* iterated hash function, based on MD4.
- Slightly modified to *SHA-1* (FIPS 180-1) in 1994 in order to fix an (undisclosed) security weakness.
  - Wang et al. (2005) found collisions for SHA in 2^{39} steps.
- Wang et al. (2005) discovered a collision-finding algorithm for SHA-1 that takes 2^{63} steps.
  - No collisions for SHA-1 have been found as yet.
- No preimage or 2nd preimage attacks (that are faster than the generic attacks) are known for SHA-1.

Microsoft’s SHA-1 Plan

- See: `http://tinyurl.com/MicrosoftSHA1`
- As of February 14, 2017: SSL server-authentication certificates that use SHA-1 will be considered invalid.
- Still unaffected:
  - Code signature file hashes
  - Code signing certificates
  - Timestamp signature hashes
  - Timestaming certificates
  - etc.

High-Level Description of SHA-1



- Iterated hash function (Merkle meta-method).
- $n = 160$, $r = 512$.
- Compression function is $f : \{0,1\}^{160+512} \longrightarrow \{0,1\}^{160}$.
- Input: bitstring x of arbitrary bitlength $b \geq 0$.
- Output: 160-bit hash value $H(x)$ of x.

SHA-1 Notation

A, B, C, D, E	32-bit quantities.
$+$	addition modulo 2^{32}.
$\overline{A}$	bitwise complement.
$A \hookleftarrow s$	rotate A left through s positions.
AB	bitwise AND.
$A \vee B$	bitwise inclusive-OR.
$A \oplus B$	bitwise exclusive-OR.
$f(A, B, C)$	$AB \vee \overline{A}C$.
$g(A, B, C)$	$AB \vee AC \vee BC$.
$h(A, B, C)$	$A \oplus B \oplus C$.

SHA-1 Constants

- 32-bit initial chaining values (IVs):
  - h_1 = `0x67452301`, h_2 = `0xefcdab89`,
    h_3 = `0x98badcfe`, h_4 = `0x10325476`,
    h_5 = `0xc3d2e1f0`.
- Per-round integer additive constants:
  - y_1 = `0x5a827999`, y_2 = `0x6ed9eba1`,
    y_3 = `0x8f1bbcdc`, y_4 = `0xca62c1d6`.

SHA-1 Preprocessing

- Pad x (with 1 followed by 0's) so that its bitlength is 64 less than a multiple of 512.
- Append a 64-bit representation of $b \bmod 2^{64}$.
- The formatted input is $x_0, x_1, \ldots, x_{16m-1}$, where each x_i is a 32-bit word.
- Initialize chaining variables:
  $(H_1, H_2, H_3, H_4, H_5) \leftarrow (h_1, h_2, h_3, h_4, h_5)$.

SHA-1 Processing

For each i from 0 to $m-1$ do the following:

- Copy the ith block of sixteen 32-bit words into temporary storage: $X_j \leftarrow x_{16i+j},\ 0 \leq j \leq 15$.
  Now process these as follows in four 20-step rounds before updating the chaining variables.
- Expand 16-word block into 80-word block:
  For j from 16 to 79,
  $X_j \leftarrow (\ X_{j-3} \oplus X_{j-8} \oplus X_{j-14} \oplus X_{j-16}) \hookleftarrow 1$.
- Initialize working variables:
  $(A, B, C, D, E) \leftarrow (H_1, H_2, H_3, H_4, H_5)$.

Note: The rotate operation "$\hookleftarrow 1$" is not present in SHA. This is the only difference between SHA and SHA-1.

SHA-1 Processing (2)

- (*Round 1*) For j from 0 to 19 do:
  $t \leftarrow ((A \hookleftarrow 5) + f(B, C, D) + E + X_j + y_1)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 2*) For j from 20 to 39 do:
  $t \leftarrow ((A \hookleftarrow 5) + h(B, C, D) + E + X_j + y_2)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 3*) For j from 40 to 59 do:
  $t \leftarrow ((A \hookleftarrow 5) + g(B, C, D) + E + X_j + y_3)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- (*Round 4*) For j from 60 to 79 do:
  $t \leftarrow ((A \hookleftarrow 5) + h(B, C, D) + E + X_j + y_4)$,
  $(A, B, C, D, E) \leftarrow (t, A, B \hookleftarrow 30, C, D)$.
- Update chaining values:
  $(H_1, H_2, H_3, H_4, H_5) \leftarrow (H_1{+}A, H_2{+}B, H_3{+}C, H_4{+}D, H_5{+}E)$.

SHA-1 Completion

Output: $H(x) = H_1 \parallel H_2 \parallel H_3 \parallel H_4 \parallel H_5$.

Open problem:
Find a message $x \in \{0,1\}^*$ such that SHA-1$(x) = 0$.

SHA-2

- In 2001, *NSA* proposed variable output-length versions of SHA-1.
- Output lengths are *224* bits (SHA-224), *256* bits (SHA-256), *384* bits (SHA-384) and *512* bits (SHA-512).
- Note: The security levels of these hash functions against collision-finding attacks is the same as the security levels of Triple-DES, AES-128, AES-192 and AES-256 against exhaustive search attacks.
- The SHA-2 hash functions are standardized in FIPS 180-2.
- No weaknesses in SHA-224, SHA-256, SHA-384 or SHA-512 have been found.

Performance

[Wei Dai] `www.cryptopp.com/benchmarks.html`
Speed benchmarks (2009) for a software implementation on an Intel Core 2 1.83 GHz processor.

Algorithm	Speed (Mbits/sec)
DES	32
3DES	13
AES	139
RC4	126
MD5	255
RIPEMD-160	106
SHA-1	153
SHA-256	111
SHA-512	99

SHA-3

- ► The SHA-2 design is similar to SHA-1, and thus there are lingering concerns that the SHA-1 weaknesses will eventually extend to SHA-2.
- ► SHA-3: NIST hash function competition.
  - • 64 candidates submitted by Oct 31 *2008* deadline.
  - • 51 were accepted for the first round.
  - • *2009*: 14 were selected for the second round.
  - • Extensive analysis by cryptographers.
  - • *2010*: 5 were selected for the final round.
  - • Extensive analysis by cryptographers.
  - • *October 2 2012*: *Keecak* was selected as the winner.
- ► Keecak uses the “sponge construction” and *not* the Merkle iterated hash design.
- ► It remains to be seen if SHA-3 will be deployed in practice.

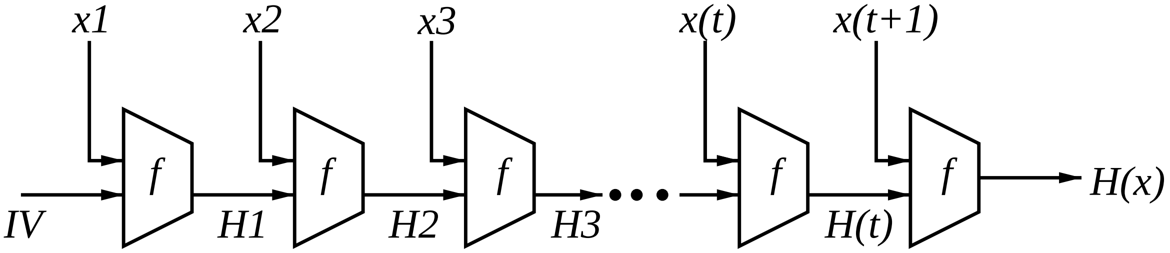
NIST’s Policy on Hash Functions

- August 5, 2015
- See: `http://csrc.nist.gov/groups/ST/hash/policy.html`
- Stop using SHA-1 for digital signatures and other applications that require collision resistance.
- May still use SHA-1 for HMAC, KDFs, and random number generators.
- Must use SHA-2 for all applications that employ secure hash algorithms.
- SHA-3 may also be used, but this is not required.

How to Exploit a Single Hash Collision

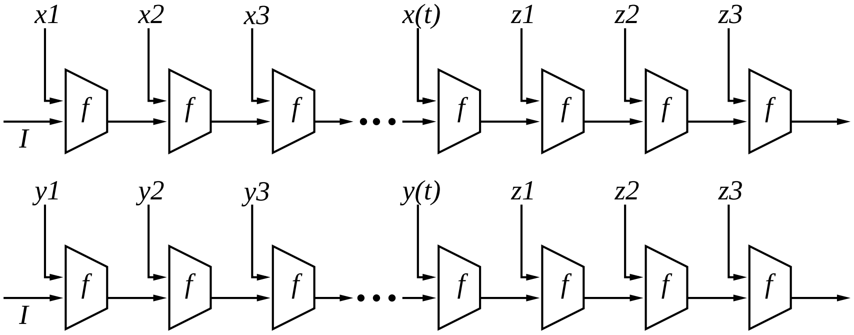
- Let H be a hash function (such as MD5 or SHA-1).
- Suppose that we can find two different messages x and y such that $H(x) = H(y)$.
- Suppose also that the collision-finding method we use does not allow us to control the structure of x and y, so that these messages are essentially meaningless.
- Suppose also that the collision-finding methods takes considerable (but feasible) time.
- Question: How can an attacker, who has expended considerable resources to find two meaningless messages x and y that collide, make repeated use of this collision in a practical setting?

MD5 and SHA-1



- MD5 is a 128-bit iterated hash function with $r = 512$.
- SHA-1 is a 160-bit iterated hash function with $r = 512$.
- MD5 and SHA-1 have slightly different padding functions (and length-blocks) than in Merkle's meta-method.

Extending a Collision



- Notation: For a t-block message $x = (x_1, x_2, \ldots, x_t)$ and n-bit string I, define $F(I, x) = H_t$, where $H_0 = I$ and $H_i = f(H_{i-1}, x_i)$ for $i = 1, 2, \ldots, t$.
- Observation: Suppose x and y are two messages of the same block-length such that $F(I, x) = F(I, y)$. Then $F(I, x \,\|\, z) = F(I, y \,\|\, z)$ for *any* message z. Furthermore, if $I = IV$, then $H(x, z) = H(y, z)$ for any message z.

Wang's Collision-Finding Attack on SHA-1

- Fix any n-bit string I.
- Wang's attack finds two (different) 1-block messages $x = x_1$ and $y = y_1$ such that $F(I, x_1) = F(I, y_1)$.
- The attack gives limited, but not complete, control over x_1 and y_1.
- The attack takes about 2^{63} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in SHA-1), Wang's attack can be used to find two one-block messages x and y such that SHA-1(x)=SHA-1(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

Wang's Collision-Finding Attack on MD5

- Fix any n-bit string I.
- Wang's attack finds two (different) two-block messages $x = (x_1, x_2)$ and $y = (y_1, y_2)$ such that $F(I, x) = F(I, y)$.
- The attack gives limited, but not complete, control over x and y.
- The attack takes about 2^{39} steps.
- By selecting $I = IV$ (where IV is the fixed initialization vector specified in MD5), Wang's attack can be used to find two two-block messages x and y such that MD5(x)=MD5(y). The attacker does not have much control over x and y, so these messages are essentially meaningless.

Exploiting Wang's Collisions

- Suppose that MD5 is being used by Alice in a hash-then-sign signature scheme.
  [The example also works for SHA-1]
- Let M_1 and M_2 be two documents in *postscript* format. Suppose that M_1 is "harmless" for Alice, and M_2 is "harmful" for Alice. We show how an MD5 collision found using Wang's attack can be used to find two new postscript files $\widehat{M_1}$ and $\widehat{M_2}$ such that:
  1. When $\widehat{M_1}$ is viewed (using a standard postscript viewer) or printed (on a postscript printer), it looks the same as M_1. Similarly for $\widehat{M_2}$.
  2. MD5($\widehat{M_1}$) = MD5($\widehat{M_2}$).

Exploiting Wang's Collisions (2)

- The attacker Eve sends the postscript file $\widehat{M_1}$ to Alice. Alice views or prints the file, and then signs it and returns to Eve. Since MD5($\widehat{M_1}$) = MD5($\widehat{M_2}$), Eve also has Alice's signature on $\widehat{M_2}$.

Details of Daum and Lucks' Attack

The attacker Eve does the following:

1. Select a postscript "preamble" so that the string

   $$p = \text{"preamble ("}$$

   has bitlength a multiple of the block-length r. (If necessary, comments can be added as padding.)
2. Compute $I = F(IV, p)$.
3. Use Wang's attack to find two distinct two-block messages x and y such that $F(I, x) = F(I, y)$.
4. Select any two postscript files M_1 and M_2. Let T_1, T_2 be the postscript commands for displaying M_1 and M_2.

Details of Daum and Lucks' Attack (2)

5. Set $\widehat{M_1}$ = preamble (x)(x)eq{T_1}{T_2}
   and $\widehat{M_2}$ = preamble (y)(x)eq{T_1}{T_2}

Why this works:

1. MD5($\widehat{M_1}$)= MD5($\widehat{M_2}$)!!
2. Postscript interprets the commands (S_1)(S_2)eq{T_1}{T_2} as follows: If $S_1 = S_2$, then execute the commands T_1; else execute the commands T_2.
   Hence, if $\widehat{M_1}$ is viewed, then M_1 is displayed.
   If $\widehat{M_2}$ is viewed, then M_2 is displayed.

Notes on Daum and Lucks' Attack

- Of course, careful examination of the postscript files $\widehat{M_1}$ and $\widehat{M_2}$ would reveal Eve's deception – but no one reads raw postscript code before viewing or printing a postscript document.
- The collision x, y can be reused for any two postscript files M_1 and M_2.
- *Gebhardt, Illies and Schindler* (2005): Similar attacks on documents in PDF, TIFF and Word formats.
- Colliding arbitrary binary executable programs: `http://www.mscs.dal.ca/∼selinger/md5collision`
- Moral of this story: Attacks only get better; they never get worse. Don't be quick to dismiss attacks on cryptographic schemes, even if the attacks appear at first to have limited practical value.

Flame Malware

- Discovered in May 2012.
- Highly sophisticated espionage tool.
- Targeted computers in Iran and the Middle East.
- Suspected to originate from the US and/or Israeli government; US government has denied all responsibility.
- Contains a forged Microsoft certificate for Windows code signing.
- Forged certificate used a new, "zero-day MD5 chosen-prefix" collision attack.
- Microsoft no longer allows the use of MD5 for code signing.

MESSAGE AUTHENTICATION CODE SCHEMES

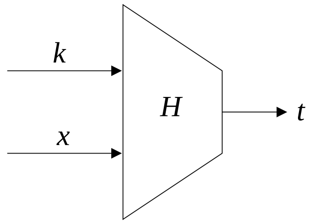
Outline

1. Definition
2. Applications
3. Generic Attacks
4. MACs Based on Block Ciphers
5. MACs Based on Hash Functions
6. Authenticated Encryption

Definition

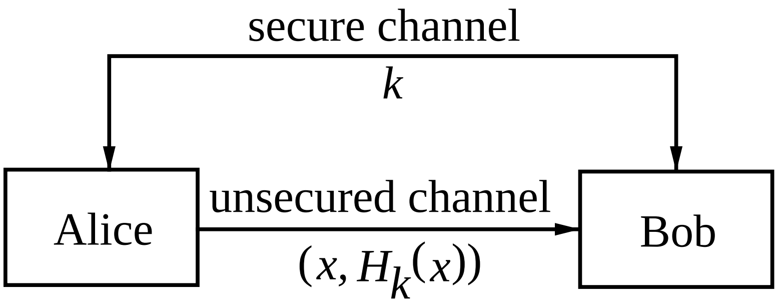
- A *message authentication code (MAC)* scheme is a family of functions $H_k : \{0,1\}^* \longrightarrow \{0,1\}^n$ parameterized by an ℓ-bit key k, where each function H_k can be efficiently computed.

- $t = H_k(x)$ is called the *MAC* or *tag* of x.



- MAC schemes are used for providing (symmetric-key) data integrity and data origin authentication.

Applications of MAC Schemes



- To provide *data integrity* and *data origin authentication*:
  1. Alice and Bob establish a secret key $k \in \{0,1\}^\ell$.
  2. Alice computes $t = H_k(x)$ and sends (x, t) to Bob.
  3. Bob verifies that $t = H_k(x)$.
- Note: No confidentiality or non-repudiation.
- To avoid *replay*, add a timestamp or sequence number.
- MACs are widely used in banking applications.

Security Definition

- Let k be the secret key shared by Alice and Bob.
- The adversary does not know k, but is allowed to obtain (from Alice or Bob) tags for messages of her choosing. The adversary's goal is to obtain the tag of any message whose tag she did not already obtain from Alice or Bob.
- Definition: A MAC scheme is *secure* if given some MAC tags $H_k(x_i)$ for x_i's of one's own choosing, it is computationally infeasible to compute (with non-negligible probability of success) a pair $(x, H_k(x))$ for any new message x.
  - That is, the MAC scheme must be *existentially unforgeable against chosen-message attack*.
- Note: A secure MAC scheme can be used to provide *data integrity* and *data origin authentication*.

Generic Attacks on MAC schemes

Guessing the MAC of a message x:

- Select $y \in_R \{0, 1\}^n$ and guess that $H_k(x) = y$.
- Assuming that H_k is a random function, the probability of success is $1/2^n$.
- Note: Guesses cannot be directly checked.
- Depending on the application where the MAC algorithm is employed, one could choose n as small as 32 (say). In general, $n \geq 80$ is preferred.

Generic Attacks (2)

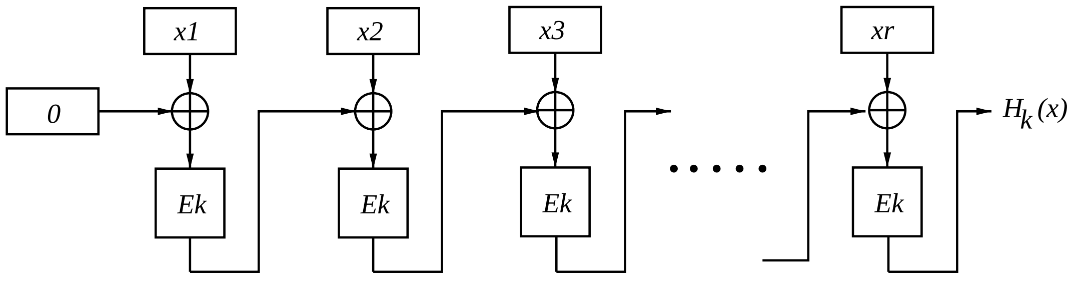
Exhaustive search on the key space:

- Given r known message-MAC pairs: $(x_1, t_1), \ldots, (x_r, t_r)$, one can check whether a guess k of the key is correct by verifying that $H_k(x_i) = t_i$, for $i = 1, 2, \ldots, r$.
- Assuming that the H_k's are random functions, the expected number of keys for which the tags verify is $1 + \text{FK} = 1 + (2^\ell - 1)/2^{nr}$.
  - Example: If $\ell = 56$, $n = 64$, $r = 2$, then FK $\approx 1/2^{72}$.
- Expected number of steps is $\approx 2^\ell$.
- Exhaustive search is infeasible if $\ell \geq 80$.

MACs Based on Block Ciphers

CBC-MAC

- Let E be an n-bit block cipher with key space $\{0,1\}^\ell$.
- Assumption: Suppose that plaintext messages all have lengths that are multiplies of n.
- To compute $H_k(x)$:
  1. Divide x into n-bit blocks $x_1, x_2, \ldots, x_r$.
  2. Compute $H_1 = E_k(x_1)$.
  3. For $2 \leq i \leq r$, compute $H_i = E_k(H_{i-1} \oplus x_i)$.
  4. Then $H_k(x) = H_r$.



CBC-MAC (2)

- *DES CBC-MAC* (with $\ell = 56, n = 64$) is widely used in banking applications.
- Rigorous security analysis [Bellare, Kilian & Rogaway 1994]:

  Informal statement of a Theorem: Suppose that E is an "ideal" encryption scheme. (That is, for each $k \in \{0,1\}^\ell$, $E_k : \{0,1\}^n \to \{0,1\}^n$ is a 'random' permutation.) Then CBC-MAC *with fixed-length inputs* is a secure MAC scheme.

Security of CBC-MAC

- CBC-MAC (as described on slide 189 without additional measures) is not secure if variable length messages are allowed.
- Here is a *chosen-message attack* on CBC-MAC:
  1. Let x_1 be an n-bit block.
  2. Let (x_1, t_1) be a known message-MAC pair.
  3. Request the MAC t_2 of the message t_1.
  4. Then t_2 is also the MAC of the 2-block message $(x_1, 0)$. (Since $t_2 = E_k(E_k(x_1))$.)

Encrypted CBC-MAC (EMAC)

- One solution for variable-length messages is *Encrypted CBC-MAC*:
  - Encrypt the last block under a second key s: $H_{k,s}(x) = E_s(H_r)$.
  - For example, ANSI X9.19 specifies that $H_{k,s}(x) = E_k(E_s^{-1}(H_r))$.
- Rigorous security analysis [Petrank & Rackoff 2000]:

  Informal statement of a Theorem: Suppose that E is an "ideal" encryption scheme. Then EMAC is a secure MAC scheme (for inputs of any length).

MACs Based on Hash Functions

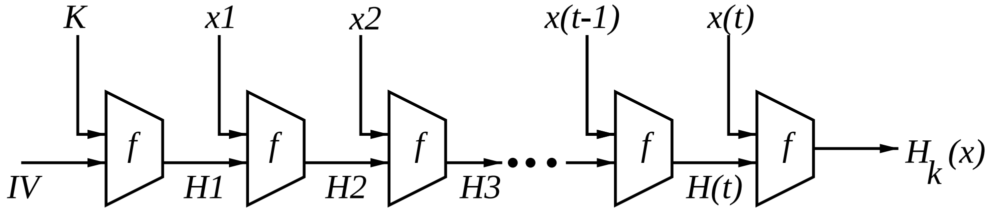
Hash functions were not originally designed for message authentication; in particular they are not "keyed" primitives.

Question: How to use them to construct secure MACs?

- Let H be an iterated n-bit hash function (without the length-block).
- Let $n + r$ be the input blocklength of the compression function $f : \{0,1\}^{n+r} \rightarrow \{0,1\}^n$.
  - Example: For SHA-1, $n = 160$, $r = 512$.
- Let $k \in \{0,1\}^n$.
- Let K denote k padded with $(r - n)$ 0's. (So K has bitlength r.)

Secret Prefix Method

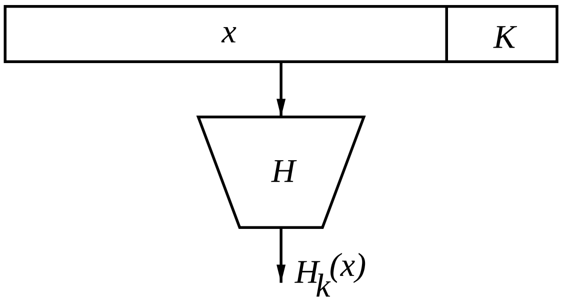
- <u>MAC definition</u>: $H_k(x) = H(K, x)$



- This is *insecure*. Here is a *length extension attack*:
  - Suppose that $(x, H_k(x))$ is known.
  - Suppose that the bitlength of x is a multiple of r.
  - Then $H_k(x \,\|\, y)$ can be computed for any y (without knowledge of k).
- Also insecure if a length block is postpended to $K \,\|\, x$ prior to application of H. <u>[Exercise]</u>

Secret Suffix Method

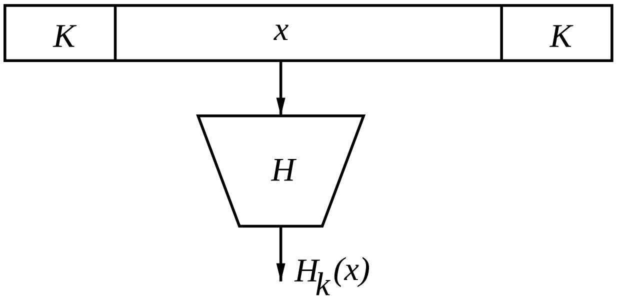
- <u>MAC definition</u>: $H_k(x) = H(x, K)$



- The attack on the secret prefix method does not work here.
- Suppose that a collision (x_1, x_2) can be found for H (i.e., $H(x_1) = H(x_2)$). We can assume that x_1 and x_2 both have bitlengths that are multiples of r.
  Thus $H(x_1, K) = H(x_2, K)$, and so $H_k(x_1) = H_k(x_2)$.
  Then the MAC for x_1 can be requested, giving the MAC for x_2.
  Hence *if H is not collision resistant*, then the secret suffix method MAC is *insecure*.

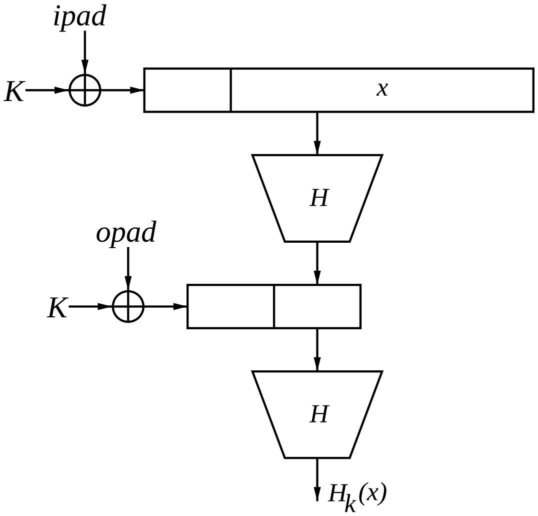
Envelope Method

- The MAC key is used both at the start and end of the MAC computation.
- MAC definition: $H_k(x) = H(K, x, K)$



- The envelope method appears to be secure (i.e., no serious attacks have been found).

HMAC



- “Hash-based” MAC; Bellare, Canetti & Krawczyk (1996).
- Define two r-bit strings (in hexadecimal notation): ipad = `0x36`, opad = `0x5C`; each repeated $r/8$ times.

- MAC definition: $H_k(x) = H(K \oplus \text{opad}, H(K \oplus \text{ipad}, x))$.

HMAC (2)

- Security analysis is rigorous:

  Informal statement of a Theorem: Suppose that the compression function f used in H is a secure MAC with fixed length messages and a secret IV as the key. Then HMAC is a secure MAC algorithm.
- Recommended for practice: use SHA-1 as the hash function.
  - The weaknesses found in SHA-1 do not appear to affect the security of HMAC.
- HMAC is specified in IETF RFC 2104 and FIPS 198. (IETF = Internet Engineering Task Force)
- HMAC is used in *IPsec* (Internet Protocol Security) and *SSL* (Secure Sockets Layer)/*TLS* (Transport Layer Security).

Authenticated Encryption (AE)

- A symmetric-key encryption scheme E provides *confidentiality*.
- A MAC scheme H provides *authentication* (i.e., *data integrity* and *data origin authentication*).
- Question: What if confidentiality and authentication are *both* required?
- First Method: *Encrypt-and-MAC*.
  - Alice sends $(c,t) = (E_{k_1}(m), H_{k_2}(m))$ to Bob, where m is the plaintext and k_1, k_2 are secret keys she shares with Bob.
  - Bob decrypts c to obtain $m = E_{k_1}^{-1}(c)$ and then verifies that $t = H_{k_2}(m)$.
  - However, this generic method might have some security vulnerabilities. [Why?]

Encrypt-then-MAC

- Second Method: *Encrypt-then-MAC*.
  - Alice sends $(c, t) = (E_{k_1}(m),\ H_{k_2}(E_{k_1}(m)))$ to Bob, where m is the plaintext and k_1, k_2 are secret keys she shares with Bob.
  - Bob first verifies that $t = H_{k_2}(c)$ and then decrypts c to obtain $m = E_{k_1}^{-1}(c)$.
- This method has been deemed to be secure, provided of course that the MAC and encryption schemes employed are secure.

Special-Purpose AE Schemes

Many specialized authenticated encryption scheme have been developed, the most popular of these being Galois/Counter Mode (*GCM*).

These modes can be faster than generic Encrypt-then-MAC, and also allow for the authentication (but not encryption) of “header” data.

[Details not covered in this course.]