Introduction to Security

Prof. Shlomo Kipnis
October 22, 2007

The World We Live In

- Personal computers (home or office)
- Data servers (databases and web serves)
- Phone systems (terminals, switches, routers)
- Cellular phones (voice, image, video, data)
- Hand-held appliances (laptops, PDA, GPS)
- Wireless communication (LAN, microwave, free air)
- Information kiosks (data, service, bank ATMs)
- Television systems (cable, satellite, free air)
- Embedded systems (in cars, in home appliances)
- Smart cards (identification, authorization, electronic cash)
- Many other systems . . .

What is Security?

- On the one hand:
  - Making sure that bad things do not happen
  - Reducing the chances that bad things will happen
  - Lowering the impact of bad things
  - Providing means to recover from bad things
- While on the other hand:
  - Allowing good things to happen
  - Managing the cost of the system
- Examples:
  - Home security
  - Car Security

Security Challenges

- Securing a variety of different systems
- Securing interfaces between different systems
- Different security goals and needs
- Attackers seek weakest link in the system
- Security people must protect all links in the system
- Maintaining system usability
- Keeping security costs under control

Threats & Attacks

- Unauthorized access
- System integrity loss
- Denial of service
- Computer viruses
- Trojan horses
- Information loss
- Data leaks
- Data manipulation
- Data fraud
- Data theft
- Data destruction
- Program manipulation

Some Security Breaches

- In 1988, a computer "worm" launched by Cornell graduate student, Robbert Morris Jr., infected 1000's of computers, causing them to shut down
- In 1994, $10.4 million dollar computer fraud by a group from Russia against Citibank
- In 1996, the USA DOJ and CIA home pages were defaced by vandals
- In 2000+, millions of attacks on governments, corporations, financial institutions, etc. every year. (Most are unsolved and are not reported.)

Eavesdropping and Packet Sniffing

- Description: Acquiring information without changing it
- Means: Packet sniffers, routers, gateways, capturing and filtering out packets
- Threats: Sniffing can be used to catch various information sent over the network
  - Login + Password
  - Credit card numbers
  - E-mails and other messages
  - Traffic analysis

Snooping

- Description: Acquiring information without modifying it
- Means: Browsing documents on disk or main memory
  - Using legitimate privileges (insiders)
  - Hacking into a system (outsiders)
  - Stealing laptops
  - Monitoring keyboard strokes
  - Observing timing information (covert channels)
- Threats:
  - Obtaining sensitive information (files with credit card numbers)
  - Discovering passwords, secret keys, etc.

Tampering

- Description: Modifying or destroying stored data
- Means: Insiders misusing privileges or outsiders breaking into system
- Threats:
  - Change records – school grades, prison records, tax payers' debts (NY $13 million property tax fraud)
  - Erase audit trails (by hacker)
  - Plant Trojan-horses for password gaining, and other uses

Spoofing

- Description: Impersonating other users or computers to obtain privileges
- Means:
  - Account stealing, password guessing, social engineering
  - IP spoofing: E-mail forging, false IP From address, hijacking IP connections
- Threats:
  - Forged messages (Dartmouth's "exam is cancelled")
  - Denial of Service (IP attacks, SYN attacks, Ping-of-Death)
  - Information sniffing (Princeton "WWW Spoofing" attack)

Jamming

- Description: Disabling a system or service
- Means: Engaging host in numerous (legitimate) activities until exhausting its resources; spoofing return addresses to avoid tracing
- Threats:
  - Consume all resources on the attacked machines, e.g., memory (SYN attack), disk (E-mail attack)
  - Exploit bug to shut down hosts (ping-of-death)

Code Injection

- Description: Injecting malicious code to execute on host with high privileges and infecting other hosts
- Means:
  - Virus: attached to executable, spread through infected floppy disks, E-mail attachments, macros
  - Worm: replicate over the Internet
- Threats:
  - Everything…

The Melissa Virus

- E-mail message; launched around end of March 1999
  - Subject: `Important Message From <name>`
  - Body: `Here is that document you asked for ... don't show anyone else ;-) +` set of pornographic links
  - Attachment: Word file with infectious macro
    - Registers itself, to avoid duplication
    - Modifies `Normal.dot` template
    - Sends Email to top 50 addresses in Outlook address book
  - When date+month matches hour+minutes, inserts `"Twenty-two points, plus triple-word-score, plus fifty points for using all my letters. Game's over. I'm outta here."`
  - Impact: Within less than 3 days, over 100,000 hosts. Some hosts had to shut down because of E-mail volume

Exploiting Flaws

- Exploit vulnerabilities in software to penetrate systems
  - Buffer overflow (e.g., 'finger', Internet Worm, Web Site apps)
  - Mobile code security flaws (Java, ActiveX)
  - Bad human engineering (Unix administrator logins, sendmail)
- Knowledge spreads faster than remedy
  - Hacker bulletins
  - Advisories:
    - Flaws/fixes repositories, e.g., CERT
    - Publicly available software kits to detect known vulnerabilities, e.g., SATAN, ISS
    - But they are not always followed readily, and are often used to the advantage of hackers
  - Publicly available hacker kits on the net, e.g., RootKit (Unix)

Password and Key Cracking

- Guessing: family member names, phone numbers, etc.
- Dictionary Attack: systematic search
  - Crack: dictionary attack extended with common patterns
    - crack is now employed by sys-admins and the passwd program
- Exhaustive search:
  - Crypt-analysis tools evolve continually
  - The Internet provides a massively parallel computing resource
- Crypt-analysis, bad generators, timing analysis
  - Kocher: discover key by the time it takes to encrypt with it
- Smart-card cracking via fault injection

Social Engineering

- Spoofing a "real system":
  - Login screen
  - Phone numbers
  - ATM story
- Spoofing a "service":
  - Stealing credit card numbers and PINs
  - Stealing passwords
- Agent-in-the-Middle Attacks
  - Special print of newspaper
  - Router, gateway, bulletin boards, etc.

Hackers – Who Are They?

- Academic Researchers:
  - Universities and research laboratories
  - Develop and analyze systems
- Consultant Hackers:
  - Employed by companies to identify weaknesses in systems
- Independent Hackers:
  - Work individually to identify weaknesses in systems
  - Motivation is social or personal
- Criminal Hackers:
  - Other side of the law
  - Motivation is mostly financial or political
- Amateur Hackers:
  - Get tools and codes from others
  - Non-professional (leave traces)

Political Hacking

- Motivation:
  - Political reasons
  - Industrial espionage
  - Military espionage
  - Information Warfare
- Resources:
  - Almost unlimited
- Risk:
  - Depends on the country
- Examples:
  - US-Russia, US-China, Israel-Hizballa

Commercial Hacking

- Motivation:
  - Kind of business intelligence
  - Gain business advantage
  - Cause direct or indirect loses
- Resources:
  - Limited by potential gain
- Risk:
  - Legal prosecution
- Examples:
  - Reverse engineering, service disruption

Social Hacking

- Motivation:
  - Political or social agenda
  - Social Hacktivism
  - Ideology
- Resources:
  - Institutional groups may have abundant resources
  - Independent groups may have public support
- Risk:
  - Almost no risk
- Examples:
  - CDC (Microsoft's SW), Russian hacker (Adobe's SW)

Financial Hacking

- Motivation:
  - Monetary gains
  - Theft of resources
- Resources:
  - Single individuals – little resources (but have the skills)
  - Organized crime – abundant financial resources
- Risk:
  - Target individuals and organizations
  - Considered to be low
- Examples:
  - Phone, TV, Web-sites, Credit card numbers

Individual Hacking

- Motivation:
  - Disgruntled employees
  - Curiosity
  - Intellectual challenge
  - Vandals
- Resources:
  - Scarce resources
  - Large community
- Risk:
  - Innocent mistakes
  - Almost no risk when crossing jurisdiction boundaries

Hacker Information Resources

- Newsgroups:
  - alt.2600
  - Cult of Dead Cow (CDC)
- Magazines:
  - Phrack
  - Web sites
- Conferences:
  - DefCon
- CERT:
  - Computer Response Emergency Team newsgroups

Recommended Books (I)

- General Security
  - "The NCSA Guide to Enterprise Security: Protecting Information Assets", Michel Kabay, McGraw Hill, 1996
  - "Practical Unix and Internet Security", Simson Garfinkel and Gene Spafford, O'Reilly & Associates, 1996
  - "Security Engineering: a Guide to Building Dependable Distributed Systems", Ross Anderson, John Wiley & Sons, 2001
  - "Firewalls and Internet Security", William Cheswick and Steven Bellovin, Addison Wesley, 1994 (and 2003 ???)
  - "Web Security Sourcebook", Aviel Rubin, Daniel Geer, and Marcus Ranum, John Wiley & Sons, 1997

Recommended Books (II)

- Applied Cryptography
  - "Cryptography and Network Security: Principles and Practice", William Stallings, 3-rd edition, Prentice Hall, 2003
  - "Network Security: Private Communication in a Public World", Charlie Kaufman, Radia Perlman, and Mike Speciner, 2-nd edition, Prentice Hall, 2002
  - "Handbook of Applied Cryptography", Alfred Menezes, Paul van Oorschott, and Scott Vanstone, CRC Press, 1997
  - "Applied Cryptography: Protocols, Algorithms, and Source Code in C", Bruce Schneier, 2-nd edition, John Wiley & Sons, 1996
  - "The Code Book", Simon Singh, Anchor Books, 1999

Recommended Books (III)

- Systems & Protocols
  - "Understanding Public-Key Infrastructure: Concepts, Standards, and Deployment Considerations", Carlisle Adams and Steve Lloyd, New Riders, 1999
  - "IPSEC: The New Security Standard for the Internet, Intranets, and Virtual Private Networks", Naganand Doraswamy and Dan Harkins, Prentice Hall, 1999
  - "SSL and TLS: Designing and Building Secure Systems", Eric Rescorla, Addison Wesley, 2001

Security Engineering

Prof. Shlomo Kipnis
October 24, 2007

Security Engineering

- Security Engineering – a disciplined approach to building security systems:
  - Security Policies
  - Security Threats
  - Security Goals
  - Security Methods
  - Security Layers
  - Security Principles

Security Policies

- Threats – what the dangers and attacks are
- Goals – what to protect, types of security services
- Entities – users, stations, devices, assets, etc.
- Actions – allowable actions in the system
- Permissions – who is allowed to do what in the system (ACL - Access Control Lists)
- Failures – they will happen, so better plan for them
- Recovery – what to do when (not if) failures happen, how to recover system assets and services

Security Threats

- Unauthorized access
- System integrity loss
- Denial of service
- Computer viruses
- Trojan horses
- Information loss
- Data leaks
- Data manipulation
- Data fraud
- Data theft
- Data destruction
- Program manipulation

Security Goals

- Entity Identification
- Entity Authentication
- Data Integrity
- Data Authenticity
- Data Confidentiality
- Data Source Verifiability
- Non-Repudiation
- Plausible Deniability
- Data Availability
- System Availability
- Data Reliability
- System Reliability
- Data Privacy
- Data Containment
- Entity Anonymity
- And more . . .

System Entities

- Users:
  - regular users, administrators, guests, clients, suppliers, etc.
- Workstations:
  - personal station, functioned server, end terminal, data server
- Gateways:
  - routers, gateways, firewalls, application proxies, data proxies
- Security Servers:
  - administration station, key centers, directory servers, etc.

System Actions

- File System:
  - open, close, read, write, execute, delete, modify, list, change permissions, etc.
- Banking System:
  - deposit, withdraw, open, close, transfer, get balance, etc.
- TV Viewing System:
  - view, purchase, store, play, connect to center, gamble, change permissions, etc.
- Router / Gateway:
  - allow packets, drop packets, apply checks to packets, etc.

System Permissions

- Tables of entities and actions:
  - Lists of subject, object, and actions
  - Grouping subjects, objects, and actions
  - Example: users, groups, files, file-actions
- Decision procedures given an entity and an action:
  - Static rules
  - Time-driven rules
  - Event-driven rules
  - Example: firewalls

System Failures

- Data loss (accidental or intentional)
- Data corruption (accidental or intentional)
- Data falling in the wrong hands
- Breaking into the system
- Leaking system secrets
- System unavailability
- System non-accessibility
- System non-operability

System Recoveries

- Data backup
- Redundant systems
- Operation resumption procedures
- Alternate operations
- Insurance

Security Methods

- Damage Prevention – keeping bad things away (guards, doors, firewalls, access controls, etc.)
- Damage Detection – detecting when bad things happen (cameras, alarms, monitoring, intrusion detection systems, log files, etc.)
- Damage Recovery – recovering from bad things (backup, alternate systems, insurance, etc.)

Security Layers

- User Awareness – assets, secrets, passwords, etc.
- Physical Protection – disks, cards, systems, locks, etc.
- Access Control – lists, devices, etc.
- Cryptography – encryption, authentication, signatures
- Backup – data, services, availability
- Monitoring – traffic, log files, etc.
- Redundancy – systems, data, people
- Deception – honey-pot systems
- Proactive Security – distributed, going after bad guys

Security Principles

- Least Privilege – action to be performed by lowest ranking entity that can perform it
- Trusted Components – identification of components and the level of trust in them
- Simple Designs – small and modular systems with well defined interfaces
- Paranoia – if your system / data is worth something – then, sooner or later, someone will break it. Stay alert.
- No Perfection – Return-On-Investment curves

Natural Faults

- Electrical power interruption – no computing
- Communication interruption – no remote access
- Hardware malfunctioning – wrong computing
- Software bugs – wrong computing
- Operator errors – loss of data or operation
- Fire – damage to hardware and data
- Flood – damage to hardware and data
- Earthquake – loss of operation and data
- Acts of War – loss of operation

Intended Attacks

- Sniffing – listening to the communication
- Snooping – stealing secrets or sensitive data
- Tampering – changing data, breaking into systems
- Spoofing – impersonating as another entity
- Code Injection – viruses, applets, etc.
- System Exploits – finding weak spots in systems
- Password Cracking – guessing or searching for passwords
- Social Engineering – convincing users to do things
- Physical Attacks – breaking devices
- And more . . . . .

System Solutions (I)

- Password systems
- Authentication devices
- Access control lists
- Firewalls
- Virus-detection software
- Intrusion detection systems
- Data backup systems
- Security evaluation software
- Hardware security devices

Security Solutions (II)

- Secure Virtual Private Networks
- Encryption software / hardware
- Signature software / hardware
- Trusted operating systems
- Trusted web server
- Software screening tools
- Program verification
- Sand-box model
- Network containment

Data Backup Systems

- What type of data needs to be backed-up?
  - Mostly user data
  - Usually not application code
- How much data needs to be backed-up?
  - Assume daily backup, 100 users, 100 MB per user per day
  - 10 GB of backup data per day……
- Ways to deal with complexity and volume:
  - Full backup
  - Incremental backup
  - Differential backup
- Backup security concerns

Full Backup

- Backup all files on the computer
- Expensive:
  - Volume to backup
  - Time to run the backup
  - Storage to be kept for many years
- Done at large time-intervals (month, year)
- Used as a complete system image at a given time
- Allows rebuilding the system as it was on a given date

Incremental Backup

- Backup files that have changed since last full backup
- Done at small time-intervals (days, weeks)
- Less expensive than full backup:
  - Volume of data to backup increases with time
  - Recovery time is fast (1 full backup + 1 incremental backup)
  - Storage can be reused after next full backup
- Can be used hierarchically and jointly with full backup:
  - Full backup once a year
  - Level 1 incremental backup once a month
  - Level 2 incremental backup once a day

Differential Backup

- Backup files that have changed since last differential backup
- Done at small time-intervals (days, weeks)
- Less expensive than full backup:
  - Volume of data to backup is more or less constant every time
  - Recovery time is variable (1 full backup + ??? differential backups)
  - Storage can be reused after next full backup
- Can be used hierarchically and jointly with full backup:
  - Full backup once a year
  - Level 1 differential backup once a month
  - Level 2 differential backup once a day

Backup Security Concerns

- Where is the backup data kept physically?
- Is the backup data kept encrypted?
  - Where is the encryption key?
  - Is the encryption key backed-up for future use?
  - Where is the encryption key kept?
  - Keep a hard copy of the key !!!
- How does the system determine which files to backup?
  - Timestamps by the operating system
  - Tables or flags for each file
  - What protects the time, table, flags?

Monitoring Systems

- Log Files:
  - user name, process id, exit status, time, etc.
- Shell History:
  - last few commands are kept in history
- Mail:
  - outgoing mail is kept in system
- Monitoring Software:
  - on the wires, in the file system, exceptional operations
- Audit Levels:
  - dictated by the US DoD "orange book"
- Log files need to be backed-up !!!

Redundant Systems

- CPUs in Server
  - Special systems with 2 or 3 CPUs per server
  - Used in mission-critical financial applications
- Servers in Farm
  - Several servers with access to same database
  - Used in database / web farms
  - Used also for load balancing
- Site Redundancy
  - Cold site – has data, operational within hours/days
  - Hot site – runs in parallel, operational immediately

Physical Security

- Physical Access
  - Buildings, computers, terminals, cables, data
- Transmission Lines
  - Electrical wires, optical fibers, wireless communication
- Display Devices
  - CRT screens, LCD screens, paper
- Magnetic Media
  - Disks, tapes, computer memory
- Computing devices
  - PC cards, smart cards

Physical Access

- Buildings
  - Doors, windows, locks, ventilation paths
- Computers
  - PCs, laptops, storage devices
- Terminals
  - Printers, screens, X-terminals, remote access
- Cables
  - Eavesdropping, networks, modems, entry points
- Data
  - Disks, tapes, backup

Transmission Lines

- Electrical Wires:
  - Electromagnetic radiation is emitted from wires carrying signals
  - Radiation can be detected up to meters from the wire
  - Signal can be re-constructed with commercial equipment
  - Same technology is used for testing communication equipment
  - Tempest standards
- Wireless Communication
  - Easy to eavesdrop a wireless line
- Optical Fibers
  - Signal can be stolen off fiber by bending the fiber and placing it in a liquid with similar diffraction index to that of wire

Display Devices

- CRT Screens:
  - Cathode Ray Tubes emit electromagnetic radiation
  - Radiation can be captured and analyzed
  - Signal / image can be reconstructed
  - Commercial products exist that do the job
- LCD Screens:
  - Less vulnerable to radiation analysis
- Paper:
  - Paper falling to the wrong hands…
  - Writing on paper leaves marks on paper / material behind it

Magnetic Media

- Delete:
  - Removes the pointer to the object
- Erase:
  - Writes zero on memory area
- Purge:
  - Wipes the memory area several times (?)
- Disks / Tapes / Magnetic Memory:
  - Digital signals are encoded by analog phenomena
  - Magnetic media have "state memory" of several generations
  - Digital values of 0 or 1 leave trace even after erased
  - Information is truly erased only after several generations of random values have been written

Computing Devices

- PC Cards:
  - Physically securing crypto-processors and keys
  - IBM 4758 processor board complies with FIPS-140 level 4
  - Tamper resistance against break-ins and other physical probing techniques (temperature, power tests, etc.)
- Smart Cards
  - Small chips with some physical protection
  - Freezing the chip and getting the info out
  - Physical probing onto the chip bus
  - Surface meshes (can be defeated with Focused Ion Beams)

Physical Attacks on Smart Cards

- Stopping control signals aimed at the card
- Freezing the card and reading secret data in memory
- Slow-clocking the card and getting data out
- Probing though the passivation layers
- Probing and reading data on bus lines
- Changing the program flow (by external probing)
- Mapping the chip's hardware

External Physical Attacks

- Fault Analysis
  - Causing voltage spikes on a computing device may change values stored in it or used in computation
- Timing Analysis
  - Measuring the time it takes to perform certain computations may reveal something about secret data stored internally
- Power Analysis
  - Measuring the power consumption of a device while it is computing something may reveal something about the values being used in the computations

Risk Analysis – Questions

- What do you want to protect?
- Why do you want to protect it?
- What is its value?
- What are the threats?
- What are the risks?
- What are the consequences of its loss?
- What are the various scenarios?
- What will the loss of the information or system cost?

Risk Analysis – Methodology

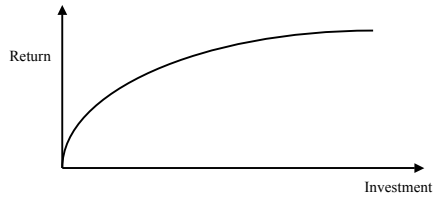
- Assessment of assets (hardware, software, data, time)
- Identification of potential threats and problems
- Identification of damage "likelihood"
- Evaluation of damage "expectation"
- Survey of potential security solutions
- Evaluation of reduction in damage "expectation"
- Decision regarding "value" of security system

Risk Analysis – Example

- Assets:
  - Home appliances, money, personal items, etc.
- Threats:
  - Damage, fire, flood, theft
- Solutions:
  - Prevention (doors, locks, guard, alarm)
  - Detection (alarm, camera)
  - Recovery (insurance, legal means)
- Costs:
  - Should be reasonable compared to damage

Security Management

- Security does not contribute to the business
- Security only reduces losses
- Return-On-Investment curves



Classical Cryptography I

Prof. Shlomo Kipnis
October 29, 2007

Cryptography

Crypt = secret
Graph = writing

Cryptography is the science / art of transforming meaningful information into unintelligible text

Cryptanalysis is the science / art of breaking cryptographic codes

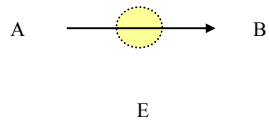
Cryptology is the science / art / study of both cryptography and cryptanalysis

Applications of Cryptography

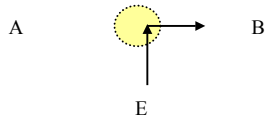
- Assuring text integrity
- Authenticating parties
- Authenticating text source
- Assuring text confidentiality
- Document fingerprinting
- Document signature
- Non-repudiation
- Secure transactions
- Creating random values
- Exchanging keys
- Sharing Secrets
- Hiding traffic existence
- Digital cash
- Preserving anonymity
- Copyright protection
- More . . .

Cryptographic Services (I)

- Source Integrity

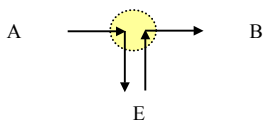


- Source Authenticity

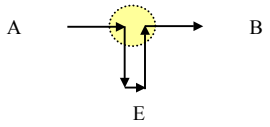


Cryptographic Services (II)

- Data Integrity

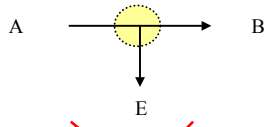


- Data Timeliness

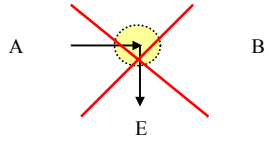


Cryptographic Services (III)

- Data Confidentiality



- Denial of Service



Steganography

Stega = hidden
Graph = writing

Steganography is the science / art of hiding information inside objects

Examples:
- Hidden message in a text file
- Hidden copyright mark in an image file
- Hidden communication between source and destination

Applications of Steganography

- Copyrighted material monitoring
- Data transmission auditing
- Data augmentation
- Tamper proofing
- Commercial espionage
- Information surveillance

Sample Steganographic Scehmes

- Sending a digital object, in which certain fields have hidden meaning between the sender and receiver
- Changing some of the low-order bits of a digital image to imprint a watermark
- Transferring information from high-security zones to low-security zones with covert channels (e.g., time, name, existence)
- Adding encrypted information as noise in a legitimate analog-to-digital object (such as image, audio, video)

Cryptography and Steganography

- Cryptography
  - Tries to conceal the contents of a communication between parties – but it does not try to conceal the existence of the communication
- Steganography
  - Tries to conceal the very existence of the communication between the parties

Early Cryptography

- Cryptography used by early civilizations (such as Egyptians, Jews, Greeks, Romans)
- Early use of cryptography consisted of encryption by substitution methods and/or transposition methods
- Early cryptography was rather simple because of the lack of sophisticated computing engines
- Early substitution methods and transposition methods are easily attacked
- Same methods are in use today, but with stronger properties and more powerful computing engines

Substitution Methods

- Methods in which the letters of the alphabet are replaced with other letters / numbers / symbols
- Examples:
  - Biblical Cipher – fixed permutation
  - Caesar Cipher – fixed permutation
  - Mono-Alphabetic Ciphers – one of many permutations
  - Playfair Cipher – keyed-table lookup
  - Hill Cipher – matrix-multiplication operation
  - Poly-Alphabetic Ciphers – changing permutations
  - Vigenere Cipher – multiple Mono-Alphabetic Ciphers
- Algorithm is known – Key is "index" of permutation

Early Biblical Cipher

- Jeremiah 51, 41:

  "איך נלכדה ששך ותתפש תהלת כל הארץ . . ."

- The word – ששך – actually refers to – בבל – if the following Hebrew substitution method is used:

  א ↔ ת
  ב ↔ ש
  . . . . .
  כ ↔ ל

- Note: בבל is Babylon

Caesar Cipher

- Cyclic shift of the 26 letters of the alphabet by 3:

  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 E F G H I J K L M N O P Q R S T U V W X Y Z A B C

- In mathematical terms:

$$C = ENC(P) = P + 3 \pmod{26}$$

- The secrecy is in the algorithm (!!!!)
- There is one key (fixed permutation)
- Easy to break (if algorithm is known)

Generalization of Caesar Cipher

- Cyclic shift of the 26 letters of the alphabet by key K, where $0 \le K < 26$.
- In mathematical terms:

$$C = ENC_K(P) = P + K \pmod{26}$$

- Algorithm is known
- There are 26 different keys
- Easy to break – check which of 26 possible keys returns the unintelligible ciphertext to a meaningful plaintext

Mono-Alphabetic Ciphers

- One of N! permutations on N letters of the alphabet
- The key is the index of the permutation
- Algorithm is known (mono-alphabetic cipher)
- Key is secret (one of N! options)
- Example:
  - N = 26 letters of the English alphabet
  - $N! = 26! \approx 4 \cdot 10^{26} \approx 2^{88}$ permutations

Attacking Mono-Alphabetic Ciphers (I)

- Appearance frequency of letters (in long enough texts) in the language is well determined
- In the English language:

A ≈ 8.2%	H ≈ 6.1%	O ≈ 7.5%	V ≈ 1.0%
B ≈ 1.5%	I ≈ 7.0%	P ≈ 1.9%	W ≈ 2.4%
C ≈ 2.8%	J ≈ 0.2%	Q ≈ 0.1%	X ≈ 0.2%
D ≈ 4.3%	K ≈ 0.8%	R ≈ 6.0%	Y ≈ 2.0%
E ≈ 12.7%	L ≈ 4.0%	S ≈ 6.3%	Z ≈ 0.1%
F ≈ 2.2%	M ≈ 2.4%	T ≈ 9.1%	
G ≈ 2.0%	N ≈ 6.7%	U ≈ 2.8%	

Attacking Mono-Alphabetic Ciphers (II)

- Appearance frequency of pairs of letters in the language is well defined:

  th, ee, oo, tt, qu, is, ae, . . .

- Appearance frequency of certain words in the language is well defined:

the ≈ 6.4%	a ≈ 2.1%	i ≈ 0.9%
of ≈ 4.0%	in ≈ 1.8%	it ≈ 0.9%
and ≈ 3.2%	that ≈ 1.2%	for ≈ 0.8%
to ≈ 2.4%	is ≈ 1.0%	as ≈ 0.8%

Attacking Mono-Alphabetic Ciphers (III)

- Using the appearance frequencies of letters, words, and pairs-of-letters – accelerates the identification of certain letter substitutions (which are part of the key)
- Identification of word patterns, vowels, and consonants helps in finding parts of the text
- The identification of the remaining parts of the key now reduces the search space dramatically (from N!)
- Using heuristics and associative word-completions, the rest of the key can be easily revealed

Playfair Cipher (I)

- A multiple-letter encryption method – encrypts pairs of letters at each step
- Use a word in the language as the key, and build a 5 X 5 table of the letters in the key and the other letters
- Example: Key = "MONARCHY"

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Playfair Cipher (II)

- Rules for substituting a pair of plaintext letters:
  - Repeated letters in the plaintext are separated with a filler letter (such as Z)
  - Two plaintext letters that fall in the same row of the 5 X 5 table, are replaced by two letters that reside to their right in the table (with circular wraparound)
  - Two plaintext letters that fall in the same column of the 5 X 5 table, are replaced by two letters that reside beneath them in the table (with circular wraparound)
  - Otherwise, each plaintext letter (of a pair) is replaced by the letter that lies in its own row and the column occupied by the other plaintext letter (of the pair)

Playfair Cipher (III)

- Example:
  - The word "funny" is transformed into the word: "funzny"
  - The pair of letters "fu" is replaced by the pair: "ev"
  - The pair of letters "nz" is replaced by the pair: "rw"
  - The pair of letters "ny" is replaced by the pair: "yg"
  - The ciphertext is therefore – "evrwyg"

Attacking Playfair Cipher

- Playfair cipher works on 26 X 26 = 676 digrams (pairs of letters) rather than on 26 letters
- The graph of letter appearance frequencies of Playfair cipher is flatter than the graph of a Mono-Alphabetic cipher
- Playfair cipher makes attacks based on appearance frequencies of letters more complicated – but it is still subject to attacks based on appearance frequencies of pairs of letters

Hill Cipher (I)

- A multiple-letter encryption method – encrypts m letters of plaintext at each step
- The encryption key K is a m X m matrix of coefficients
- To encrypt – multiply the matrix K by a vector of m plaintext letters to receive a vector of m ciphertext letters. (Arithmetic is modulo the size of the alphabet.)
- Example: m=3

$$C_1 = K_{11} P_1 + K_{12} P_2 + K_{13} P_3$$
$$C_2 = K_{21} P_1 + K_{22} P_2 + K_{23} P_3$$
$$C_3 = K_{31} P_1 + K_{32} P_2 + K_{33} P_3$$

Hill Cipher (II)

- The encryption key K is a $m \times m$ matrix of coefficients
- The decryption key K^{-1} is the $m \times m$ matrix of coefficients that is the inverse of matrix K:

$$K K^{-1} = I$$

- To decrypt – multiply the matrix K^{-1} by the vector of m ciphertext letters to receive the vector of m plaintext letters. (Arithmetic is modulo the size of the alphabet.)

Attacking Hill Cipher

- Hill Cipher is rather strong (for large values of m) against "Known Ciphertext" attacks
- Problem with Hill Cipher arises with "Known Plaintext" attacks:
  - If there are m pairs of < plaintext , ciphertext > vectors of length m, then one can write a system of m^2 equations with the unknowns being the m^2 coefficients of the $m \times m$ matrix K.
  - If some of the pairs of < plaintext , ciphertext > vectors of length m do not contribute new independent equations – then one needs to obtain more pairs of < plaintext , ciphertext > vectors of length m, until the system of m^2 equations becomes regular

Poly-Alphabetic Ciphers

- Use different Mono-Alphabetic Ciphers at different parts of the plaintext
- Using many Mono-Alphabetic Ciphers will more-or-less equate the appearance frequencies of letters
- Well-designed, and sufficiently long, Poly-Alphabetic Ciphers can be quite strong
- A common scheme to build a Poly-Alphabetic Cipher:
  - Use a collection of related Mono-Alphabetic Ciphers
  - Use a key to determine which one of the Mono-Alphabetic Ciphers in the collection to use at each stage

Vigenere Cipher (I)

- The collection of Mono-Alphabetic Ciphers consists of the 26 options for Caesar Cipher (with $K = 0, 1, 2, \ldots, 25$)
- Each of the 26 Caesar Ciphers is denoted by a letter, which is the ciphertext letter that replaces the letter 'a'
- In practice:
  - A table of 26 rows by 26 columns is built. Row i in the table contains the 26 letters of the alphabet circularly shifted by i.
  - A keyword is used (over and over again) to select which of the mono-alphabetic ciphers to use. The cipher used is selected by the current letter in the keyword.

Vigenere Cipher (II)

- Example: Table with 6 letters (A, B, C, D, E, F):

	a	b	c	d	e	f
A	A	B	C	D	E	F
B	B	C	D	E	F	A
C	C	D	E	F	A	B
D	D	E	F	A	B	C
E	E	F	A	B	C	D
F	F	A	B	C	D	E

- Keyword = "BAD"
- Plaintext = "cabfaded"
- Ciphertext = "DAEAAAFD"

Vigenere Cipher (III)

- The strength of Vigenere Cipher is based on the fact that there are multiple ciphertext letters to which each plaintext letter can be mapped
- Question: To how many possible ciphertext letters can a single plaintext letter be mapped?
- Answer: The number of different letters of which the keyword is constructed
- Comment: Typical keywords are not too long

Attacking Vigenere Cipher

- Check whether the cipher is Mono-Alphabetic
  - Check whether the appearance frequency of letters in the ciphertext complies with that of a Mono-Alphabetic cipher
- Determine the length of the keyword
  - If two identical sequences of plaintext letters occur at a distance that is an integer multiple of the keyword length – than the two corresponding sequences of ciphertext letters will be identical
  - Detect identical sequences of ciphertext letters
  - Conjecture that the keyword length is the GCD (greatest common divisor) of distances between identical sequences of ciphertext
- Neutralize shifts and break each of the suspected Mono-Alphabetic Ciphers independently

Some Cryptanalytic Attacks

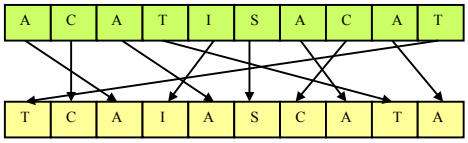
- Known Cipehrtext
  - Only the ciphertext is known to attacker
  - Cryptanalysis aims at revealing the plaintext and/or the key
- Known Plaintext
  - Pairs of < plaintext , ciphertext > are known to attacker
  - Cryptanalysis aims at revealing the key
  - Relevant when plaintext is known / can be obtained
- Chosen Plaintext
  - Attacker chooses the plaintext and receives the ciphertext
  - Cryptanalysis aims at revealing the key
  - Relevant when attacker can "inject" plaintext messages

Classical Cryptography II

Prof. Shlomo Kipnis
October 31, 2007

Transposition Methods

- Plaintext is buffered in a buffer of size N characters
- Plaintext is scrambled by a fixed transposition on the locations of the characters in the buffer



- Algorithm is known
- Key is the indication of the transposition mapping

Early Spartan Cipher

- The Spartan scytale is a wooden staff around which a strip of leather is wound. The text is written along the length of the scytale, and then the strip is unwounded. The result is a long meaningless string.
- Example: Try to decipher the following text –

```
TETHIETLTHRRNOHCIYSRHEGOTIEIERSPOIOTOUASRT
SETRNFUITAPOT
```

- Answer: Reading every 14-th letter gives the text –

```
The secret is thy prisoner; if thou let it
go, thou art a prisoner to it
```


Rail-Fence Cipher

- Method:

  Plaintext is written down as a sequence of diagonals and then read of as a sequence of rows
- Example:

```
m   e   m   a   t   r   h   t   g   p   r   y
  e   t   e   f   e   t   e   o   a   a   t
```

- Plaintext:

```
meet me after the toga party
```


Row-Column Cipher

- Plaintext is written in a rectangle, row by row, and ciphertext is read from the rectangle, column by column, in a permuted column order
- Example:

```
Key:         2   4   1   5   3

Plaintext:   a   t   t   a   c
             k   f   r   o   m
             e   a   s   t   a
             t   d   a   w   n

Ciphertext: trsaaketcmantfadaotw
```


Repeated Row-Column Cipher

- Repeat the Row-Column Transposition several times

```
Key:              2   4   1   5   3
Plaintext-1:      attack from east at dawn
Table-1:          a   t   t   a   c
                  k   f   r   o   m
                  e   a   s   t   a
                  t   d   a   w   n
Ciphertext-1:     trsa aket cman tfad aotw
Plaintext-2:      trsaa ketcm antfa daotw
Table-2:          t   r   s   a   a
                  k   e   t   c   m
                  a   n   t   f   a
                  d   a   o   t   w
Ciphertext-2:     stto tkad amaw rena acft
```

Attacking Transposition Methods

- A pure Transposition Cipher can be easily recognized, because it has the same letter appearance frequencies as the original text
- Appearance frequencies of di-grams or tri-grams may also be useful in breaking the code
- Frequent plaintext words (or combinations of letters) may repeat at same locations in many buffers, which will result in repetition of certain letter combinations in the ciphertext
- Key can be determined by placing the ciphertext in a rectangle and playing with the rows and the columns

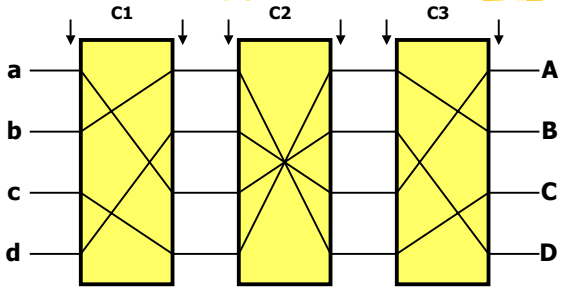
Rotor Machines (I)

- Rotor Machines combine principles of Substitution Methods and Transposition Methods
- Rotor Machines produce ciphers that are very difficult to break
- Rotor Machines in World War II:
  - "Enigma" used by the German
  - "Purple" used by the Japanese
- The breaking of both Rotor Machines by the Allies was a significant factor in the outcome of the war

Rotor Machines (II)

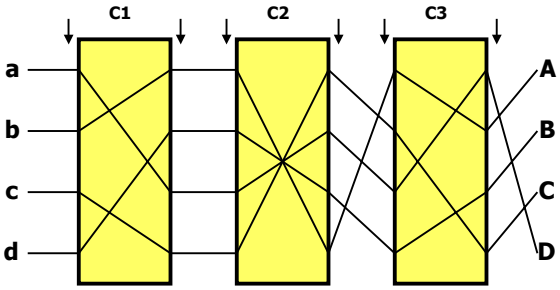
- Rotor Machine is a set of L independently rotating cylinders, each having 26 input pins and 26 output pins, with internal wiring that connect each input pin to a unique output pin
- Each cylinder implements a fixed permutation between its 26 input pins and its 26 output pins
- The 26 rotations of a cylinder implement 26 permutations
- The set of L cylinders implements 26^L permutations
- The key of the Rotor Machine consists of the relative arrangement of the L cylinders

Rotor Machine Example (I)



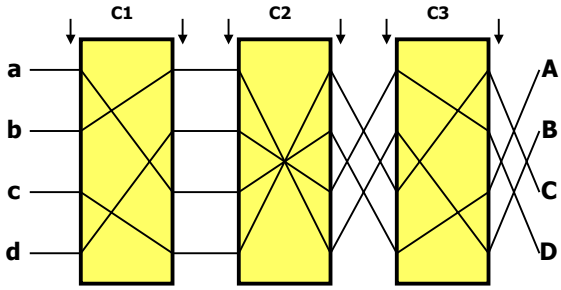
This setting maps: a→D, b→C, c→B, d→A

Rotor Machine Example (II)



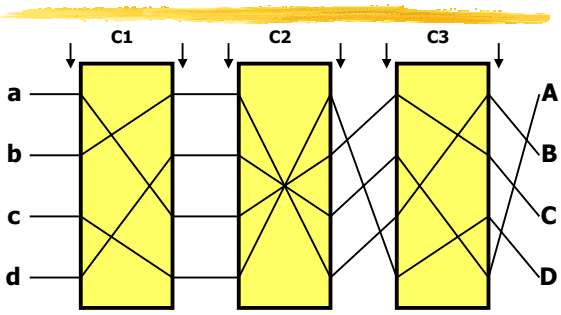
This setting maps: a→D, b→A, c→C, d→B

Rotor Machine Example (III)



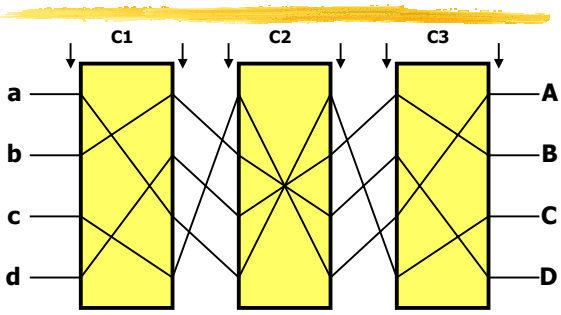
This setting maps: a→A, b→B, c→C, d→D

Rotor Machine Example (IV)



This setting maps: a→C, b→B, c→D, d→A

Rotor Machine Example (V)



This setting maps: a→C, b→D, c→A, d→B

Enigma Machine (I)

- Commercial Version of Enigma Machine:
  - Uses 3 cylinders, each of which has 26 input pins and 26 output pins, totaling $26^3 = 17{,}576$ permutations
  - Has a reflector at the end
  - A plaintext letter typed at the keyboard goes through 3 cylinders, through the reflector, back through the 3 cylinders, and lights up a lamp of the ciphertext letter
  - Every encryption (of a single letter) steps the cylinders to the next setting of the cylinders
  - Decryption is mirror image of Encryption (due to reflector). If letter X is mapped to letter Y, than letter Y is mapped to letter X

Enigma Machine (II)

- Commercial Version of Enigma Machine (continued):
  - Since number pf permutations (17,576) was not considered high enough – the design of the Enigma machines allowed swapping the 3 cylinders. This multiplied the number of initial settings by $3! = 6$
  - In addition, there were 6 cables that enabled swapping pairs of letters at the keyboard. This multiplied the number of initial options by 100,391,791,500
  - The key is the initial setting of the machine
  - Total number of initial settings of the Enigma Machine is about $10^{16} = 10{,}000{,}000{,}000{,}000{,}000$

Enigma Machine (III)

- Commercial Version of Enigma Machine (continued):
  - In the early 1930's, the German used the commercial version of the Enigma Machine to encrypt military content
  - Germans used to transmit the "daily key" setting encrypted with the key of the previous day. The "daily key" consisted of a set of 3 letters repeated **twice** (for a total of 6 letters)
  - The Polish managed to get a hold of an Enigma machine. Polish mathematicians tabulated patterns of encryptions of the Enigma, and they searched for desired patterns.
  - Commercial version on Enigma was analyzed and broken by Polish mathematicians in mid 1930's

Enigma Machine (IV)

- German Military Version of Enigma Machine:
  - In 1939, the Germans increased the Enigma security
  - Use of 3 out of 5 cylinders increased number of options from 6 to 60
  - Use of 10 instead of 6 plug-board cables increased number of options by another factor of about 1600
  - Total number of initial settings of the German Military Enigma Machine is about – $1.59 \cdot 10^{20} = 159{,}000{,}000{,}000{,}000{,}000{,}000$
  - Polish cryptanalysts weren't able to crack the Enigma anymore
  - Action now moves to Britain

Enigma Machine (V)

- German Military Version of Enigma Machine (continued):
  - British focused on finding "cribs" – pieces of plaintext that can be associated with pieces of ciphertext.
  - Within "cribs" – they tried to find chains of letters that encrypt to one another. Such chains could help in automatic searches. Automatic searches were handled by electrical wiring of chains of Enigma Machines.
  - Problem of plug-board setting was isolated from that of the cylinder arrangements.
  - To try all possible "3 out of 5" cylinder arrangements – 60 machines were built to run in parallel.
  - In the early 1940's, the British were able to crack the Enigma

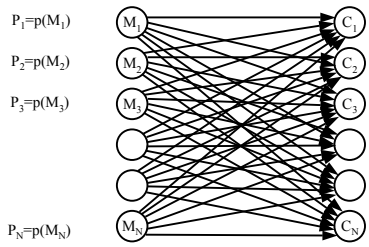
Classical Cryptography – Summary

- Use of Substitution Ciphers (permutations on the letters) and/or Transposition Ciphers (permutations on the places)
- Require increasingly more sophisticated computing machinery to implement (encrypt and decrypt)
- Assume the attacker doesn't have enough text, computing machinery, and time to break the ciphers
- Question: Is there any hope of coming with a cryptographic scheme that cannot be broken?

Shannon Theory of Secrecy Systems

- Let $\{M_1, M_2, \ldots, M_N\}$ be the message space
- Messages $M_1, M_2, \ldots, M_N$ are distributed with probabilities $p(M_1), p(M_2), \ldots p(M_N)$ (not necessarily uniform)
- Let $\{K_1, K_2, \ldots, K_L\}$ be the key space
- Keys $K_1, K_2, \ldots, K_L$ are distributed with probabilities $p(K_1), p(K_2), \ldots p(K_L)$ (usually uniform – $p(K_i) = 1/L$)
- Each key defines a projection of all the messages onto all the cipher texts, giving a bipartite graph

Shannon's Message-Ciphertext Graph



Perfect Ciphers (I)

- A cipher is perfect if for any i and j we have:

$$p(M_i|C_j) = p(M_i)$$

- This means that the ciphertext does not reveal any information about the plaintext
- By this definition – a perfect cipher is 100% resilient against Known Ciphertext Attacks, even if the opponent has infinite computational power

Perfect Ciphers (II)

- Note: $p(M)\, p(C|M) = p(M,C) = p(C)\, p(M|C)$
- Theorem: A cipher is perfect if and only if

$$\forall M,C: \quad p(C) = p(C|M)$$

- Note:

$$p(C|M) = \sum p(K)$$

(where sum is over all K such that $E_K(M)=C$)

- Therefore: A cipher is perfect if and only if

$\forall C$: $\sum p(K)$ over all K such that $E_K(M)=C$ is independent of M

Perfect Ciphers (III)

- L – number of keys; N – number of plaintext messages
- Theorem: A perfect cipher satisfies $L \geq N$ (proof omitted)
- Example: Encrypting only one letter with Caesar Cipher:
  - $L = 26$ and $N = 26$
  - $p(C) = p(C|M) = 1/26$
- Example: Encrypting two letters with Caesar Cipher:
  - $L = 26$ and $N = 26^2$
  - $P(C) = (1/26)^2$
  - But – $p(C|M)$ is either $(1/26)$ or 0

Perfect Ciphers (IV)

- Question: How can one make the Caesar Cipher perfect for more than one letter?
- Answer: By using a new random key for encrypting each letter of the plaintext
- Example:

  Message = HELLOCAESAR

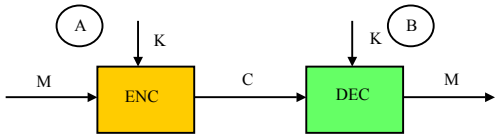
  Key = `gxaudovjrak`

  Ciphertext = `nblfrqvnjab`

Perfect Encryption and Authentication

- These are cryptographic schemes that cannot be broken – no matter how much text, computation power, and time the opponent has
- Encryption with "One-Time PAD" and Authentication with "One-Time MAC" require that:
  - key be long (same size as message)
  - key be random
  - key be one-time
- Perfect Cryptography is provably 100% secure

Symmetric Message Encryption



- A encrypts the message M using the algorithm ENC with the key K to receive the cipher C
- B decrypts the cipher C using the algorithm DEC with the key K to receive the message M
- Key K is symmetric, secret, and known only to A and B

OTP – One-Time Pad

- Message M of length n bits
- Key K of length n bits (same as M)
- Key K is random bit by bit
- Key K is used only once
- Key K is known only to A and B

- Encryption by A: $C = M \oplus K$ (bit-wise XOR)
- Decryption by B: $M = C \oplus K$ (bit-wise XOR)

OTP – Proof of Perfection

- Because of the randomness of the bits of K – the opponent's knowledge about the message M is identical whether or not the cipher C has been seen
- Perfect Security Claim:

  $$Prob(M = x \mid C = y) = Prob(M = x)$$

- The opponent can only guess the message M, of length n bits, with success probability of $(1/2)^n$

OTP – Proof of Perfection (explained)

- Question: Why can't the opponent start from the ciphertext C and reveal something about plaintext M?
- Answer: What the opponent can do is:
  - Start from the ciphertext C
  - Guess one of the keys K with probability $(1/2)^n$
  - Obtain a viable plaintext M with probability $(1/2)^n$
  - But this is not different than just guessing M with probability $(1/2)^n$

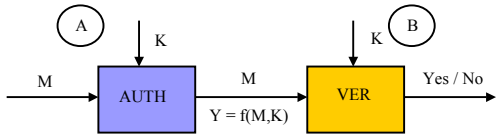
OTP – Pad Reuse Problems

- If two messages are encrypted with the same key:
  $C_1 = M_1 \oplus K$
  $C_2 = M_2 \oplus K$
- Then – the following relation holds:
  $C_1 \oplus C_2 = M_1 \oplus M_2$
- This means that the messages M_1 and M_2 can be correlated, and security is not 100% anymore

OTP – Practicality Issues

- Generating large random pads
  - Random physical phenomena
  - Generating random numbers in software
- Distributing large random pads:
  - Physical delivery
  - Delivery over secure media
  - Quantum key-distribution

Symmetric Message Authentication



- A computes a Message Authentication Code (MAC) for the message M using the algorithm AUTH with the key K to receive the code Y
- B uses the algorithm VER to verify that the code Y is indeed the MAC of the message M with the key K
- Key K is symmetric, secret, and known only to A and B

OTM – One-Time MAC

- Large prime P
- Message M with numeric value between 1 and P-1
- Key K is a pair of random numbers, a and b, such that:
  $0 < a, b < P$
- Key K is used only once
- Message Authentication Code (MAC) is:
  $$Y = f(M, K) = a \bullet M + b \pmod P$$
- MAC is computed by the sending party (who knows K)
- MAC is verified by the receiving party (who knows K)

OTM – Attacking the MAC

- What can an opponent do?
- Find the key K = (a,b)
  modify the message M to M'
  modify the MAC Y to Y'
  in a way that will not be detected by B
- Even without knowing K:
  come up with another message M'
  come up with another code Y'
  such that M' and Y' will be accepted by B

OTM – Key Reuse Problems

- If two different messages M_1 and M_2 are authenticated with the same key $K = (a, b)$ – then:
  $Y_1 = a \bullet M_1 + b \pmod P$
  $Y_2 = a \bullet M_2 + b \pmod P$
- Two equations with two unknowns – a and b
- This system can be solved, which means that the key $K = (a, b)$ can be found

OTM – Proof of Perfection (I)

- Why the opponent cannot reveal the key?
- The opponent:
  - knows the value of P
  - sees two values – M and Y
  - knows that $Y = a \bullet M + b \pmod P$
- For every "guess" of the value of 'a' – there is exactly one value of 'b' that solves the equation; and also vice versa
- Perfect Security Claim:
  $\text{Prob}(\text{ opponent finds } K\ ) = 1/(P-1)$

OTM – Proof of Perfection (II)

- Why the opponent cannot forge a message and a MAC?
- The opponent:
  - knows the value of P
  - sees two values – M and Y
  - knows that $Y = a \bullet M + b \pmod P$
- If the opponent can come up with another message M' and another code Y' such that $Y' = a \bullet M' + b \pmod P$
  - Then, the opponent has two equations with two unknowns (a and b), and the opponent can reveal the key $K = (a, b)$

Using Perfect Cryptography

- Perfect cryptography is 100% secure (with mathematical proofs to back this claim)
- Usage of perfect cryptography is limited due to the need to negotiate a new, long, and random one-time key for every encryption or authentication
- OTP and OTM are the ultimate cryptographic schemes, and they provide a reference point for security
- OTP and OTM are used in highly-critical environments, where key negotiation / distribution can be handled over off-line channels

Modern Cryptography

Prof. Shlomo Kipnis
November 5, 2007

Unconditional Cryptography

- Cryptographic schemes that are provably 100% secure
- An opponent cannot break the scheme:
  - even if unlimited text is available
  - even if unlimited time is available
  - even if unlimited computing power is available
- Strength of scheme relies on a mathematical proof that the opponent does not have enough information to break the cryptographic scheme
- There are unconditionally secure cryptographic schemes. Unfortunately, they are not too practical.

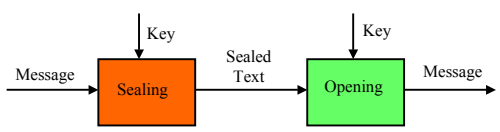
Conditional Cryptography

- Cryptographic schemes, for which we do not have a mathematical proof that they are 100% secure
- An opponent can break the scheme:
  - if unlimited text is available
  - if unlimited time is available
  - if unlimited computing power is available
- Strength of such a scheme relies on the assumption that the opponent does not have sufficient resources (text, time, computers, money, etc.) to break the scheme
- All modern practical cryptographic schemes are only conditionally secure

Notations

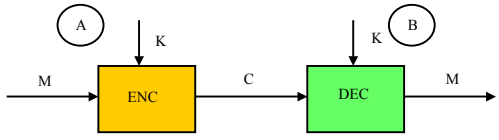
- A – Alice (one party)
- B – Bob (another party)
- E – Eve (eavesdropper / attacker)
- M – Message (plaintext, sometimes secret)
- K – Key (highly secret)
- C – Cipher (ciphertext, seen by attacker)
- ENC – Encryption (secret / known)
- DEC – Decryption (secret / known)
- AUTH – Authentication (secret / known)
- VER – Verification (secret / known)

Symmetric Cryptography



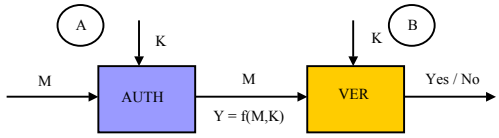
- Same key is used on both sides
- Algorithms are usually similar on both sides

Symmetric Encryption



- A encrypts the message M using the algorithm ENC with the key K to obtain the cipher C
- B decrypts the cipher C using the algorithm DEC with the key K to obtain the message M
- Key K is symmetric, secret, and known only to A and B

Symmetric Authentication

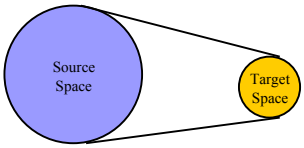


- A computes a Message Authentication Code (MAC) for the message M using the algorithm AUTH with the key K to obtain the code Y
- B uses the algorithm VER to verify that the code Y is indeed the MAC of the message M with the key K
- Key K is symmetric, secret, and known only to A and B

Hash Functions

- Names:
  - Cryptographic Checksum
  - Message Digest
  - Fingerprint
  - Hash Value
- Hash function generates a small identifier of a large object (message, file, document, etc.)
- Identifier is used to identify / authenticate / sign / verify the original object
- It should be difficult to forge the identifier

Properties of Hash Functions

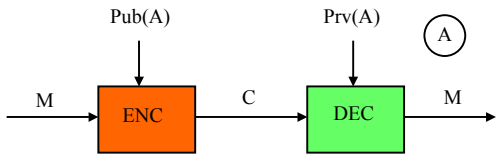


- Should be difficult to reverse a hash function
- Should be difficult to find a collision for some object
- Should be difficult to find a collision of the hash function

Public-Key Cryptography

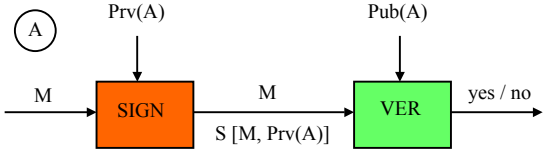
- Entity A has two keys
  - Prv(A) – Private Key of A, kept secretly only by A
  - Pub(A) – Public Key of A, made public to the world
- Two functions used for encryption/decryption:
  - ENC – uses the public key
  - DEC – uses the private key
  - Such that DEC [ENC (M, Pub(A)) , Prv(A)] = M
- Two functions used for signature/verification:
  - SIGN – uses the private key
  - VER – uses the public key
  - Such that VER [SIGN (M, Prv(A)) , Pub(A)] = yes / no

Public-Key Encryption



- Anyone knowing Pub(A) can encrypt message M for A
- Only A can decrypt, using Prv(A), message M that was encrypted specifically for A

Public-Key Signatures



- Only A can sign, using Prv(A), some message M
- Anyone knowing Pub(A) can verify that message M is signed properly by A

Selecting Algorithms

- Published Algorithm – reviewed by many cryptographers and is probably strong
- Manufacturer Algorithm – trust the reputation of the manufacturer
- Private Consultant – trust the reputation of the expert
- Government Standard – trust the government and its intentions
- Own Algorithm – develop own secret algorithm by trusting your skills

Secret vs. Known Algorithms

- Secret algorithms:
  - used in closed environments (military, monopoly)
  - need to be internally analyzed
  - hard to keep secret
  - get broken eventually
- Known algorithms:
  - have a commercial value
  - easy to keep small and changing secret (key)
  - are publicly reviewed

Kerckhoff's Principle

The security model does not rely on the obscurity of the encryption/decryption algorithm. The algorithm is assumed to be known to the adversary. The security depends on the secrecy of the key.

Algorithms and Keys

- Encryption / Decryption / Authentication / Verification algorithms can be either secret or known
- In either case – keys are used as part of the scheme
- An encryption / decryption algorithm can be viewed as a large collection of un-keyed transformations (all having a similar structure, but are different in the details) from plaintext to ciphertext. The secret key can be viewed as the index that selects a particular transformation from the collection.

Selecting Keys

- The key should be selected from a large set of keys:
  - to decrease the probability of guessing the secret key
  - to increase the time required by an attacker to try all the keys in the set
- Key sizes:
  - 40 bits ($2^{40} \approx 10^{12}$) were used in the 1980's and 1990's in Internet applications
  - 56 bits ($2^{56} \approx 10^{17}$) are used by DES; good in the 1980's; not strong enough today
  - 64 bits ($2^{64} \approx 10^{20}$) are used by some ciphers today
  - 128 bits ($2^{128} \approx 10^{40}$) are considered the smallest number of bits to be used by modern algorithms today

Requirements from Cipher Algorithms

- For the legitimate user:
  - Easy to encrypt / decrypt having the key
- For the attacker:
  - Difficult to encrypt / decrypt without the key
  - Difficult to recover the key
  - Difficult to get any information on the plaintext
  - All the above, even if many encrypted texts are seen
- Above should hold, assuming algorithm is not secret

Encryption and Compression

- Encryption is reversible when the key is known; it is not (it should not be) reversible without the key
- Compression can be either fully reversible or partially reversible; but it does not use a key
- If both encryption and compression are desired – do the compression first and then do the encryption
- Good encryption generates output that looks like a random text (and, therefore, cannot be compressed)
- Compressed text is less material to encrypt and is harder to analyze

Non-Cryptanalytic Attacks (I)

- Compromising the Key
  - Stealing devices and keys
  - Getting old (backup) keys
  - Attacking the environment and leaking the key
- Exhaustive Search
  - Searching all the key space (if feasible)
  - Checking which keys encrypt / decrypt the text

Non-Cryptanalytic Attacks (II)

- In many commercial products, one can identify the security algorithms used and break them without sophisticated computations. Many systems use simple obscurity-based algorithms to hide information
- In some commercial systems, the keys are kept in the clear at some known places (key files, registries, beginning of an encrypted file, etc.). Not much is required to break these systems
- Conclusion: It is not sufficient to use good cryptography. Good system design (with security in mind) is needed.

Cryptanalytic Attacks (I)

- Cryptanalysis is the techniques used to recover the secret information hidden in cryptographic algorithms
- Usually, cryptanalysis attempts to find the secret key
- In some cases, cryptanalysis attempts to find the encrypted text
- Cryptanalysts develop techniques that rely on analyzing long texts to find the secret information
- The job of the cryptanalyst gets more difficult with the introduction of powerful algorithms and faster machines

Cryptanalytic Attacks (II)

- Known Cipehrtext
  - Only the ciphertext is known to attacker
  - Cryptanalysis to reveal the plaintext and/or the key
- Known Plaintext
  - Pairs of < plaintext , ciphertext > are known to attacker
  - Cryptanalysis to reveal the key
  - Relevant when plaintext is known / can be obtained
- Chosen Plaintext
  - Attacker chooses the plaintext and receives the ciphertext
  - Cryptanalysis to reveal the key
  - Relevant when attacker can "inject" plaintext messages

Cryptanalytic Attacks (III)

- Chosen Ciphertext
  - Attacker chooses the ciphertext and receives the plaintext that encrypts to the chosen ciphertext
  - Cryptanalysis to reveal the key
  - Relevant when attacker can "inject" ciphertext messages to the decryption module
- Adaptive Chosen Text
  - Attacker chooses successive plaintext and/or the ciphertext messages in accordance to attack plan
  - Cryptanalysis to reveal the key
  - Relevant when attacker can "control" the encryption and decryption modules to respect chosen messages

Modern Cryptography

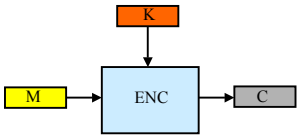
- Development of ciphers with the understanding of the principles of Perfect Ciphers
- Mathematical proofs (when possible) regarding the strength of the ciphers
- Unconditional Cryptography – theoretically 100% secure, but it is not very practical
- Conditional Cryptography – theoretically not 100% secure, but it is very practical

Block Ciphers and Stream Ciphers

- When encrypting large volumes of data – the plaintext M is divided into fixed-size blocks $M = M_1, M_2, \ldots, M_L$
- Each block M_i is encrypted to a ciphertext block C_i and the results are concatenated to produce the ciphertext $C = C_1, C_2, \ldots, C_L$
- There are two main ways to encrypt the plaintext M:
  - Block Ciphers – Plaintext is fragmented into fixed-sized blocks (typically 64 / 128 bits long) and the encryption transformation is applied to each block separately
  - Stream Ciphers – Plaintext is handled in small groups of bits (typically 1 or 8 each time) and the encryption transformation maintains memory (state) between encryption steps

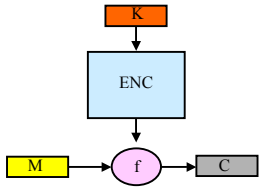
Block Ciphers

- Encryption of one block of data at a time
- Need to partition message into blocks
- Need to pad small blocks
- Key is used many times before it is changed
- ENC should scramble the data and the key well
- With enough text and time – key can be revealed



Stream Ciphers

- Encryption of one bit or one byte at a time
- Message is encrypted as it comes
- No need to pad data
- Key is used many times before it is changed
- ENC function is used primarily to prepare a pad
- With enough text and time – key can be revealed

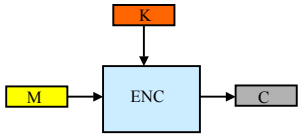


Deterministic Block Ciphers (I)

- Given $M = M_1, M_2, \ldots, M_L$
- Each block is encrypted with the same encryption function $E_K(\bullet)$ and the same key K – $C_i = E_K(M_i)$
- The encryption function $E_K(\bullet)$ should not be vulnerable to Known Plaintext Attacks (that is, having the attacker be able to collect all the < plaintext, ciphertext > block pairs for all the keys, and being able to tell for each ciphertext block which plaintext block and/or key are being used)
- This requirement imposes some constraints on the block size and on the key size

Deterministic Block Ciphers (II)

- $|K|$ must not be too small and must not be too large
- Typical sizes for $|K|$ were 64 / 128 bits
- Moving towards 128 / 192 / 256 bits



- Must have $|C| \geq |M|$
- Typically $|C| = |M|$
- Typical sizes for $|M|$ were 64 / 128 bits
- Moving towards 128 / 192 / 256 bits

Deterministic Block Ciphers (III)

- Block Ciphers (on blocks on length N bits) are actually Substitution Ciphers in which the alphabet is the set of all the binary blocks of length N
- Example:
  - When $N=64$ – there are 2^{64} blocks of plaintext / ciphertext
  - When $N=128$ – there are 2^{128} blocks of plaintext / ciphertext
- For each key K – the function $E_K(\bullet)$ is a permutation from $\{0,1\}^N$ to $\{0,1\}^N$
- The function $D_K(\bullet)$ is the inverse permutation of $E_K(\bullet)$

Deterministic Block Ciphers (IV)

- Some permutations from $\{0,1\}^N$ to $\{0,1\}^N$ are weak
- Let $M = b_1 b_2 \ldots b_N$ (binary representation of M)
- Examples of Weak Block Ciphers:
  - $E_0(b_1 b_2 \ldots b_{N-1} b_N) = b_1 b_2 \ldots b_{N-1} b_N$ (identity permutation)
  - $E_1(b_1 b_2 \ldots b_{N-1} b_N) = b_2 b_3 \ldots b_N b_1$ (cyclic shift of the bits)
  - $E_2(b_1 b_2 \ldots b_{N-1} b_N) = b_2 b_1 \ldots b_N b_{N-1}$ (bit transpositions)
  - $E_3(M) = M+3 \pmod{2^N}$ (large Caesar Cipher)
  - $E_4(M) = M+K \pmod{2^N}$ (generalized Caesar)
- Luckily – there are enough good permutations …

Deterministic Block Ciphers (V)

- For $N=64$ – there are $2^{64}!$ permutations
- Question: How does one specify all (or part of) these $2^{64}!$ permutations (since the permutation is the key)?
- Obvious Answer: It is not practical to write down all the permutations in a "Code Book" and lookup the key…
- Another Answer: One needs to find an efficient way to compute the permutation when the key is given

Deterministic Block Ciphers (VI)

- No simple substitutions / transpositions of the bits of the plaintext block
- Heavy involvement of the key K in the generation of the ciphertext block from the plaintext block
- For key K – the cipher mapping should be non-linear in the plaintext M (since linear mappings can be attacked by algebraic techniques)
- For any two keys K_1 and K_2 – their mappings should be non-linear in K_1 and K_2 (since linear mappings can be attacked by algebraic techniques)

Deterministic Block Ciphers (VII)

- Cipher blocks should look random (pass statistical tests, look random over time, etc.) when messages change
- Cipher blocks should look random (pass statistical tests, look random over time, etc.) when keys change
- Opponent should not be able to detect a cipher block from a random block
- Cipher blocks should change dramatically even when small changes are made to plaintext blocks
- Each bit in the plaintext block should influence all the bits in the cipher block ("avalanche effect")

Deterministic Block Ciphers (VIII)

- Known Plaintext Attack:
  - Choose random messages $M_1, M_2, \ldots, M_L$
  - Opponent receives $M_1, M_2, \ldots, M_L$ and $C_1 = E_K(M_1),\ C_2 = E_K(M_2),\ \ldots,\ C_L = E_K(M_L)$
  - Opponent tries to recover key – that is – to output K' as guess for K
- An encryption algorithm should be secure at least against Known Plaintext Attacks
- What about Chosen Plaintext Attacks ?
- Does security against key recovery from Known Plaintext imply security against key recovery from Chosen Plaintext ?

Deterministic Block Ciphers (IX)

- Assume E_K is secure against key recovery from known plaintext attacks, and define:

$$E'_K(M) = \begin{cases} K & \text{for} \quad M = 0 \\ E_K(0) & \text{if} \quad E_K(M) = K \\ E_K(M) & \text{else} \end{cases}$$

- Then, E'_K is secure against Known Plaintext, but it is not secure against Chosen Plaintext (since randomly selected messages will not be useful to recover the key, but messages chosen purposely will recover the key)

Distinguishability Tests (I)

- Key Recovery is not a good test for security (it is not sufficient to require that E_K is secure against key recovery)
- Example: $E_K(M) = M$ does not recover the key, but it is not a very good encryption function
- Distinguishability Tests capture the essence of good encryption functions:
  - Opponent is given several encryptions of known / chosen plaintexts
  - Opponent is given / chooses two new plaintexts X_1 and X_2 that were not seen before
  - Opponent is given encryption C of one of the new plaintexts
  - Opponent needs to distinguish whether $C = E_K(X_1)$ or $C = E_K(X_2)$

Distinguishability Tests (II)

- Known Plaintext Attack (with non-repeating inputs):
  - Pick random key K (unknown to opponent)
  - Let $i \leftarrow 1$
  - Repeat until opponent decides to abort or until $i = 2^N - 1$
    - Pick new message M_i not in $\{M_1, M_2, \ldots, M_{i-1}\}$
    - Give the opponent M_i and $C_i = E_K(M_i)$
    - $i \leftarrow i + 1$
  - Give the opponent two plaintexts X_1 and X_2 not in $\{M_1, M_2, \ldots, M_{i-1}\}$
  - Pick a random bit b
  - Give the opponent $C = E_K(X_b)$
  - Opponent outputs guess b' for b

Distinguishability Tests (III)

- Chosen Ciphertext Attack (with non-repeating inputs):
  - Pick random key K (unknown to opponent)
  - Let $i \leftarrow 1$
  - Repeat until opponent decides to abort or until $i = 2^N - 1$
    - Opponent selects M_i
    - Opponent is given $C_i = E_K(M_i)$
    - $i \leftarrow i + 1$
  - Opponent picks two plaintexts X_1 and X_2 not in $\{M_1, M_2, \ldots, M_{i-1}\}$
  - Pick a random bit b
  - Give the opponent $C = E_K(X_b)$
  - Opponent outputs guess b' for b

Random Permutations (I)

- Random Deterministic Block Cipher:
  - For any K, select a random permutation $f_K : \{0,1\}^N \rightarrow \{0,1\}^N$.
  - Define $E_K(M) = f_K(M)$.
  - Intuitively, opponent learns nothing by seeing a new ciphertext (or seeing a new pair of < plaintext, ciphertext >), except that it differs from previously seen ciphertexts (or pairs of previously seen < plaintext, ciphertext >).
  - In particular, opponent will not be able to distinguish between encryptions of any two messages that were not seen before (because the permutation f_K is random).

Random Permutations (II)

- Question: Why not use a random permutation $f_K : \{0,1\}^N \rightarrow \{0,1\}^N$ as our encryption function E_K?
- Answer: The number of permutations from $\{0,1\}^N$ to $\{0,1\}^N$ is too large – $2^N!$, and there is no easy way of indexing (keying) these permutations.
- Example: For $N=5$, there more than 2^{116} permutations, and there is no easy way to index (key) them
- Idea: Use a smaller subset of permutations that can be easily indexed (keyed), and for which it would still be difficult to distinguish between unseen inputs

Pseudo-Random Permutations (I)

A collection of efficient (easy to compute) permutations $\{f_K\}$ on $\{0,1\}^N$, such that for any randomly selected K, the opponent cannot efficiently (in polynomial time, in polynomial space, with large probability, etc.) distinguish between the permutation f_K and a random permutation

Pseudo-Random Permutations (II)

- Distinguishability Test for a collection $\{f_K\}$ of pseudo random permutations on $\{0,1\}^N$:
  - Select random b in $\{0,1\}$
  - Select random f in $\{f_K\}$
  - Select random permutation p from $\{0,1\}^N$ to $\{0,1\}^N$
  - Repeat until opponent decides to abort:
    - Opponent selects X in $\{0,1\}^N$ and bit b'
    - If $b' = b$ then opponent gets $f(X)$
      else opponent gets $p(X)$
    - Opponent succeeds if it outputs guess b' such that $b' = b$

Pseudo-Random Permutations (III)

- Using pseudo random permutations:
  - Developing secure deterministic block ciphers: encryption function $E_K(\bullet)$ and decryption function $D_K(\bullet)$.
  - Generating independent random values: $V_i = E_K(i)$ where K is a secret key and i is the index of the independent random value.
    Comment: If K is shared, then values of V_i can be shared
  - Proving security of protocols (based on the assumption that the opponent cannot "guess" certain pseudo-random values)

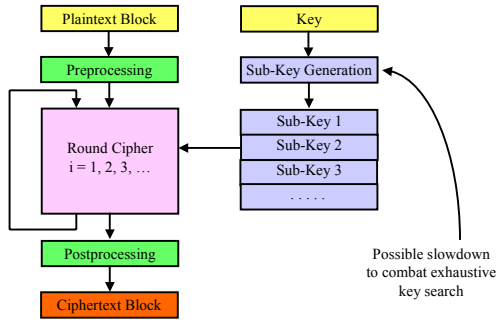
Pseudo-Random Permutations (IV)

- Analyzing a PRP (Pseudo-Random Permutation) should be difficult and time consuming
- To increase security – use several lines of defense
- Cascading PRP:
  - Given PRP f_{K_1} and g_{K_2}, one can compose them to come with a permutation h_{K_1,K_2} such that $h_{K_1,K_2}(X) = f_{K_1}(g_{K_2}(X))$
  - Claim: If f and g are PRP, then h is also a PRP
  - Use: Developing block ciphers that contain several rounds of scrambling the plaintext with PRP

Symmetric Cryptography I

Prof. Shlomo Kipnis
November 7, 2007

Modern Block Ciphers

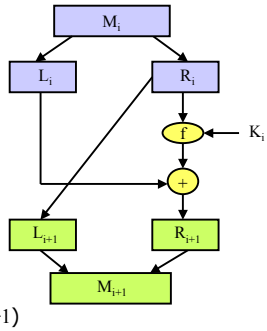


Feistel Ciphers (I)

- Feistel was one of the designers of early cryptographic algorithms at IBM in the 1970's
- Feistel Cipher is a scheme / template for specifying the algorithm of a block cipher
- The Feistel scheme allows encryption and decryption with the same hardware circuit / piece of software
- Algorithms that use the Feistel scheme:
  - DES
  - IDEA
  - RC5

Feistel Ciphers (II)

- There are r rounds
- In round i – input block M_i is broken into two half-blocks L_i and R_i
- Input half-block R_i is copied to output half-block L_{i+1} (to be used as input in round $i+1$)
- Input half-block R_i and round key K_i are scrambled by function f. The result is XORed with input half-block L_i to create output half-block R_{i+1} (to be used as input in round $i+1$)



Feistel Ciphers (III)

- K is used to generate r round keys – $K_1, K_2, \ldots, K_r$
- Input block M_i of round i split into two halves L_i and R_i
- In each round $1 \le i \le r$ do the following:
  - $L_{i+1} \leftarrow R_i$
  - $R_{i+1} \leftarrow L_i \oplus f(R_i, K_i)$
- After last round ($i=r$) – swap the two output blocks:
  - $C \leftarrow R_{i+1} \,\|\, L_{r+1}$

Feistel Ciphers (IV)

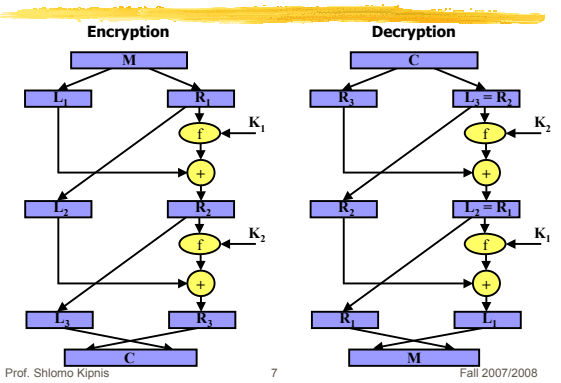
- Examples:
- What happens if $f(R, K) = R$?

  If $M = L \,\|\, R$ – then $C = L \oplus R \,\|\, R$
- What happens if $f(R, K) = K$?

  Multiple encryptions XOR their left-halves L's with the same K (same problem as with one-time-pad)
- What happens if $f(R, K) = R \oplus K$?

  Possible attack if $R = L$ – then K is revealed

Feistel Ciphers (V)



Feistel Ciphers (VI)

- In Feistel Cipher – decryption is the same as encryption, only reversing the order of usage of the r round keys
- Function f can be any function; usually it is a function that is easy to compute but is hard to reverse
- The reversibility of the Feistel Cipher depends on the reversibility of the XOR function
- Function f only serves to generate a pad to be XORed with the L half-block; function f creates this pad from the R half-block and from the round key K

Feistel Ciphers (VII)

- The Feistel scheme does not specify:
  - Block size (N)
  - Key size (L)
  - Number of rounds (r)
  - Round-key generation algorithm ($K_1, K_2, \ldots, K_r$)
  - Scrambling function (f)

DES History

- In 1973, NBS (National Bureau of Standards) came out with an RFP (Request for Proposals) for a commercial encryption standard
- IBM proposed its strong Lucifer algorithm (developed by Feistel and others)
- NSA (National Security Agency) requested to weaken the strength of Lucifer (by shortening the key)
- NSA also made changes to IBM's Lucifer algorithm
- Data Encryption Standard (DES) accepted in 1976

DES Design Criteria

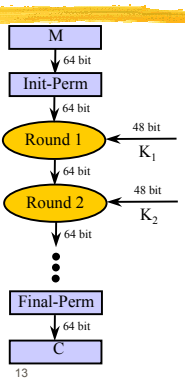
- NBS had set the following design criteria for DES:
  - Algorithm must provide high level of security
  - Algorithm must be completely specified
  - Security of the algorithm must reside in the key
  - Algorithm must be available to all users
  - Algorithm must be adaptable for use in diverse applications
  - Algorithm must be efficiently implemented in hardware
  - Algorithm must be efficient to use
  - Algorithm must be able to be validated
  - Algorithm must be exportable

DES Structure (I)

- Block size – 64 bits
- Key size – 56 bits (in a 64-bit buffer)
- Fixed initial permutation on input block (64 bits)
- 16 round keys (48 bits) derived from key (56 bits)
- Key scheduling scheme for 16 round keys
- 16 iterations each consisting of scrambling the round-block (64 bits) with the round-key (48 bits)
- Scrambling function detailed later
- Fixed inverse initial permutation on output block

DES Structure (II)

Overall structure of DES is as follows



DES Structure (III)

Initial Permutation (64 inputs / 64 outputs):

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

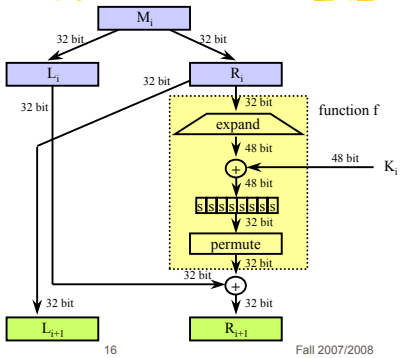
DES Structure (IV)

Initial and Final Permutations:

- "Final Permutation" is inverse of "Initial Permutation"
- "Initial Permutation" and "Final Permutation" are fully specified and do not add to the security of DES
- Purpose of "Initial Permutation" and "Final Permutation" is to make software implementations of DES slow
- Question: Why not use only "Initial Permutation"?
- Answer: To support encryption/decryption with the Feistel Scheme !

DES Structure (V)

Each round of DES consists of the following operations



DES Structure (VI)

- Message block M_i (64 bits) is split into left half-block L_i (32 bits) and right half block R_i (32 bits)
- Right half block R_i is copied to become left half-block L_{i+1}
- Right half block R_i is expanded to 48 bits and is XORed with round key K_i (48 bits)
- The eight S-Boxes – each takes 6 bits (of the 48 bits above) and generates 4 bits (resulting in 32 bits)
- The resulting 32 bits are permuted and XORed with the left half-block L_i to create right half-block R_{i+1}

DES Structure (VII)

Expand Function (32 inputs / 48 outputs):

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

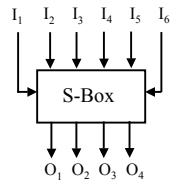
DES Structure (VIII)

Internal Permute (32 inputs / 32 outputs):

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

DES Structure (IX)

- S-Box-1 (6 inputs / 4 outputs):
  - First and sixth input bits select row (between 0 and 3 in table below)
  - Four middle input bits select column (between 0 and 15 in table below)
  - Value of four output bits is depicted in decimal (between 0 and 15) in table entry below



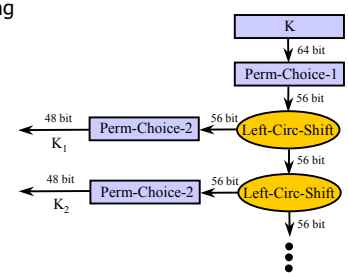
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

DES Structure (X)

- All the operations in DES are linear, with the exception of the S-Boxes
- If the S-Boxes were selected to be easily described by linear transformations – DES would have been weaker
- The strength of DES depends heavily on the selection of the S-Boxes
- NSA had changed the S-Boxes that IBM had proposed
- Other S-Boxes are possible
- Random selection of S-Boxes proves easier to break

DES Structure (XI)

Key-Scheduling



DES Structure (XII)

- DES Key Scheduling – Permuted Choice 1:

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

DES Structure (XIII)

- DES Key Scheduling – Permuted Choice 2:

14	17	11	24	1	5	3	28
15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56
34	53	46	42	50	36	29	32

DES Structure (XIV)

- DES Key Scheduling – Shift Schedule:

Round:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Shift:	1	1	2	2	2	2	2	2	1	2	2	2	2	2	1

Design Issues for Block Ciphers (I)

S-Boxes:

- Output bits of S-Box should not be a linear function of input bits
- Each row of the S-Box should be a permutation of the possible input / output values
- Inputs that differ in one bit should generate outputs that differ in many bits
- Output bits of S-Box should be distributed such that they affect other S-Boxes in the following round

Design Issues for Block Ciphers (II)

Number of Rounds:

- More rounds are generally better, but they cost in reduced performance
- Number of rounds should maximize the Avalanche Effect (about 50% of output bits should change for any change in input bit)
- Number of rounds should be selected to make the effects of advanced attacks (differential / linear / etc) be similar to exhaustive search (when taking into account the overhead required to run the attacks)

Design Issues for Block Ciphers (III)

F-Function:

- Must be difficult to un-scramble
- Should be non-linear
- SAC (Strict Avalanche Criteria) – any output bit should be inverted with probability ½ when some input bit is changed
- BIC (Bit Independence Criteria) – any two output bits should change independently when some input bit is changed

DES Design Issues (I)

- Unofficial requirement to make DES slow in software
- The NSA reduction of the key-size to 56 bits
- NSA changes to the S-Boxes
- "Initial Permutation" and "Inverse Initial Permutation"
- Effects of the "Expand" and "Permute" operations
- Effects of the "f" scrambling function
- Effects of the S-Boxes
- Exportability of cryptographic algorithms, software, and hardware

DES Design Issues (II)

Avalanche Effect in DES – Change in Plaintext:

- Number of output bits that change when one input bit is changed

Round:	0	1	2	3	4	5	6
Bits:	1	6	21	35	39	34	32

Round:	7	8	9	10	11	12	13
Bits:	31	29	42	44	32	30	30

Round:	14	15	16
Bits:	26	29	34

DES Design Issues (III)

Avalanche Effect in DES – Change in Key:

- Number of output bits that change when one key bit is changed

Round:	0	1	2	3	4	5	6
Bits:	0	2	14	28	32	30	32
Round:	7	8	9	10	11	12	13
Bits:	35	34	40	38	31	33	28
Round:	14	15	16				
Bits:	26	34	35				

DES Design Issues (IV)

Avalanche Effect in DES and Number of Rounds:

- For output to appear random - number of bits that change should be around 50% (that is – 32 bits)
- With 16 DES rounds – the Avalanche Effect DES is about optimal
- Also 16 rounds is large enough to withstand certain cryptanalytical attacks

DES Design Issues (V)

Weak DES Keys:

- 4 keys in which each half of the key (after PC-1) is either all 0's or all 1's
- For these keys: $E_K(E_K(X)) = X$

Semi-Weak DES Keys:

- 12 keys in which each half of the key (after PC-1) is one of the following: all 0's, all 1's, alternating 0's and 1's, and alternating 1's and 0's
- For pairs of keys: $E_{K_1}(E_{K_2}(X)) = X$

DES Strength

- Since 1975, there was a debate regarding the selection of only 56 bits for the DES key size
- Exhaustive Search Attack:
  Requires searching $O(2^{56})$ keys
- Differential Cryptanalysis:
  Requires analyzing $O(2^{47})$ chosen plaintexts
- Linear Cryptanalysis:
  Requires analyzing $O(2^{47})$ known plaintexts
- In 1990's – DES was declared not secure enough by the technical community (IETF)

DES – Exhaustive Search

- Exhaustive Search Attack:
  - Search space of $O(2^{56}) = O(10^{17})$ keys
  - In the 1970's, Diffie and Hellman suggested a $20M-machine that will crack DES in about one day
  - In the 1990's, Wiener suggested a $1M-machine that will crack DES in 3.5 hours
  - Assume about 10^9 encryptions per second on today's computers. Then about 10^8 computers seconds are required to crack DES
  - In 1990's, DES challenges were broken in matter of days using distributed clusters of computers
  - Presumably, most national security agencies have the hardware and software to crack DES in hours

DES – Differential Cryptanalysis

- Differential Cryptanalysis Attack:
  - Study the differences between two encryptions of two different plaintext blocks M and M*
  - Study the probability of output differences in each S-Box
  - Trace back differences to specific S-Boxes
  - Estimate the likelihood of key-bits involved in the XOR operation before the S-Boxes
  - Continue developing estimates for key until one key emerges as the only ultimate option
  - Chosen space of $O(2^{47})$ plaintexts
  - Not practical – but theoretically important

DES – Linear Cryptanalysis

- Linear Cryptanalysis Attack:
  - Approximate the DES key as a linear transformation of the plaintext bits and the ciphertext bits
  - Change the coefficients based on multiple values of pairs of <plaintext,ciphertext>
  - Requires known space of $O(2^{47})$ <plaintext,ciphertext> pairs
  - Not practical – but theoretically important

DES Strength – Summary

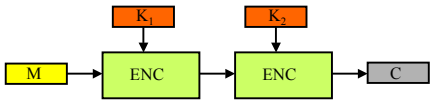
- Since early 1990's – DES is considered not secure enough for technical and commercial use
- Several approaches:
  - Strengthening DES – 2-DES
  - Strengthening DES – 3-DES
  - Strengthening DES – DES-X
  - Other Algorithms

Symmetric Cryptography II

Prof. Shlomo Kipnis
November 12, 2007

Double DES (I)

- Apply two iterations of DES with two keys K_1 and K_2



- What if DES has a structure of an algebraic group, such that for each K_1 and K_2 there is a K_3 with the property:

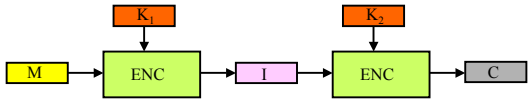
$$E_{K_2}(E_{K_1}(X)) = E_{K_3}(X)$$

- This is not the case (proved in 1992)
- BUT - is the security level $O(2^{112})$?

Double DES (II)

Meet-in-the-Middle Attack:

- For given M and C – search only $O(2^{56})$ pairs of keys K_1 and K_2 at the intermediate point I



- Encrypt M under all 2^{56} options for K_1
  - Denote the results by $X_1, X_2, \ldots, X_{2^{56}}$
- Decrypt C under all 2^{56} options for K_2
  - Denote the results by $Y_1, Y_2, \ldots, Y_{2^{56}}$

Double DES (III)

Meet-in-the-Middle Attack (continued):

- Sort the values $X_1, X_2, \ldots, X_{2^{56}}$
- Sort the values $Y_1, Y_2, \ldots, Y_{2^{56}}$
- Find collisions between values of X_i and Y_j – there should be about $(2^{112} / 2^{64}) = 2^{48}$ such collisions
- Mark the 2^{48} potential candidates for key-pair K_1 and K_2
- Take another pair M' and C', and repeat the above test with the new pair and the suspected 2^{48} key-pairs
- The second test will pin down the correct pair K_1 and K_2 (since the success rate will be $(2^{48} / 2^{64}) = 2^{-16}$)

Double DES (IV)

Meet-in-the-Middle Attack (continued):

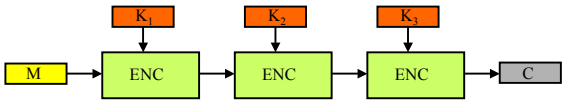
- Time requirement of attack – for encryptions, decryptions, sorting, and comparing:

  about $O(2^{64})$ steps
- Space requirement of attack – for keeping the encryption and decryption values:

  about $O(2^{60})$ bytes
- Although attack is not very practical – it is sufficiently intriguing to consider Double-DES not secure enough

Triple-DES (I)

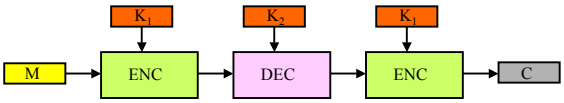
- EEE Mode:
  - DES Encrypt-Encrypt-Encrypt with three keys K_1, K_2, and K_3



- Properties:
  - Three keys (168 bits)
  - Strength about $O(2^{110})$ against Meet-in-the-Middle
  - Not compatible with regular DES

Triple-DES (II)

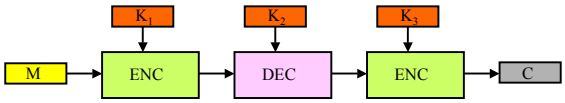
- EDE Mode:
  - DES Encrypt-Decrypt-Encrypt with two keys K_1, and K_2



- Properties:
  - Two keys (112 bits)
  - Strength about $O(2^{110})$ against Meet-in-the-Middle
  - Compatible with regular DES when $K_1 = K_2$

Triple-DES (III)

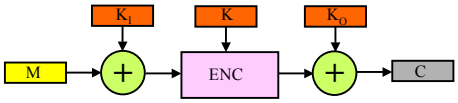
- EDE Mode with three keys:
  - DES Encrypt-Decrypt-Encrypt with keys K_1, K_2, and K_3



- Properties:
  - Three keys (168 bits)
  - More resilient against attacks
  - Compatible with regular DES when $K_1 = K_2$ or $K_2 = K_3$

DES-X

- Whitening Technique:
  - Three keys – K_O ; K ; K_I
  - $C = K_o \oplus DES_K(M \oplus K_I)$



- Strength of 3-DES and of DES-X is considered to be about $O(2^{120})$

Other Algorithms

- DES is (was) the most widely used encryption algorithm
- Other popular encryptions algorithms include:
  - IDEA (1990)
  - RC5 (1990)
  - AES (2000)

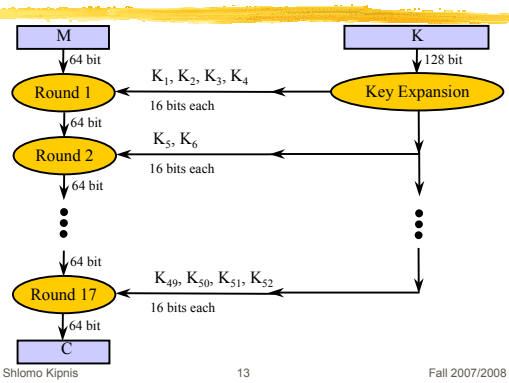
IDEA (I)

- IDEA – International Data Encryption Algorithm
- Developed in late 1980's in Switzerland
- Efficient in software and hardware
- Used in many software pakages (e.g., PGP)
- Block size – 64 bits
- Key size – 128 bits
- Based on a variation of Feistel scheme
- Encryption and decryption are identical with the only difference in the sub-key generation scheme

IDEA (II)

- IDEA uses 17 rounds (or 8½ double-rounds)
  - Each odd round uses 4 sub-keys (16 bits each)
  - Each even round uses 2 sub-keys (16 bits each)
  - Total number of sub-keys used in IDEA is 52
  - IDEA generates the 52 sub-keys from the 128-bit key by first consuming the 128 bits of the key (to generate 8 sub-keys), and then shifting the key by 25 bits to generate more sub-keys, and so on until all 52 sub-keys are generated:
    - $K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8$ are taken from $K[1..128]$
    - $K_9, K_{10}, K_{11}, K_{12}, K_{13}, K_{14}, K_{15}, K_{16}$ are taken from $K[26..25]$
    - $K_{17}, K_{18}, K_{19}, K_{20}, K_{21}, K_{22}, K_{23}, K_{24}$ are taken from $K[51..50]$
    - Etc.

IDEA (III)

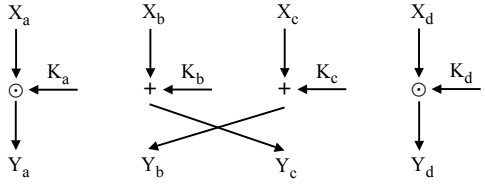


IDEA (IV)

- Scrambling functions in IDEA are based on mathematical operations (XOR, modular ADD, modular MULT):
  - $\oplus$ – Bitwise XOR: on registers of 16 bits
  - $+$ – Addition modulo $65536 = 2^{16}$: on numbers represented in 16-bits
  - $\odot$ – Multiplication modulo $65537 = 2^{16}+1$: on numbers represented in 16-bits

IDEA (V)

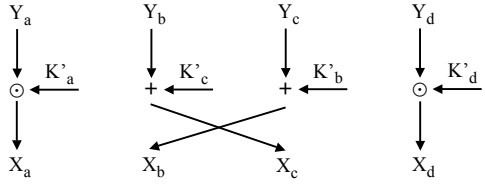
- Scrambling function of odd rounds:



- X_a, X_b, X_c, X_d are four 16-bit inputs to the round
- K_a, K_b, K_c, K_d are four 16-bit keys of the round
- Y_a, Y_b, Y_c, Y_d are four 16-bit outputs from the round

IDEA (VI)

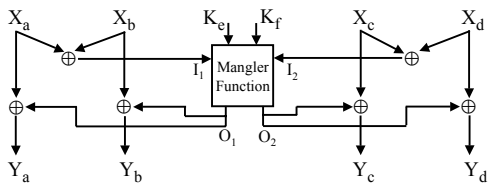
- Inverse scrambling of odd rounds:



- K'_a and K'_d are multiplicative inverses of K_a and K_d
- K'_b and K'_c are additive inverses of K_b and K_c
- X_a, X_b, X_c, X_d are derived back from Y_a, Y_b, Y_c, Y_d

IDEA (VII)

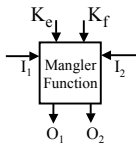
- Scrambling function of even rounds:



- X_a, X_b, X_c, X_d are four 16-bit inputs to the round
- K_e and K_f are two 16-bit keys of the round
- Y_a, Y_b, Y_c, Y_d are four 16-bit outputs from the round

IDEA (VIII)

- Mangler Function:



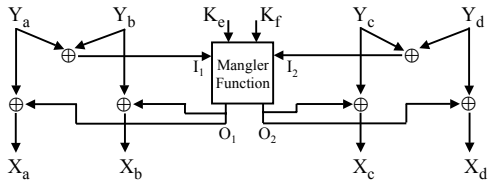
- $O_1 = ((K_e \odot I_1) + I_2) \odot K_f$
- $O_2 = (K_e \odot I_1) + O_1$
- K_e and K_f are 16-bit keys of the round
- A reminder:

$$I_1 = X_a \oplus X_b$$

$$I_2 = X_c \oplus X_d$$

IDEA (IX)

- Inverse scrambling of even rounds:



- Same K_e and K_f are used in the inverse scrambling
- Mangler function gets same inputs and $\oplus$ is invertible
- X_a, X_b, X_c, X_d are derived back from Y_a, Y_b, Y_c, Y_d

IDEA (X)

- Inputs to Mangler Function:

  $I_1 = X_a \oplus X_b = Y_a \oplus Y_b$

  $I_2 = X_c \oplus X_d = Y_c \oplus Y_c$

  Same K_e and K_f
- Therefore – outputs from Mangler Function are identical in encryption and decryption
- Now if $Y = X \oplus O$
  then $X = Y \oplus O$

IDEA (XI)

- Encryption Keys:

  $K_1, K_2, K_3, K_4, K_5, K_6, K_7, K_8, K_9, K_{10}, K_{11}, K_{12}, K_{13}, K_{14}, K_{15}, K_{16}, K_{17}, K_{18}, K_{19}, K_{20}, K_{21}, K_{22}, K_{23}, K_{24}, K_{25}, K_{26}, K_{27}, K_{28}, K_{29}, K_{30}, K_{31}, K_{32}, K_{33}, K_{34}, K_{35}, K_{36}, K_{37}, K_{38}, K_{39}, K_{40}, K_{41}, K_{42}, K_{43}, K_{44}, K_{45}, K_{46}, K_{47}, K_{48}, K_{49}, K_{50}, K_{51}, K_{52}$
- Decryption Keys:

  $K_{49}^{-1}, -K_{51}, -K_{50}, K_{52}^{-1}, K_{47}, K_{48}, K_{43}^{-1}, -K_{45}, -K_{44}, K_{46}^{-1}, K_{41}, K_{42}, K_{37}^{-1}, -K_{39}, -K_{38}, K_{40}^{-1}, K_{35}, K_{36}, K_{31}^{-1}, -K_{33}, -K_{32}, K_{34}^{-1}, K_{29}, K_{30}, K_{25}^{-1}, -K_{27}, -K_{26}, K_{28}^{-1}, K_{23}, K_{24}, K_{19}^{-1}, -K_{21}, -K_{20}, K_{22}^{-1}, K_{17}, K_{18}, K_{13}^{-1}, -K_{15}, -K_{14}, K_{16}^{-1}, K_{11}, K_{12}, K_{7}^{-1}, -K_{9}, -K_{8}, K_{10}^{-1}, K_{5}, K_{6}, K_{1}^{-1}, -K_{3}, -K_{2}, K_{4}^{-1}$

IDEA (XII)

- Summary:
  - IDEA is a strong encryption algorithm
  - Simple binary operations (XOR, ADD, MULT)
  - Difficult to cryptanalyze (because of 3 operations, use of 16 bits in the manipulations, and number of rounds)
  - Efficient in software (simple operations, no convoluted permutations, same algorithm for encryption and decryption)
  - Efficient in hardware (simple modules, simple operations on 16-bit registers)
  - Used in commercial products (PGP and some standards)

RC5 (I)

- Designed by Rivest in 1994
- Design Characteristics:
  - Efficient in hardware and software
  - Fast implementations
  - Different word lengths
  - Variable number of rounds
  - Variable key length
  - Low memory requirement
  - High security
  - Data-dependant rotations

RC5 (II)

- RC5 Operational Parameters:
  - w – word size in bits (16 / 32 / 64)
  - r – number of rounds (between 0 and 255)
  - b – number of bytes in secret key (between 0 and 255)
- Nominal values for parameters:
  - w = 32
  - r = 12
  - b = 16

RC5 (III)

Key Expansion:

- Given key K with b bytes – K[0] . . . K[b-1]
- When r rounds are used – then t = 2r+2 sub-keys need to be prepared
- Array S will keep the t sub-keys – S[0] . . . S[t-1]
- Two hexadecimal constants are used to initialize S. For w = 16 / 32 / 64 bits, these constants are:
  - P_w = B7E1 / B7E15163 / B7E151628AED2A6B
  - Q_w = 9E37 / 9E3779B9 / 9E3779B97F4A7C15

RC5 (IV)

Key Expansion (continued):

- Initialization code for array S:

```
S[0] = P_w
for i=1 to t-1 do
    S[i] = S[i-1] + Q_w        /// addition modulo 2^w
```

- Define c = ⌈ 8b/w ⌉
- The b-bite array K is copied into a c-word array L (with leaving zeros at the end of array L)

RC5 (V)

Key Expansion (continued):

- Array S is mixed with array L to produce the set of final values in array S:

```
i = j = X = Y = 0
do 3 • max(t, c) times
    S[i] = (S[i] + X + Y) <<< 3
    X = S[i]
    i = i + 1 (mod t)
    L[j] = (L[j] + X + Y) <<< (X+Y)
    Y = L[j]
    j = j + 1 (mod c)
```


RC5 (VI)

Encryption and Decryption Primitives:

Three primitive operations (and their inverses) are used in the encryption (and decryption):

+	Addition of w-bit words modulo 2^w
–	Subtraction modulo 2^w (inverse of addition)
⊕	Bit-wise XOR
⊕	XOR is its own inverse
<<<	Left circular shift
>>>	Right circular shift (inverse of left circular shift)

RC5 (VII)

Encryption:

- Plaintext block resides initially in two w-bit registers A and B
- Following round i – left half of data is called LE_i and right half of data is called RE_i

```
LE_0 = A + S[0]
RE_0 = B + S[1]
for i=1 to r do
    LE_i = ((LE_{i-1} ⊕ RE_{i-1})) <<< RE_{i-1}) + S[2i]
    RE_i = ((RE_{i-1} ⊕ LE_i)) <<< LE_i) + S[2i+1]
```

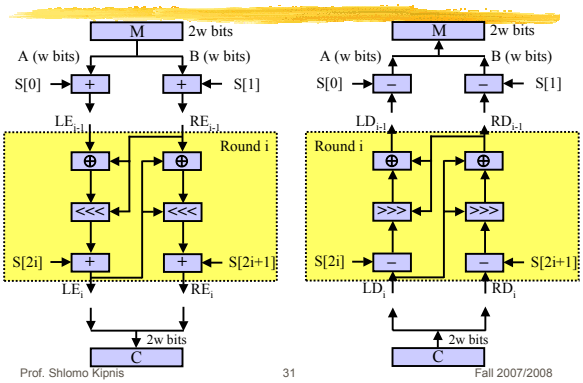

RC5 (VIII)

Decryption:

- Ciphertext block resides initially in two w-bit registers LD_r and RD_r
- Before round i – left half of data is called LD_i and right half of data is called RD_i

```
for i=r down to 1 do
    RD_{i-1} = ((RD_i – S[2i+1] >>> LD_i) ⊕ LD_i)
    LD_{i-1} = ((LD_i – S[2i] >>> Rd_{i-1}) ⊕ Rd_{i-1})
B = RD_0 – S[1]
A = LD_0 – S[0]
```

RC5 (IX)



RC5 (X)

RC5 Modes:

- RC5 Block Cipher – This is the raw algorithm which takes $2w$-bit plaintext blocks and produces $2w$-bit ciphertext blocks. This mode is also called ECB (Electronic Code Book) Mode
- RC5-CBC – This is the cipher-block-chaining mode of RC5. This mode only handles plaintext with a multiple of $2w$ bits.
- RC5-CBC-Pad – This is the cipher-block-chaining mode of RC5 that handles plaintext of any length. It does so by padding the plaintext to be of size of the next multiple of $2w$ bits.
- RC5-CBC-CTS – This is a Cipher Text Stealing mode. This mode handles plaintext of any length and produces a corresponding ciphertext of the same length.

RC5 (XI)

CBC Padding:

- If the length of the message is not a multiple of $2w$ bits, than the message can be padded.
- Up to $2w/8$ bytes might be added to the length of the message
- The pad bytes are all the same and are set to a value that represents the number of bytes of padding. (For example, if there are 6 bytes of padding, then the last 6 bytes will be set to the binary value 00000110.)
- Padding may not be desirable in applications or communications that need to preserve the size of the data

RC5 (XII)

RC5-CBC-CTS Mode:

If plaintext consists of N blocks, and last block is only L bytes long, where $L < 2w/8$:

- Encrypt the first $(N-2)$ blocks using RC5-CBC mode
- XOR block M_{N-1} with the previous ciphertext block C_{N-2} and denote the resulting block Y_{N-1}
- Encrypt block Y_{N-1} to create E_{N-1}
- Select the first L bytes of E_{N-1} to create C_N
- Pad M_N with 0's at the end and XOR with E_{N-1} to create Y_N
- Encrypt Y_N to create C_{N-1}
- Last two blocks of the ciphertext are C_{N-1} and C_N

AES (I)

- AES History:
  - Other algorithms were not appropriate
    - Not strong enough (DES)
    - Not fast enough (3-DES)
    - Not available freely (IDEA, RC5)
  - NIST published RFP for Advanced Encryption Algorithm in 1997:
    - Fully specified and explained algorithm
    - Variable strength by key size (from 128 to 256 bits)
    - Efficient implementation on various SW & HW platforms
  - About 20 algorithms were proposed
  - Open review process for about 3 years
  - Rijndael was selected in November 2001

AES (II)

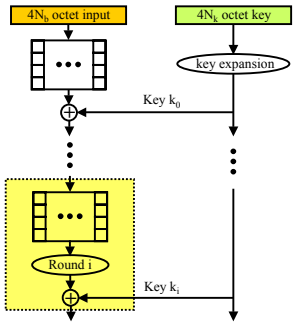
- Rijndael Parameters:
  - N_b = Block size in 32-bit words – 4 / 5 / 6 / 7 / 8 (that is – 128 / 160 / 192 / 224 / 256 bits)
  - N_k = Key size in 32-bit words – 4 / 5 / 6 / 7 / 8 (that is – 128 / 160 / 192 / 224 / 256 bits)
  - N_r = Number of rounds – $6 + \max(N_b, N_k)$
- AES Parameters:
  - N_b = Block size in 32-bit words – 4 (that is – 128 bits)
  - N_k = Key size in 32-bit words – 4 / 6 / 8 (that is – 128 / 192 / 256 bits)
  - Number of rounds –
    - AES-128 – 10 rounds
    - AES-192 – 12 rounds
    - AES-256 – 14 rounds

AES (III)

- AES maintains a state:
  - State is a rectangular array of 4 rows by $N_b = 4$ columns
  - Each of the $4N_b = 16$ array entries holds an octet (8 bits)
  - Initial value of state is the plaintext block entered column by column
  - State is transformed during N_r rounds
  - Final value of state is the ciphertext block read column by column
- AES maintains a key-expansion scheme:
  - Key is used to generate a sequence of key-sets
  - Each key-set consists of N_k columns of 4 octets (32 bits) each
  - Key-expansion generates $(N_r+1)N_b$ 4-octet columns

AES (IV)

- $4N_b$ octet input
- $4N_k$ octet key
- Placement of input in the state array
- Key expansion
- XOR input with K_0
- N_r rounds of state transformation and XOR with K_i



AES (V)

Four primitive operations inside each round:

- Bit-wise XOR – ⊕
- S-box that substitutes octet for octet. The S-box substitution is implemented as a table lookup.
- Rearrangement of octets that consists of rotating rows by some number of cells.
- A Mix-Column operation that replaces a 4-octet column with another 4-octet column. The Mix-Column operation can be implemented as a table lookup.

AES (VI)

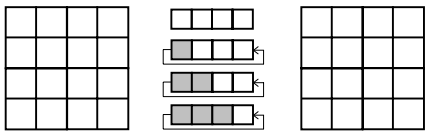
S-Box Permutation

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	64	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	09	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0d	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

AES (VII)

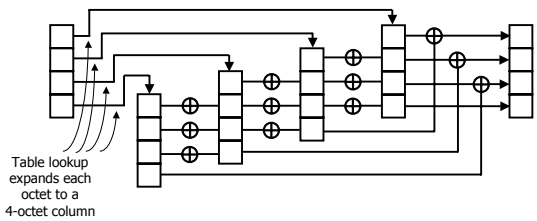
The Row Rotations ($4N_b$ cells of 8 bits each):

- Row 0 of the state is not rotated
- Row 1 of the state is rotated left 1 column
- Row 2 of the state is rotated left 2 columns
- Row 3 of the state is rotated left 3 columns



AES (VIII)

The Mix-Column operation on each 4-octet column:



This operation computes a new 4-octet column

AES (IX)

Inverse AES:

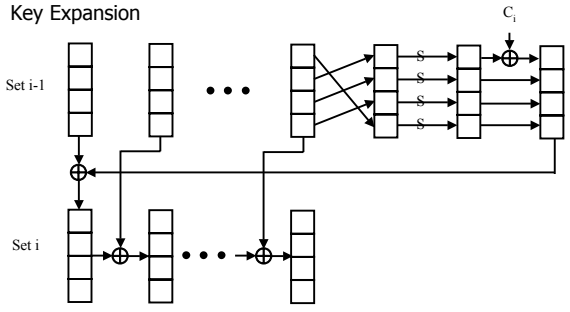
- XOR – $\oplus$ – is its own inverse
- Since the S-box is a permutation – there is an inverse S-Box. The inverse S-box substitution can be implemented as a table lookup.
- Inverses of the row and column rotations are rotations in the opposite directions.
- The inverse of the Mix-Column operation is the Inv-Mix-Column operation that can be implemented as another table lookup.

AES (X)

Key Expansion:

- Arrange the initial $4N_k$ octet key as N_k columns of 4 octets each. This is key-set number 0.
- Iteratively generate the next key-sets. Each key-set consists of N_k columns and is derived from the previous key-set.
- Key-set number i is generated as follows:
  - Column 0 is obtained by rotating the last column of key-set i-1 upward one cell, applying the S-box to each octet, and XORing the first octet with a constant C_i (which is based on i).
  - Other columns are generated by XORing the previous column in key-set number i with the corresponding column in key-set number i-1.

AES (XI)



AES (XII)

AES Operation:

- Can be implemented with table lookup operations, instead of multiplications and additions of polynomials
- Round transformations of AES can be parallelized
- AES can be efficiently implemented over a wide range of platforms:
  - hardware (8051, 68xxx, Pentium)
  - software (ANSI C and Java over 32-bit processors)
- AES has been tested and found resilient against all known cryptanalysis techniques

Mathematics of AES (I)

One AES Field:

- AES uses arithmetic over the field $GF(2^8)$.
- Each byte (octet) in AES represents an element in $GF(2^8)$.
- Bit-wise XOR of two octets in AES is the addition of corresponding elements in $GF(2^8)$.
- A second application of bit-wise XOR (with the same value) achieves the effect of subtracting.
- AES defines an irreducible polynomial $m(x) = x^8+x^4+x^3+x+1$ over $\mathbf{Z}_2$. Multiplication of octets in AES is done by multiplying the polynomials (which correspond to the octets) and taking modulo $m(x)$. The result also fits in an octet.

Mathematics of AES (II)

One AES Field (continued):

- The multiplication table of $GF(2^8)$ is of size $256 \times 256 = 65536$ entries of 1-octet each.
- Luckily, AES requires multiplication only by six different constants, which can be done by a much smaller table of size $256 \times 6 = 1536$ entries of 1-octet each.
- Multiplicative inverses of the 255 elements (excluding the 0) are obtained by a table lookup of size 255 entries of 1-octet each.
- The constants C_i used in the key-expansion are defined to be: $C_i = x^{i-1} \bmod m(x)$. These constants are also kept in a table.

Mathematics of AES (III)

Another AES Field:

- AES also uses polynomials (of degree up to 3) with coefficients in $GF(2^8)$, that is, with 8-bit vectors as coefficients.
- Addition of coefficients of these polynomials is bit-wise XOR of the 8-bit vectors.
- For multiplication of these polynomials, the following degree-4 irreducible polynomial x^4+1 over $GF(2^8)$ is used.
- Each of these polynomials is represented by a 4-octet column.
- For these polynomials, multiplication by x is a left-cyclic rotation.
- The Mix-Column operation is a multiplication by the fixed polynomial $c(x) = 03x^3+01x^2+01x+02$. And, the Inv-Mix-Column operation is a multiplication by the fixed polynomial $d(x) = 0Bx^3+0Dx^2+09x+0E$.

Mathematics of AES (IV)

Yet Another AES Field:

- AES also uses polynomials (of degree up to 7) with coefficients in $\mathbf{Z}_2$. For these operations, the irreducible polynomial x^8+1 over $\mathbf{Z}_2$ is used.
- The AES S-Box transformation consists of taking an octet (which is a representation of a polynomial of degree at most 7) and doing:
  - taking the multiplicative inverse modulo $m(x)$ of the octet
  - multiplying by $x^4+x^3+x^2+x+1$ modulo x^8+1
  - adding x^6+x^5+x+1
- The AES inverse S-Box consists of doing:
  - multiplying by x^6+x^3+x modulo x^8+1
  - adding x^2+1
  - taking the multiplicative inverse modulo $m(x)$ of the result

Symmetric Cryptography III

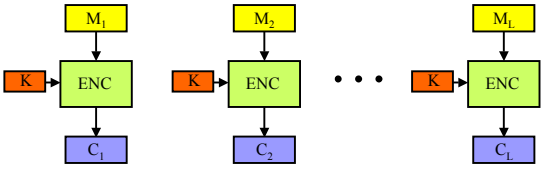
Prof. Shlomo Kipnis
November 14, 2007

Encrypting Long Texts (I)

- Message M is divided into L blocks, each consisting of N bits: $M = M_1, M_2, \ldots, M_L$
- Cipher C is constructed of L blocks, each consisting of N bits: $C = C_1, C_2, \ldots, C_L$
- Three modes of building the cipher blocks
- ECB – Electronic Code Book
- CTR – Counter Mode
- CBC – Cipher Block Chaining

Encrypting Long Texts (II)

ECB – Electronic Code Book



- Each block is encrypted independently of the others
- Encryption: $C_i = E_K(M_i)$
- Decryption: $M_i = D_K(C_i)$

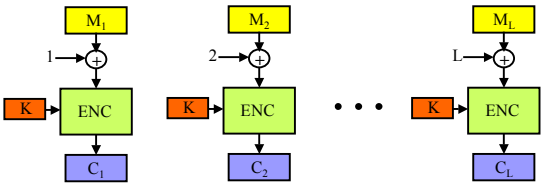
Encrypting Long Texts (III)

ECB – Electronic Code Book

- Advantages:
  - Simple implementation
  - Parallel encryption
  - Parallel decryption
  - Random Access to blocks
- Disadvantages:
  - Identical blocks encrypt the same
  - No protection on order of blocks
  - No protection against block loss / change

Encrypting Long Texts (IV)

CTR – Counter Mode



- Each block is encrypted independently of the others
- Encryption: $C_i = E_K(M_i \oplus i)$
- Decryption: $M_i = D_K(C_i) \oplus i$

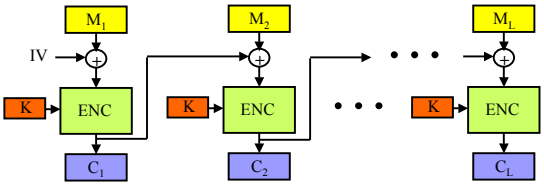
Encrypting Long Texts (V)

CTR – Counter Mode

- Advantages:
  - Simple implementation
  - Parallel encryption
  - Parallel decryption
  - Random Access to blocks
  - Identical blocks encrypt differently
  - Some protection on order of blocks
- Disadvantages:
  - No protection against block loss
  - Explicit synchronization is needed

Encrypting Long Texts (VI)

CBC – Cipher Block Chaining



- Each block is encrypted and decrypted as follows:
- Encryption: $C_i = E_K(M_i \oplus C_{i-1})$
- Decryption: $M_i = D_K(C_i) \oplus C_{i-1}$

Encrypting Long Texts (VII)

CBC – Cipher Block Chaining

- Advantages:
  - Parallel decryption
  - Identical blocks encrypt differently
  - Protection on order of blocks
  - Implicit synchronization (blocks C_i and C_{i-1} needed for M_i)
  - Using IV to randomize ciphertext
- Disadvantages:
  - Serial encryption

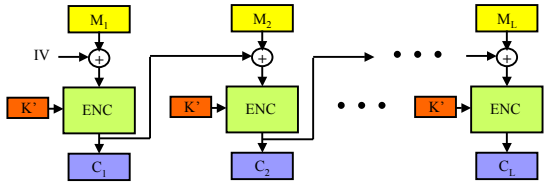
Encrypting Long Texts (VIII)

Attacks on Chaining Modes

- Detecting similar ciphertext blocks in ECB mode
- Dropping the last block of ciphertext in all modes
- Rearranging ciphertext blocks in ECB and CTR modes
- Changing certain bits in block of ciphertext in CBC mode

CBC Message Authentication Code (I)

- Computing MAC of a message with CBC-Mode:



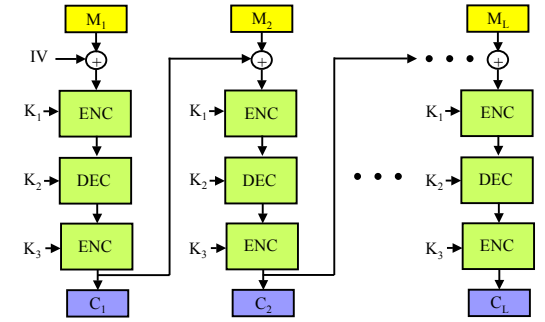
- If key K' is secret between sender and receiver – then last block C_L can serve as MAC
- **Note**: Should use different keys for encryption and MAC

CBC Message Authentication Code (II)

- Computing MAC of a message with CBC-Mode:
  - CBC residue is dependant on all the plaintext blocks
  - Chance of $1/2^{64}$ (in the case of DES) of changing the message and getting the MAC right
  - MAC of 64 bits might be too weak – one might need to use MAC of double the size (using two keys)
  - Option of computing the CBC residue of the plaintext, and then encrypting it with another key

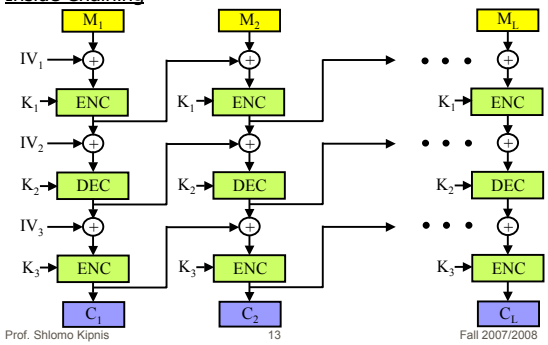
Encrypting Long Texts with 3-DES (I)

Outside Chaining



Encrypting Long Texts with 3-DES (II)

Inside Chaining



Encrypting Long Texts with 3-DES (III)

Outside Chaining

- Attack of changing certain bits in the ciphertext to cause specific changes in the plaintext is still valid
- Slower by a factor of 3 and cannot be pipelined
- Maintains the self-synchronization property
- Better treated as a "black box" for purposes of different chaining modes
- Widely used in practice

Encrypting Long Texts with 3-DES (IV)

Inside Chaining

- Attack of changing certain bits in the ciphertext to cause specific changes in the plaintext is not valid
- Slower by a factor of 3 but can be pipelined
- Does not maintain the self-synchronization property
- Chaining and composition modes need to get into the "black box"
- Used less in practice

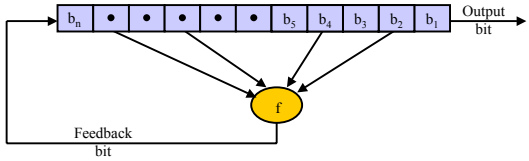
Stream Ciphers

Stream Ciphers:

- Plaintext is handled in small groups of bits (typically 1 or 8 each time) and the encryption transformation maintains memory (state) between encryption steps
- Linear Feedback Shift Registers
- Output Feedback Mode
- Cipher Feedback Mode
- RC4
- A5

Linear Feedback Shift Registers (I)

- LFSR – Linear Feedback Shift Register



- Generation of pseudo random bits
- Dependency on initial value
- Length of cycle depends on function f and on length n

Linear Feedback Shift Registers (II)

- Cycle length:
  - Maximal cycle length is 2^n-1
  - Mostly used function is XOR
  - To obtain a maximal cycle – the polynomial formed from the tap sequence plus the constant 1 must be a primitive polynomial modulo 2.
    (The polynomial must divide $X^{2^n-1}+1$ but not other X^d+1 for any d that divides 2^n-1.)
  - Example:

    The polynomial $X^{32}+X^7+X^5+X^3+X^2+X+1$ is a primitive polynomial modulo 2.

    Therefore, the 32-bit LFSR that XOR's bits 32, 7, 5, 3, 2, and 1 cycles through all the $2^{32}-1$ states.

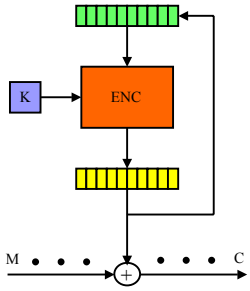
Linear Feedback Shift Registers (III)

- Most early stream ciphers were based on LFSR's
- LFSR's are still in wide use today in military and in civilian applications
- Many systems combine several LFSR's to increase the complexity and to reduce the correlation of bits
- For an LFSR to be cryptographically secure, it must use many bits in the shift register (i.e., the polynomial must be 'dense')
- LFSR's can be made to be very fast in hardware
- LFSR's are slow in software

Output Feedback Mode (I)

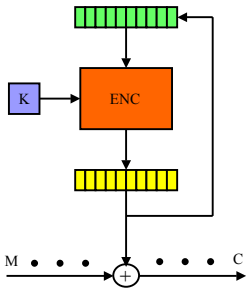
OFM – Output Feedback Mode

- Block encryption algorithm is used to generate a pad
- Pad is used to XOR with bits of incoming message M
- When pad bits are exhausted, the output register is fed into the input register, and another cycle of encryption is performed



Output Feedback Mode (II)

- Specified as one of the operation modes of DES
- Can be used with any encryption algorithm
- Input register is initially loaded with an IV
- IV is transmitted between the parties at the beginning of the communication
- Feedback can be full size of output register or only b bits



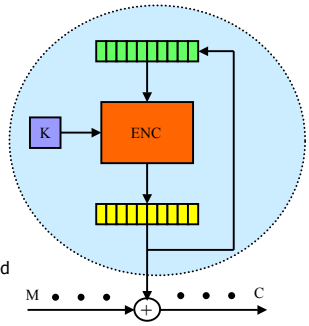
Output Feedback Mode (III)

Advantages:

- OFM is a Finite Automaton with internal feedback
- Easy to analyze
- Offline preparation of pad bits
- Fast online performance

Disadvantages:

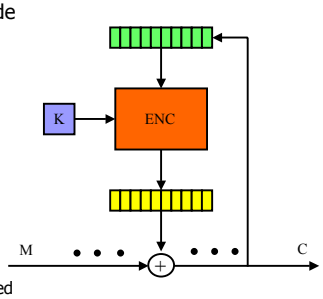
- Pad is not dependant on input bits
- Explicit synchronization is needed in cases of communication loss
- Bit positions can be switched



Cipher Feedback Mode (I)

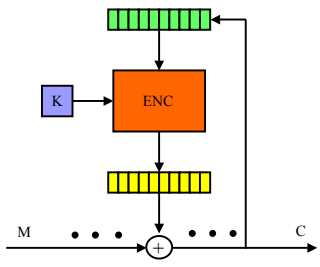
CFM – Cipher Feedback Mode

- Block encryption algorithm is used to generate a pad
- Pad is used to XOR with bits of incoming message M
- When pad bits are exhausted, cipher bits are fed into the input register, and another cycle of encryption is performed



Cipher Feedback Mode (II)

- Specified as one of the operation modes of DES
- Can be used with any encryption algorithm
- Input register is initially loaded with an IV
- IV is transmitted between the parties at the start of the communication
- Feedback can be full size of output register or only b bits



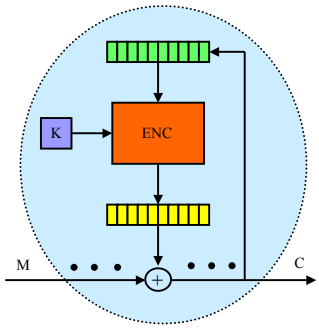
Cipher Feedback Mode (III)

Advantages:

- Pad is dependant on input bits
- Synchronization is implicit
- Automatic recovery from communication errors
- More resistant to attacks

Disadvantages:

- Cannot generate long pads in advance
- Online performance is less than that of OFM



RC4 (I)

- Developed by Rivest in 1994
- Kept as a trade secret (but leaked)
- Key can be between 1 and 256 bytes
- Used as a simple and fast generator of pseudo-random sequences of bytes (to be used as "one-time-pad")
- Should discard first 256 bytes of generated pad
- Passes all usual randomness tests

RC4 (II)

Code for RC4 – state initialization:

```
static unsigned char state[256], x, y;
void        rc4init( key, length)
            unsigned char *key;
            unsigned short length;
            {
                        int         i;
                        unsigned char t, j, k=0;

                        for ( i=256; i--; )
                                    state[i] = i;
                        for ( i=0, j=0; i<256; i++, j=(j+1) % length )
                                    t = state[i],  state[i] = state[k += key[j] + t],  state[k] = t;
                        x = 0;
                        y = 0;
            }
```


RC4 (III)

Code for RC4 – generation of pad:

```
unsigned char  rc4step()
                        {
                                    unsigned char         t;

                                    t = state[y += state[++x],  state[y] = state[x],  state[x] = t;
                                    return( state[ state[x] + state[y]] );
                        }
```


A5

The stream cipher used in GSM phones:

- Held in confidence but leaked
- Papers and code available on Internet
- Includes 3 LFSR (of sizes 19, 22, and 23 bits)
- Each register is clocked separately
- There is an attack requiring $O(2^{40})$ encryptions:
  - Guess content of first two LFSR's
  - Determine third LFSR from the keystream
- Overall design of A5 is good
- Weaknesses can be overcome with longer registers

Summary

- The design of a good encryption algorithm is not a simple task; it is better to leave this task to skilled cryptographers
- There are many good commercial algorithms from which one may select
- Today's keys are long enough (128 / 192 / 256 bits) to overcome all known attacks
- One may compose several algorithms, and the result will probably be strong
- Use algorithms with care

Authentication and Integrity

Prof. Shlomo Kipnis
November 19, 2007

Authenticity and Integrity Attacks

- Source Impersonation / Source Spoofing
- Message Injection / Message Fabrication
- Message Modification / Message Alteration
- Message Re-Sequencing / Message Re-Ordering
- Message Re-Timing / Message Replaying
- Message Repudiation / Transaction Denying

Authentication and Integrity Services

- Identity Authentication – proving the identity of an entity in a conversation
- Source Authentication – proving that a message has indeed arrived from the claimed source
- Message Authentication – proving that the content of the received message was not altered
- Sequence Authentication – proving order of messages
- Timing Authentication – proving timeliness of messages
- Message Signing – proving to a third-party that some message was indeed sent by a known entity

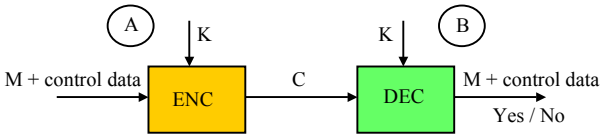
Authentication Functions

- Creating an authenticator for a particular type of authentication service may involve functions of:
  - Entity Name / Message / Text
  - Time Stamp / Sequence Number / Random Value
  - Symmetric Secret Key / Asymmetric Private Key
- The sender computes and sends the authenticator as part of / in addition to the regular message
- The recipient compares the received authenticator with the expected authenticator

Authentication Methods

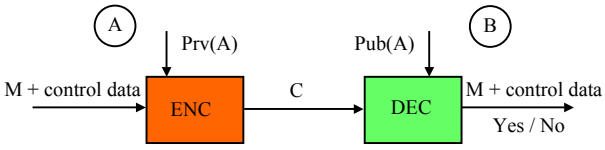
- Authentication by Encryption – using encryption functions with secret keys:
  - Symmetric: key shared between sender and receiver
  - Asymmetric: private key by sender / public key by receiver
- Authentication by MAC – using MAC functions of the text and secret keys:
  - Symmetric: key shared between sender and receiver
- Authentication by Hash – using hash functions and involving secret keys in the computations:
  - Symmetric: key shared between sender and receiver
  - Asymmetric: private key by sender / public key by receiver

Authentication by Symmetric Encryption



- A encrypts message M and some control data using algorithm ENC with symmetric key K to obtain cipher C
- B decrypts cipher C using algorithm DEC with symmetric key K to obtain message M and control data
- B checks control data and outputs a Yes / No answer

Authentication by Asymmetric Encryption



- A encrypts message M and some control data using algorithm ENC and private key Prv(A) to obtain cipher C
- B decrypts cipher C using algorithm DEC with public key Pub(A) to obtain message M and control data
- B checks control data and outputs a Yes / No answer

Authentication by Encryption

- "Symmetric Encryption" and "Asymmetric Encryption" provide source authenticity and message integrity
- Both methods provide also secrecy of the messages
- Recipient can check whether meaningful message and control data are obtained after decryption
- Changes to cipher are likely to result in meaningless message and control data

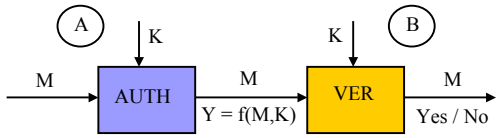
Authentication by Encryption – Requirements

- Valid messages must be "sparse" in the domain of all possible messages. (Otherwise, it will be easy to generate a cipher that will decrypt to a valid message.)
- Valid messages must be "distinguishable" from non-valid messages. (Otherwise, any random cipher will decrypt to a valid message.)
- Valid messages must be "automatically" distinguishable from non-valid messages. This can be done by adding some control data and message formats.
- Control data may contain tags, time stamps, sequence numbers, format fields, etc.

Authentication by Encryption – Properties

- Achieves secrecy, authenticity, and integrity in one operation
- Requires automatic distinguishability between valid messages and non-valid messages
- Requires full decryption of the message in order to determine authenticity and integrity
- Requires heavy calculations of the encryption functions (which may be too costly if secrecy is not needed)
- The authenticator is as long as the encrypted message (which may be problematic for large messages)

Authentication by MAC



- A computes a Message Authentication Code (MAC) for message M using algorithm AUTH with symmetric key K to obtain the code Y
- B uses algorithm VER with symmetric key K to verify that code Y is indeed the MAC of message M
- B outputs a Yes / No answer

One-Time MAC – Reminder

- Large prime P
- Message M with numeric value between 1 and P-1
- Key K is a pair of random numbers, a and b, such that: $0 < a, b < P$
- Key K is used only once
- Message Authentication Code (MAC) is:

$$Y = f(M, K) = a \cdot M + b \pmod{P}$$

- MAC is computed by the sending party (who knows K)
- MAC is verified by the receiving party (who knows K)

One-Time-MAC – Practicality Issues

- One-Time MAC (OTM) is 100% secure, with a mathematical proof to back this claim
- Use of OTM is limited due to the need to negotiate a new, long, random one-time key for every authentication
- OTM is the ultimate authentication scheme, and it can provide a reference point for authentication
- OTM may be used in highly-critical environments, where key negotiation and distribution can be handled using off-line channels

Authentication by MAC

- MAC should be a fixed-size code that is appended to the message; typical sizes of MAC range from 64 to 256 bits
- Message can be sent in the clear without encryption
- No need to add control data to the message in order to protect it
- MAC is a function of the message and a secret key
- MAC should not be reversible
- The strength of the MAC depends on the function and on the secrecy of the key

Authentication by MAC – Properties

- MAC can be computed quickly by both the sender and the receiver
- MAC need not be computed by the receiver on all the received messages; it may be computed only on some selected messages
- MAC need not be computed by the receiver when the message is received; it may be computed later
- MAC is much smaller than the message; it can be stored for later reference and use
- Knowledge of the MAC does not endanger the message

Authentication by MAC – Requirements

- All the possible MAC's should be equally probable; that is, the appearance probability of a particular n-bit MAC should be 2^{-n}
- The probability that the n-bit MAC's of two different messages M_1 and M_2 will be the same should be 2^{-n}
- Given a message M and its MAC Y, it should infeasible for an attacker (who doesn't know the key K) to find another message M' with the same MAC Y
- Having seen the MAC's $Y_1, Y_2, \ldots, Y_L$ of some (known or chosen) messages $M_1, M_2, \ldots, M_L$, it should be infeasible for an attacker to find the MAC of a new message M'

Authentication by MAC – Bad Example

- Divide message M into L blocks of n bits each: $M = M_1, M_2, \ldots, M_L$
- Let K be a key of the encryption algorithm E
- Let $X(M) = M_1 \oplus M_2 \oplus \ldots \oplus M_L$ (that is, the bit-wise XOR of the L blocks)
- Let $MAC_K(M) = E_K(X(M))$ (that is, the encryption with algorithm E and key K of the result of the bit-wise XOR of the L blocks of M)
- It is easy to forge MAC's with this scheme (change order of blocks, change corresponding bits in two blocks, etc.)

CBC-MAC Authentication

- Divide message M into L blocks of size n bits each: $M = M_1, M_2, \ldots, M_L$
- Let K be a key of the encryption algorithm E
- Let $C_0 = IV$ be a random block of n bits
- Let $C_i = E_K(M_i \oplus C_{i-1})$ for $i = 1, 2, \ldots, L$ (that is, the $L+1$ blocks $C_0, C_1, \ldots, C_L$ are the CBC encryption with algorithm E and key K of message M)
- Let $MAC_K(M) = (C_0, C_L) = (IV, \text{CBC-MAC-}E_K(M))$: that is, the first and last blocks of CBC encryption

CBC-MAC Authentication – Properties

- CBC-MAC computations are simple and efficient (depending on the encryption algorithm)
- The CBC-MAC depends on all the L blocks of the message in a similar manner
- The CBC-MAC is uniformly distributed over the 2^n output blocks of n-bits
- The CBC-MAC is hard to forge given only the random IV and the last block C_L

CBC – Encryption and Authentication

- Question: What about using CBC both for encrypting a message and for authentication it?
  If $M = M_1, M_2, \ldots, M_L$ and $C_i = E_K(M_i \oplus C_{i-1})$, then use $C = IV, C_1, C_2, \ldots, C_L$ as the encryption of M, and use the last block C_L as the MAC of M
- Answer: Not good if encryption and authentication use same key K – attacker can truncate the message to get $M' = M_1, M_2, \ldots, M_{L-1}$ and send $C' = IV, C_1, C_2, \ldots, C_{L-1}$ as the cipher and send (IV, C_{L-1}) as the MAC
- Answer: Good if encryption and authentication use two independent keys K and K' – compute $IV, C_1, C_2, \ldots, C_L$ with key K and compute MAC (IV', C'_L) with key K'

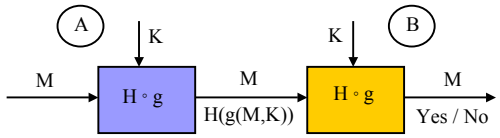
CBC-MAC Authentication – Issues

- Problem: Consider a 1-block message $M = M_1$ that is fed into a CBC-MAC scheme with a key K and an arbitrary IV. The output will be $MAC_K(M) = (IV, C_1)$. An attacker can create the following 2-block message $M' = M_1, M_2$, where the new block M_2 is defined by $M_2 = IV \oplus M_1 \oplus C_1$. Then, we get $MAC_K(M) = MAC_K(M') = (IV, C_1)$.
- Generalized Problem: Let message $M = M_1, M_2, \ldots, M_L$ and let message $M' = M'_1, M'_2, \ldots, M'_T$. An attacker can "combine" messages M and M' into a new message for which the attacker will have the MAC. (Specific details of the construction are left as exercise.)

CBC-MAC Authentication – Solutions

- Solution: Include the length L in blocks of the message M in the MAC computation – $MAC_K(\text{"L,M"}) = (IV, C^*)$. This will not allow "extending" or "combining" messages to form new and longer messages.
- Bad Solution: C^* is last block of the CBC-MAC of the new message $M \| L$ (that is, $L+1$ blocks consisting of the L blocks of M and one more block containing the value L).
- Good Solution: C^* is last block of the CBC-MAC of the new message $L \| M$ (that is, $L+1$ blocks consisting of one block containing the value L followed by the L blocks of M).

Authentication by Symmetric Hash



- A computes a hash $H()$ on the value of some function $g()$ of the message M and of the symmetric key K
- B also computes the hash $H()$ on the value of function $g()$ of the message M and of the symmetric key K.
- B compares the resultant hash value with the received hash value, and outputs a Yes / No answer

Authentication by Hash – Properties

- Construction is similar to "Authentication by MAC"
- Hash functions are more efficient than MAC functions
- Hash functions do not require the use of encryption functions
- Hash functions are standard and are freely available
- Hash functions are not reversible (by definition)
- The strength of the MAC depends on the hash function and on the secrecy of the key

Hash Functions – Requirements

- Operability:
  - $H()$ should work on any input length
  - $H()$ should produce output of fixed size
  - $H()$ should be easy to compute
- Security:
  - One-way – given value y it is hard to find value X such that $Y = H(X)$
  - Weak Collision Resistance – given value X_1 it is hard to find another value X_2 such that $H(X_1) = H(X_2)$
  - Strong Collision Resistance – it is hard to find two different values X_1 and X_2 such that $H(X_1) = H(X_2)$

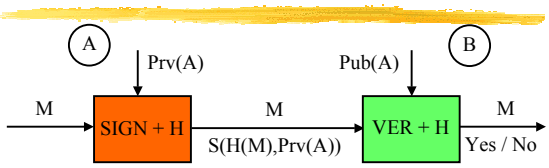
Authentication by Hash – Constructions

- Hash functions $H()$ are functions of one argument
- Most hash functions $H()$ use some chaining mode of the blocks of their argument
- Several alternatives for $H_\circ g$:
  - $H_\circ g(M,K) = H(K \| M)$ – may not be good because K is only involved at the beginning of the message M
  - $H_\circ g(M,K) = H(M \| K)$ – may not be good because K is only involved at the end of the message M
  - $H_\circ g(M,K) = H(K \| M \| K)$ – seems like a good scheme because K wraps the message M

Authentication by Hash – HMAC

- $HMAC_K(M)$ is a scheme to create a MAC of a message M using a hash function $H()$ and a secret key K
- $K^+ = K$ padded with zeros to the left
- ipad = 00110110 repeated to fill block
- opad = 01011010 repeated to fill block
- $HMAC_K(M) = H((K^+ \oplus \text{opad}) \| H((K^+ \oplus \text{ipad}) \| M))$
  - Pad K and XOR with ipad to produce S_i (first "random" key)
  - Append M to S_i and apply $H()$ to them
  - Pad K and XOR with opad to produce S_o (second "random" key)
  - Append result of first hash to S_o and apply $H()$

Authentication by Asymmetric Signature



- A computes a hash function $H()$ of the message M. A signs the hash value with the asymmetric key $Prv(A)$.
- B also computes a hash function $H()$ of the message M. B verifies the signature of A on $H(M)$ using the asymmetric key $Pub(A)$.
- B outputs a Yes / No answer.

Authentication by Signature – Properties

- Hash function reduces the size of the message from an arbitrary length to a fixed length (usually between 128 and 256 bits)
- The signature function operates on a fixed-length input
- Hash functions are fast and efficient on all texts
- Signature functions are slow and inefficient for long texts
- Strength of the scheme depends on the hash function, on the signature function, and on the secrecy of the key
- Scheme also provides non-repudiation

Signature Functions – Requirements

- Hash function $H()$ should be:
  - One-way
  - Weak Collision Resistance
  - Strong Collision Resistance
- Signature function and verification function should be efficient
- Signature should be difficult to forge
- Public key should be known to all parties wishing to authenticate the source / message
- Private key should be kept in strict confidence only at the source

Hash Functions

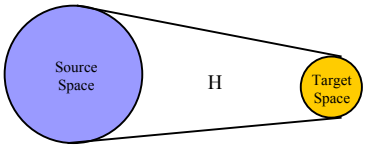
Prof. Shlomo Kipnis
November 21, 2007

Hash Functions

- Hash function generates a small identifier of a large object (number, message, file, document, etc.)
- Identifier is used to identify / authenticate / hide / scramble / sign / verify the original object
- Common Names:
  - Hash Value
  - Message Digest
  - Fingerprint
  - Checksum
- Notice: Hash function does not use a key

Cryptographic Hash Functions

- Hash function maps from the very large message space to the relatively small hash values space
- It should be difficult to reverse the hash function
- It should be difficult to find collisions of the hash function



Crypto Hash Functions – Uses

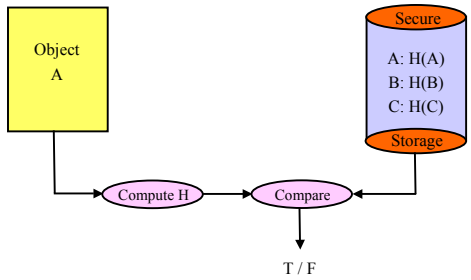
- Object identifiers
- Authentication Protocols
- Commitment Protocols
- Symmetric Encryption
- Symmetric MAC's
- Asymmetric Signatures

Object Identifiers with Hash (I)

Object Digest

- Computing hash of files / documents / emails
- Keeping the hash values in a secure place
- Files / documents / emails may reside in non-secure places
- Before using an object, the system computes a hash of the object and compares it with the stored hash
- If the hash values do not match – the object was probably changed in an unauthorized manner

Object Identifiers with Hash (II)



Authentication Protocols with Hash (I)

Goal: A wishes to identify and authenticate himself to B

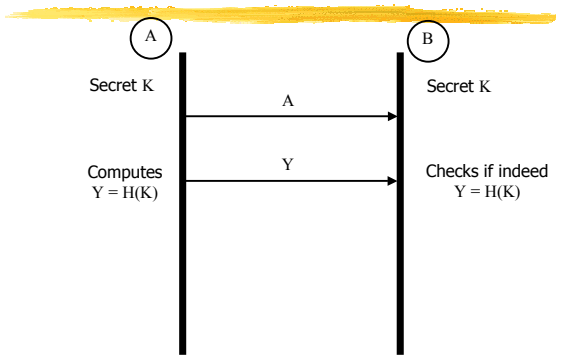
Infrastructure: A and B share a long-lived secret key K

Naive Authentication Protocol

- A identifies himself to B
- A sends to B hash of secret key K
- B verifies hash of secret key K

Problem: Above protocol is subject to a "replay attack"

Authentication Protocols with Hash (II)



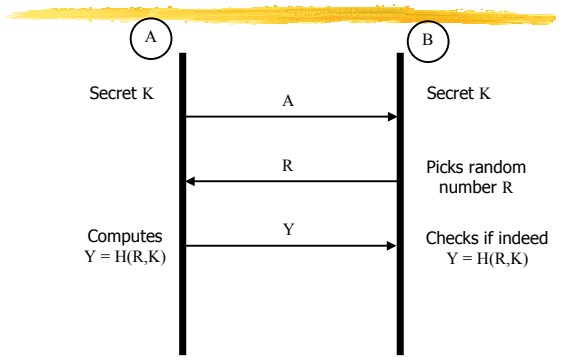
Authentication Protocols with Hash (III)

Modified Authentication Protocol

- A identifies himself to B
- B sends to A a random number R
- A sends to B hash of random number R and secret key K
- B verifies hash of random number R and secret key K

Solution: This protocol is sound against sniffing and replay

Authentication Protocols with Hash (IV)



Commitment Protocols with Hash (I)

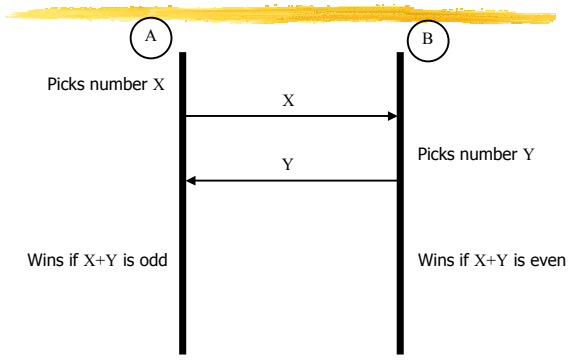
Goal: A and B wish to play "odd or even" over the network

Naive Commitment Protocol

- A picks a number X and sends it to B
- B picks a number Y and sends it to A
- A wins if X+Y is odd
- B wins if X+Y is even

Problem: How can we guarantee that B doesn't cheat?

Commitment Protocols with Hash (II)



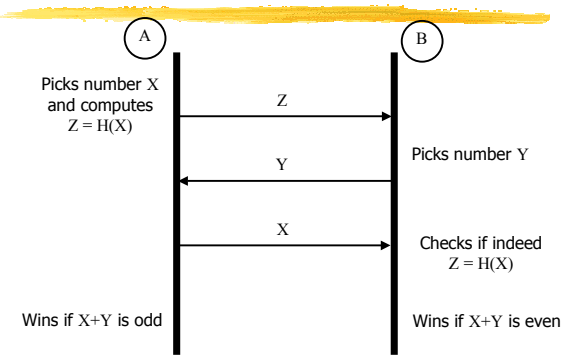
Commitment Protocols with Hash (III)

Modified Commitment Protocol

- A picks a number X and sends value of Z = H(X) to B
- B picks a number Y and sends value of Y to A
- A now sends value of X to B
- B checks if X complies with Z that was sent before
- A wins if X+Y is odd
- B wins if X+Y is even

Solution: In this protocol B cannot cheat

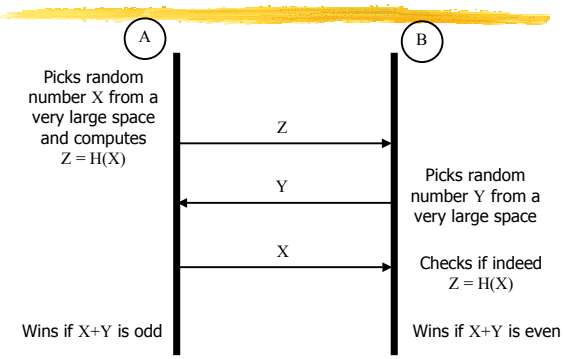
Commitment Protocols with Hash (IV)



Commitment Protocols with Hash (V)

- Hash function does two things in the protocol:
  - Hides the number X from B at the beginning of the game
  - Makes A commit to the number X until the end of the game
- Question: What if A always picks small numbers so that B can make a list of all the hash values?
- Answer: A should select random values for the protocol:
  - Select the number X from a very large space of numbers
  - Mask the number X with a random noise from a very large space

Commitment Protocols with Hash (VI)



Encryption with Hash (I)

Goal: A and B wish to create a "Pseudo-One-Time-Pad"

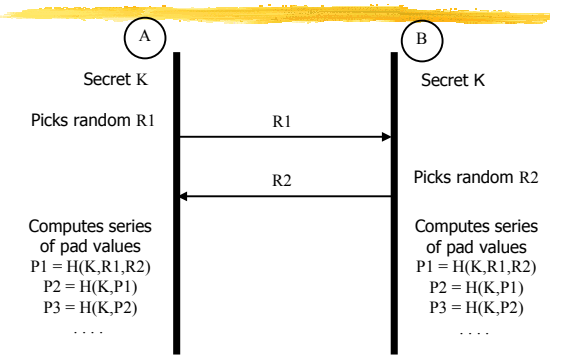
Infrastructure: A and B share a long-lived secret key K

Generation of Pseudo-One-Time-Pad

- A and B exchange random numbers
- A and B compute a series of hash values of the random numbers and the secret key to generate encryption pads

Parties can "encrypt" without an encryption function

Encryption with Hash (II)



MAC's with Hash

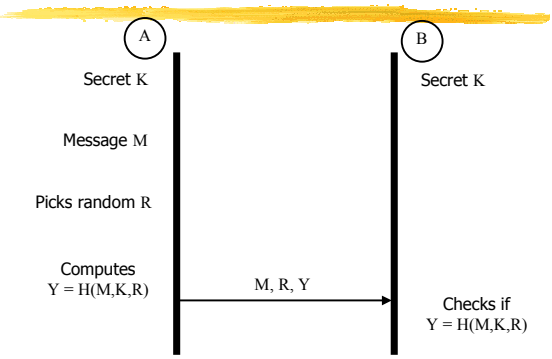
Goal: A and B wish to create MAC's of messages

Infrastructure: A and B share a long-lived secret key K

Message Authentication Code

- A creates a random R
- A computes the MAC of a message M as the hash of message M, secret key K, and random R
- A sends message M, random R, and value of MAC
- B compares MAC with the expected value

MAC's with Hash (II)



Signatures with Hash (I)

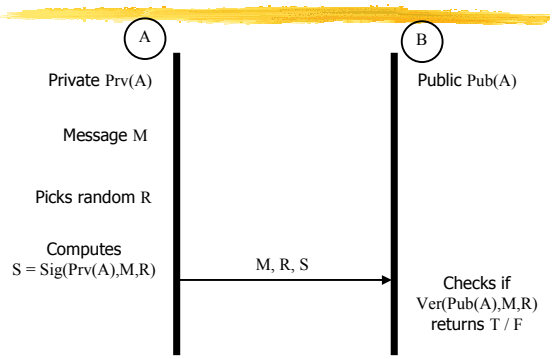
Goal: A wishes to sign messages for B

Infrastructure: A has private Prv(A) and B has public Pub(A)

Signature

- A creates a random R
- A signs message M by applying the signature function (with Prv(A)) to the hash of message M and random R
- A sends message M, random R, and value of signature
- B verifies signature with Pub(A)

Signatures with Hash (II)



Crypto Hash Functions – Requirements

- Operability:
  - H() should work on any input length
  - H() should produce output of fixed size
  - H() should be easy to compute
- Security:
  - One-way – given value y it is hard to find value X such that $Y = H(X)$
  - Weak Collision Resistance – given value X_1 it is hard to find another value X_2 such that $H(X_1) = H(X_2)$
  - Strong Collision Resistance – it is hard to find two different values X_1 and X_2 such that $H(X_1) = H(X_2)$

Hash Function – Bad Example I

- Divide message M into L blocks of n bits each: $M = M_1, M_2, \ldots, M_L$
- Let E be an encryption algorithm, let K be a known key for algorithm E (such as K=0), and let IV be a known initial vector for algorithm E (such as IV=0)
- Let H(M) be the last block of $CBC\text{-}E_K(M)$ with the known K and the known IV
- Problem: H is easy to inverse – since given some output value Y, one can compute a message $X = D_K(Y) \oplus IV$

Exhaustive Search Attack

- Values produced by a good hash function should be randomly spread over the target space
- If hash function H produces an output of length n bits, then collisions will be found after searching $O(2^n)$ input values
- Consequently – n should be large enough to reduce the risk of finding collisions – $n \geq 100$ seems good enough

Hash Function – Bad Example II

- Divide message M into L blocks of 56 bits each: $M = M_1, M_2, \ldots, M_L$
- Use the DES encryption algorithm as follows:
  - Let P be a known text block (such as $P=0$)
  - Let $C_0=IV$ be a known initial vector (such as $IV=0$)
  - Let C_i be the value of $DES_{M_i}(P \oplus C_{i-1})$ for $i = 1, 2, \ldots, L$
  - Define $H(M) = C_L$ – the last block of the encryption
- Problem: Given message $M = M_1, M_2, \ldots, M_L$ it is easy to find another message $M' = M'_1, M'_2, \ldots, M'_T$ that will be mapped to same C_L by checking $O(2^{64})$ options for M'

Birthday Paradox

- Assume a target space Y with N possible outcomes
- Assume a function F that maps input values from a source space X to the target space Y with a uniform distribution
- Then, if R input values from the source space X are picked at random, such that $R \geq 1.18\sqrt{N}$, then the probability that two inputs map to the same output value will be greater than ½
- Example: if there are over 23 people in a room, then the probability is greater than ½ that two people will have the same birthday (out of 365 days in the year)

Birthday Paradox – Proof

- Let $Q(N,T)$ be the probability that T independently selected inputs from the source space X still do not create any collision in the target space Y
- $$Q(N,T) = 1 \cdot (1 - 1/N) \cdot (1 - 2/N) \cdot \ldots \cdot (1 - (T-1)/N)$$
  $$\leq e^{-1/N} \cdot e^{-2/N} \cdot \ldots e^{-(T-1)/N}$$
  $$= e^{-(T-1)T/2N}$$
- Now, for values of $T \geq 1.18\sqrt{N}$, we get $Q(N,T) < ½$, which means that the probability that a collision will occur is greater than ½.

Birthday Attack

- Hash function H produces output of length n bits
- Compose two documents – good doc and bad doc
  - Generate about $\sqrt{2^n} = 2^{(n/2)}$ semantically-identical variations of good doc
  - Generate about $\sqrt{2^n} = 2^{(n/2)}$ semantically-identical variations of bad doc
- With probability at least ½ – there will be a pair of good doc and bad doc with the same hash value
  - Signing the hash of good doc also signs bad doc
- To escape the birthday attack – should select $n \geq 150$

Hash Function – Bad Example III

- Divide message M into L blocks of 128 bits each: $M = M_1, M_2, \ldots, M_L$
- Use the AES encryption algorithm as follows:
  - Let H_0 be a known 128-bit block (such as $H_0=0$)
  - Let $H_i = AES_{M_i}(H_{i-1})$ for $i = 1, 2, \ldots, L$
  - Define $H(M) = H_L$ – the last block of this process
- Birthday Attack:
  - Take message $M = M_1, M_2, \ldots, M_L$ and compute $H(M)$
  - Take message $M' = M_1, M_2, \ldots, M_{L-2}$ and compute $H(M')$
  - Check $O(2^{64})$ options for M_{L-1} and compute $AES_{M_{L-1}}(H_{L-2})$
  - Check $O(2^{64})$ options for M_L and compute $AES^{-1}_{M_L}(H_L)$
  - Collisions will be found with probability at least ½

Practical Cryptographic Hash Functions

- MD5 – generates outputs of 128 bits
- SHA-1 – generates outputs of 160 bits
- RIPMED – generates outputs of 160 bits
- HMAC – a standard scheme for generating safe MAC's with a given hash function H

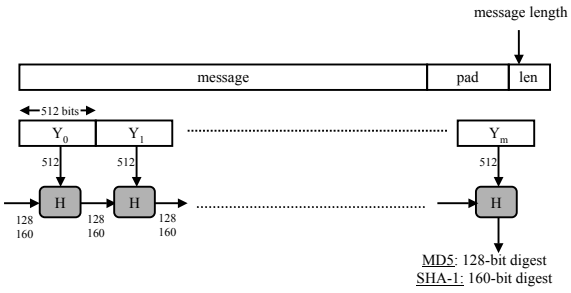
Description of MD5

- Works on blocks of 512 bits
- Produces hash of length 128 bits
- Input is padded to length 448 mod 512; and original input length is written in last 64 bits
- Each input block is processed by the compression function – H_{MD5} – that works in four rounds:
  - use 128-bit state buffer ABCD, with initial known values
  - use lookup table, produced by "sine" function, to map inputs
  - each round uses a different primitive logical function
  - at end of last round – add result to current value of ABCD

Description of SHA-1

- Works on blocks of 512 bits
- Produces hash of length 160 bits
- Input is padded to length 448 mod 512; and original input length is written in last 64 bits
- Each input block is processed by the compression function – H_{SHA} – that works in four rounds:
  - uses five-word state buffer ABCDE with initial known values, producing 160-bit hash
- More resilient to brute-force attacks than MD5
- Less known vulnerabilities than MD5

Schematics of MD5 & SHA-1



Description of HMAC

- $HMAC_K(M)$ is a scheme to create a MAC of a message M using a hash function H() and a secret key K
- K^+ = K padded with zeros to the left
- ipad = 00110110 repeated to fill block
- opad = 01011010 repeated to fill block
- $HMAC_K(M) = H((K^+ \oplus opad) \| H((K^+ \oplus ipad) \| M))$
  - Pad K and XOR with ipad to produce S_i (first "random" key)
  - Append M to S_i and apply H() to them
  - Pad K and XOR with opad to produce S_o (second "random" key)
  - Append result of first hash to S_o and apply H()

HMAC Claims

- There are formal proofs regarding HMAC security:
  - $HMAC_K()$ with function H() is as secure as finding collisions of the function H() when the key K is unknown and the initial value IV is random
  - $HMAC_K()$ with function H() is as secure as computing the output of the compression function H() when the key K is unknown and the initial value IV is random
  - $HMAC_K()$ with function H() is not subject to birthday attacks on the key K since, in effect, two different keys K_I and K_O are used

Algebra and Number Theory

Prof. Shlomo Kipnis
November 26, 2007

Outline

- Division
- Common Divisors
- Euclid's GCD Algorithm
- Modular Arithmetic
- Groups and Subgroups
- Solving Modular Equations
- Chinese Remainders
- Modular Powers
- Fields
- Polynomials
- Galois Fields

Division (I)

Definition: The set of integers is:

$$\mathbf{Z} = \{ \ldots, -2, -1, 0, 1, 2, \ldots \}$$

Definition: The set of natural numbers is:

$$\mathbf{N} = \{ 0, 1, 2, \ldots \}$$

Definition: Let d and a be integers. We say that d divides a if there exists an integer k such that $a = kd$. In this case, we also say that d is a divisor of a and that a is a multiple of d. The notation $d \mid a$ is used to denote that d divides a.

Convention: Every integer divides 0.

Division (II)

Observation: If $d \mid a$, then so does $(-d) \mid a$.

Convention: In writing $d \mid a$, we assume that d is positive.

Observation: A divisor of a is at least 1 and at most $|a|$.

Example: The divisors of 24 are $1, 2, 3, 4, 6, 8, 12,$ and 24.

Observation: Every integer a is divisible by the two trivial divisors 1 and $|a|$.

Definition: Non-trivial divisors of a are called factors of a.

Example: The factors of 20 are $2, 4, 5,$ and 10.

Division (III)

Definition: An integer $p \geq 2$ is called a prime if it is divisible only by 1 and by itself (that is, it has no factors).

Theorem: There are infinitely many primes.

Definition: An integer $a > 1$ that is not a prime is said to be a composite number.

Convention: Integer 1 is said to be a unit (neither a prime nor a composite).

Convention: The negative integers and 0 are considered neither prime nor composite.

Division (IV)

Division Theorem: For any integer a and for any positive integer n, there are two unique integers q and r, such that $0 \leq r < n$ and $a = qn + r$.

Notation: The value q is called the quotient of the division.

Notation: The value r is called the residue or remainder of the division, and we write $r = a \bmod n$.

Definition: For two integers a and b, and for any positive integer n, if $n \mid (a-b)$ (that is, $a \bmod n = b \bmod n$, which means that a and b have the same residue modulo n), then we say that a is congruent to b modulo n, and we write $a \equiv b \pmod{n}$.

Division (V)

Definition: For any positive integer n, the integers can be divided into n equivalence classes according to their residues modulo n:

$$[a]_n = \{ a + kn : k \in \mathbf{Z} \}$$

Definition: The collection of all the n equivalence classes of the residues modulo n is called Z_n:

$$\mathbf{Z}_n = \{ [a]_n : 0 \le a \le n-1 \}$$

Observation: If we represent each $[a]_n$ by its least nonnegative element, then we get:

$$\mathbf{Z}_n = \{ 0, 1, \ldots, n-1 \}$$

Common Divisors (I)

Definition: If d is a divisor of a and d is also a divisor of b, then d is called a common divisor of a and b.

Example: All the common divisors of 24 and 30 are $1, 2, 3,$ and 6.

Observation: If $d \mid a$ and $d \mid b$, then $d \mid (a+b)$ and $d \mid (a-b)$.

Observation: If $d \mid a$ and $d \mid b$, then $d \mid (ax+by)$ for any integers x and y.

Observation: If $a \mid b$, then either $|a| \le |b|$ or $b=0$.

Observation: If $a \mid b$ and $b \mid a$, then $|a| = |b|$.

Common Divisors (II)

Definition: The greatest common divisor of two integers a and b, not both zero, is the largest of the common divisors of a and b, and it is denoted by $\gcd(a, b)$.

Examples: $\gcd(24, 30) = 6$
$\gcd(15, 7) = 1$
$\gcd(0, 9) = 9$

Observation: If a and b are both not zero, then $\gcd(a, b)$ is an integer between 1 and $\min(|a|, |b|)$.

Definition: $\gcd(0, 0) = 0$.

Common Divisors (III)

Observations: $\gcd(a, b) = \gcd(b, a)$
$\gcd(a, b) = \gcd(-a, b)$
$\gcd(a, b) = \gcd(|a|, |b|)$
$\gcd(a, 0) = |a|$
$\gcd(a, ka) = |a|$ for any $k \in \mathbf{Z}$

GCD Characterization Theorem: If a and b are any integers, not both zero, then $\gcd(a, b)$ is the smallest positive element of the set $LC(a, b) = \{ ax+by : x, y \in \mathbf{Z} \}$ of linear combinations of a and b.

Common Divisors (IV)

Proof of GCD Characterization Theorem:

- Let s be the smallest positive element of the set $LC(a, b)$.
- Let $s = ax+by$ for some $x, y \in \mathbf{Z}$.
- Let the unique residual representation of a modulo s be $a = qs + r$.
- Then, $a \bmod s = r$
  $= a - qs$
  $= a - q(ax+by)$
  $= a(1 - qx) + b(-qy)$,
  so $-(a \bmod s)$ is also in the set $LC(a, b)$.
- However, since $(a \bmod s) < s$, we must have $(a \bmod s) = 0$.
- Therefore – $s \mid a$.
- Similarly – $s \mid b$.

Common Divisors (V)

Proof of GCD Characterization Theorem (continued):

- Therefore, s is a common divisor of a and b.
- Therefore, we must have $s \le \gcd(a, b)$.
- However, in general – since $\gcd(a, b) \mid a$ and $\gcd(a, b) \mid b$ – we must have that $\gcd(a, b) \mid s$ (since s is a linear combination of a and b).
- Now, if $\gcd(a, b) \mid s$ and $s > 0$, then we must have that $\gcd(a, b) \le s$.
- We now have $s \le \gcd(a, b)$ and $\gcd(a, b) \le s$.
- This implies that $\gcd(a, b) = s$.
- Therefore, the greatest common divisor of a and b is the smallest positive linear combination of a and b.
- QED.

Common Divisors (VI)

Corollary: For any integers a and b, if $d \mid a$ and $d \mid b$, then $d \mid \gcd(a, b)$.

Proof:

- According to the GCD Characterization Theorem, $\gcd(a, b)$ is a linear combination of a and b. And – any common divisor d of a and b must also divide every linear combination of a and b.

Corollary: For any integers a and b and any nonnegative integer n, we have $\gcd(an, bn) = n \gcd(a, b)$.

Proof:

- If $n = 0$, the corollary is immediately true. If $n > 0$, then $\gcd(an, bn)$ is the smallest positive element of the set $\{anx+bny\}$, which is n times the smallest positive element of the set $\{ax+by\}$.

Common Divisors (VII)

Fundamental Theorem of Arithmetic: For all positive integers d, a, and b, if $d \mid ab$ and $\gcd(a, d) = 1$, then $d \mid b$.

Proof:

- Since $d \mid ab$, there exists integer k such that $ab = kd$.
- Since $\gcd(a, d) = 1$, there exist integers x and y such that $ax + dy = 1$.
- By manipulating, we get $b = (kx + yb)d$, which implies that $d \mid b$.

Definition: Two integers a and b are relatively prime if their only common divisor is 1, that is, if $\gcd(a, b) = 1$.

Examples: $\gcd(8, 15) = 1$.

Common Divisors (VII)

Theorem: For any integers a, b, and p, if both $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$, then $\gcd(ab, p) = 1$.

Proof:

- By the GCD Characterization Theorem, there exist x, x', y, y', such that $ax + py = 1$ and $bx' + py' = 1$.
- By multiplying, we get $ab(xx') + p(ybx'+y'ax+pyy') = 1$.
- This linear combination of ab and p implies that $\gcd(ab, p) = 1$.

Definition: Integers $n_1, n_2, \ldots, n_k$ are said to be pairwise relatively prime if, for all $i \neq j$, we have $\gcd(n_i, n_j) = 1$.

Common Divisors (VIII)

Theorem: For all primes p and all integers a and b, if $p \mid ab$, then $p \mid a$ or $p \mid b$ (or both).

Proof:

- Assume that p divides ab, but p doesn't divide a and p doesn't divide b.
- Since p is prime, its only divisors are 1 and p.
- Thus, $\gcd(a, p) = 1$ and $\gcd(b, p) = 1$. And, by previous theorem, $\gcd(ab, p) = 1$, which contradicts the assumption that p divides ab.

Unique Factorization Theorem: A composite a can be written in exactly one way as a product of the form:

$$a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$$

where $p_1 < p_2 < \cdots < p_r$ are primes, and e_i are positive integers.

Euclid's GCD Algorithm (I)

Observation: Discussion can be restricted to nonnegative integers (since $\gcd(a, b) = \gcd(|a|, |b|)$).

Observation: If $a = p_1^{e_1} p_2^{e_2} \cdots p_r^{e_r}$ and $b = p_1^{f_1} p_2^{f_2} \cdots p_r^{f_r}$, where $p_1 < p_2 < \cdots < p_r$ are prime, and e_i and f_i are nonnegative, then $\gcd(a, b) = p_1^{\min(e_1, f_1)} p_2^{\min(e_2, f_2)} \cdots p_r^{\min(e_r, f_r)}$.

Proof:

- Left as exercise.

Problem: Above technique requires factoring the numbers a and b, which is a hard problem by itself.

Euclid's GCD Algorithm (II)

GCD Recursion Theorem: For any nonnegative integer a and positive integer b, we have $\gcd(a, b) = \gcd(b, (a \bmod b))$.

Proof:

- We show that $d = \gcd(a, b)$ and $e = \gcd(b, (a \bmod b))$ divide one another.
- Let $d = \gcd(a, b)$, then $d \mid a$ and $d \mid b$.
- Now, $(a \bmod b) = a - qb$, for some q, which is a linear combination of a and b. This implies that $d \mid (a \bmod b)$.
- Since $d \mid b$ and $d \mid (a \bmod b)$, it follows that $d \mid \gcd(b, (a \bmod b))$.
- Let $e = \gcd(b, (a \bmod b))$, then $e \mid b$ and $e \mid (a \bmod b)$.
- Now, $a = qb + (a \bmod b)$, for some q, which is a linear combination of b and $(a \bmod b)$. This implies that $e \mid a$.
- Since $e \mid a$ and $e \mid b$, it follows that $e \mid \gcd(a, b)$.
- Since $d \mid e$ and $e \mid d$, we have $\gcd(a, b) = d = e = \gcd(b, (a \bmod b))$.

Euclid's GCD Algorithm (III)

Euclid's Algorithm:

```
Euclid(a, b)
1   if (b=0)
2       then return a
3       else return Euclid(b, (a mod b))
```

Example: Euclid(30, 21) =
Euclid(21,9) =
Euclid(9, 3) =
Euclid(3, 0) =
3

Euclid's GCD Algorithm (IV)

Running Time: Worst case for Euclid's algorithm appears when the numbers a and b are Fibonacci numbers. In this case, the running time of Euclid's algorithm is logarithmic (in base $\Phi = (1+\sqrt{5})/2$ – the Golden Ratio) in the size of a or b. For other numbers, the running time is even lower.

Lemma: If $a > b \geq 0$ and the invocation Euclid(a, b) makes $k \geq 1$ recursive calls, then $a \geq F_{k+2}$ and $b \geq F_{k+1}$.

Proof: (by induction on k)

Lamé's Theorem: For any integer $k \geq 1$, if $a > b \geq 0$ and $b < F_{k+1}$, then the call to Euclid(a, b) makes less than k calls.

Euclid's GCD Algorithm (V)

Extended Euclid's Algorithm: Euclid's algorithm can be extended to compute the coefficients x and y, such that $d = \gcd(a, b) = ax + by$.

```
Extended-Euclid(a, b)
1   if (b=0)
2       then return (a,1, 0)
3   (d', x', y') ← Extended-Euclid(b, (a mod b))
4   (d, x, y) ← (d', y', x' – ⌊a/b⌋ y')
5   return (d, x, y)
```


Euclid's GCD Algorithm (VI)

Extended Euclid's Algorithm Example:

a	b	⌊a/b⌋	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	-	3	1	0

Euclid's GCD Algorithm (VII)

Extended Euclid's Algorithm Explained:

- Line 1 of Extended-Euclid is identical to line 1 of Euclid (checking whether $b = 0$ to end the recursion)
- If $b = 0$, then Line 2 of Extended-Euclid returns the following values (which solve the linear combination $\gcd(a, b) = ax + by$):
  - a as the value of the $\gcd(a, b)$,
  - 1 as the value of the coefficient x, and
  - 0 as the value of the coefficient y
- If $b \neq 0$, then, in Line 3, Extended-Euclid first computes the values of (d', x', y') such that:
  - $d' = \gcd(b, (a \bmod b))$ and
  - $d' = bx' + (a \bmod b)y'$.

Euclid's GCD Algorithm (VIII)

Extended Euclid's Algorithm Explained (continued):

- In line 4, after returning from the recursive call, Extended-Euclid sets:
  - $d = \gcd(a, b) = \gcd(b, (a \bmod b)) = d'$
  - To obtain x and y such that $d = ax + by$, we notice that
    $d = d' = bx' + (a \bmod b)y'$
    $= bx' + (a - \lfloor a/b \rfloor b)y'$
    $= ay' + b(x' - \lfloor a/b \rfloor y')$
  - Now, the selection of $x = y'$ and $y = x' - \lfloor a/b \rfloor y'$, satisfies the equation $d = ax + by$.
- Line 5 of Extended-Euclid returns the values computed for d, x, and y.
- The running time of Extended-Euclid is the same order of magnitude as that of Euclid.

Modular Arithmetic (I)

Observation: For any $n \geq 2$, modular arithmetic can be described as working with the numbers $\{0, 1, \ldots, n-1\}$ with operations (such as addition, subtraction, multiplication, and division) modulo n.

Definition: Addition modulo n is denoted by $+_n$ and is defined by $[a]_n +_n [b]_n = [a + b]_n$. We shall also write $(a \bmod n) + (b \bmod n) = (a + b) \pmod n$.

Observation: Addition modulo n has an inverse operation, called subtraction modulo n.

Modular Arithmetic (II)

Definition: Subtraction modulo n is denoted by $-_n$ and is defined by $[a]_n -_n [b]_n = [a - b]_n$. We shall also write $(a \bmod n) - (b \bmod n) = (a - b) \pmod n$.

Definition: The additive inverse of element $(a \bmod n)$ is the element $((n - a) \bmod n)$. We denote the additive inverse of a by $-a$.

Examples:

$(8 + 3) \pmod 7 = (8 \bmod 7) + (3 \bmod 7) = 4$

$(15 - 22) \pmod{10} = (15 \bmod 10) - (22 \bmod 10) = 3$

$(-3 \bmod 8) = 5$

Modular Arithmetic (III)

Definition: Multiplication modulo n is denoted by $\bullet_n$ and is defined by $[a]_n \bullet_n [b]_n = [a \bullet b]_n$. We shall also write $(a \bmod n) \bullet (b \bmod n) = (a \bullet b) \pmod n$.

Observation: Multiplication modulo n does not necessarily have an inverse operation (that is – division modulo n).

Observation: The inverse of an element modulo n does not necessarily exist for all the elements.

Modular Arithmetic (IV)

Definition: Multiplicative inverse of an element $(a \bmod n)$, if exists, is an element $(b \bmod n)$, such that $(a \bullet b) \pmod n = 1$. The multiplicative inverse of a, if exists, is denoted by a^{-1}.

Definition: Modular division, if exists, is denoted by $/_n$ and is defined by $[a]_n /_n [b]_n = [a]_n \bullet_n [b^{-1}]_n$. In such cases, we shall also write $(a \bmod n) / (b \bmod n) = (a / b) \pmod n$.

Examples:

$26 = 14^{-1} \pmod{11}$ since

$26 \bullet 14 \pmod{11} = (26 \bmod 11) \bullet (14 \bmod 11) = 1$

$(8 / 4) \pmod{10}$ is not defined since

4 is not invertible mod 10

Groups and Subgroups (I)

Definition: A group $G = (S, \oplus)$ is a set S with a binary operation $\oplus$, for which the following properties hold:

1. Closure: For all $a, b \in S$, we have $a \oplus b \in S$.

2. Identity: There is an element $e \in S$, called the identity of the group, such that $e \oplus a = a \oplus e = a$, for all $a \in S$.

3. Associativity: For all elements $a, b, c \in S$, we have $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.

4. Inverses: For each $a \in S$, there exists a unique element $b \in S$, called the inverse of a, such that $a \oplus b = b \oplus a = e$.

Groups and Subgroups (II)

Definition: For $n \geq 2$, the additive group $\mathbf{Z}_n$ is:

- $S = \{0, 1, \ldots, n-1\}$ – the n residues modulo n,
- $\oplus$ is addition modulo n.

Example: The additive group $\mathbf{Z}_6 = (\{0,1,2,3,4,5\}, +_6)$:

$+_6$	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Groups and Subgroups (III)

Theorem: For $n \geq 2$, the system $\mathbf{Z}_n$ is an abelian group.

Proof:

- Closure: For all $a, b \in \{0, 1, \ldots, n-1\}$, we have $((a+b) \bmod n) \in \{0, 1, \ldots, n-1\}$.
- Identity: The element 0 is the identity for addition modulo n.
- Associativity: For all $a, b, c \in \{0, 1, \ldots, n-1\}$, we have $((((a+b) \bmod n) + c) \bmod n) = ((a + ((b+c) \bmod n)) \bmod n)$.
- Inverses: For all $a \in \{0, 1, \ldots, n-1\}$, we have $-a = ((n-a) \bmod n)$, since $a+(n-a) = 0 \pmod n$.
- Abelian: For all $a, b \in \{0, 1, \ldots, n-1\}$, we have $((a+b) \bmod n) = ((b+a) \bmod n)$.

Groups and Subgroups (IV)

Definition: For $n \geq 2$, the multiplicative group $\mathbf{Z}_n^*$ is:

- $S = \{1 \leq a \leq n-1 : \gcd(a, n) = 1\}$ – relative primes to n between 1 and $n-1$,
- $\oplus$ is multiplication modulo n.

Example: Multiplicative group $\mathbf{Z}_{18}^* = (\{1,5,7,11,13,17\}, \bullet_{18})$:

$\bullet_{18}$	1	5	7	11	13	17
1	1	5	7	11	13	17
5	5	7	17	1	11	13
7	7	17	13	5	1	11
11	11	1	5	13	17	7
13	13	11	1	17	7	5
17	17	13	11	7	5	1

Groups and Subgroups (V)

Theorem: For $n \geq 2$, the system $\mathbf{Z}_n^*$ is an abelian group.

Proof:

- Closure: For all $1 \leq a, b \leq n-1$, if $\gcd(a, n) = 1$ and $\gcd(b, n) = 1$, then $\gcd(a \bullet b, n) = 1$ and $1 \leq ((a \bullet b) \bmod n) \leq n-1$.
- Identity: The element 1 is the identity for multiplication modulo n, and 1 is in $\mathbf{Z}_n^*$ since $\gcd(1, n) = 1$.
- Associativity: For all $1 \leq a, b, c \leq n-1$, we have $((((a \bullet b) \bmod n) \bullet c) \bmod n) = ((a \bullet ((b \bullet c) \bmod n)) \bmod n)$.
- Inverses: For $1 \leq a \leq n-1$, if $\gcd(a, n) = 1$, then there exist integers x any y, such that $ax + ny = 1$. This implies $ax = 1 \pmod n$, which means that $x = a^{-1} \pmod n$.
- Abelian: For all $1 \leq a, b \leq n-1$, we have $((a \bullet b) \bmod n) = ((b \bullet a) \bmod n)$.

Groups and Subgroups (VI)

Euler's Phi Function: The size of $\mathbf{Z}_n^*$ is:

$$\Phi(n) = n \prod_{p|n} (1 - (1/p))$$

(where p runs over all the primes that divide n).

Examples:

$\Phi(7) = 7 \bullet (1-(1/7)) = 6$ and $\mathbf{Z}_7^* = \{1, 2, 3, 4, 5, 6\}$

$\Phi(10) = 10 \bullet (1-(1/2)) \bullet (1-(1/5)) = 4$ and $\mathbf{Z}_{10}^* = \{1, 3, 7, 9\}$

$\Phi(18) = 18 \bullet (1-(1/2)) \bullet (1-(1/3)) = 6$ and $\mathbf{Z}_{18}^* = \{1, 5, 7, 11, 13, 17\}$

Groups and Subgroups (VII)

Definition: If $(S, \oplus)$ is a group, if $S' \subseteq S$, and if $(S', \oplus)$ is also a group, then $(S', \oplus)$ is a subgroup of $(S, \oplus)$.

Example: The even integers with addition are a subgroup of the integers with addition.

Theorem: If $(S, \oplus)$ is a finite group and S' is a non-empty subset of S, such that for all $a, b \in S'$ we have $a \oplus b \in S'$, then $(S', \oplus)$ is a subgroup of $(S, \oplus)$.

Example: The set $\{0, 2, 4, 6\}$ forms a subset of $\mathbf{Z}_8$ that is closed under operation $+_8$ and is therefore a subgroup of $\mathbf{Z}_8$.

Groups and Subgroups (VIII)

Lagrange's Theorem: If $(S, \oplus)$ is a finite group, and if $(S', \oplus)$ is a subgroup of $(S, \oplus)$, then $|S'|$ is a divisor of $|S|$.

Definition: A subgroup S' of group S is said to be a proper subgroup of S if $S' \neq S$.

Corollary: If S' is a proper subgroup of a finite group S, then $|S'| \leq |S| / 2$.

Example: The size of the subgroup $S' = \{0, 2, 4, 6\}$ of $\mathbf{Z}_8$ is 4 which is a divisor of 8.

Groups and Subgroups (IX)

Notation: If $(S, \oplus)$ is a group, and if $a \in S$, we denote successive applications of the group operator $\oplus$ to a by $a^{(k)}$.

Notation: $< a > = \{a^{(k)} : k \geq 1\}$.

Lemma: If $(S, \oplus)$ is a finite group, and if $a \in S$, then $< a >$ is a subgroup of S.

Proof: $< a >$ is closed under the operator $\oplus$.

Definition: If $(S, \oplus)$ is a group, and if $a \in S$, then the subgroup generated by successive applications of $\oplus$ to a is called the subgroup generated by a and is denoted by $< a >$. The element a is called the generator of $< a >$.

Groups and Subgroups (X)

Example: In the additive group $\mathbf{Z}_6$, we have:

$< 0 > = \{0\}$

$< 1 > = < 5 > = \{0, 1, 2, 3, 4, 5\}$

$< 2 > = < 4 > = \{0, 2, 4\}$

$< 3 > = \{0, 3\}$

Example: In the multiplicative group $\mathbf{Z}_7^*$, we have:

$< 1 > = \{1\}$

$< 2 > = < 4 > = \{1, 2, 4\}$

$< 3 > = < 5 > = \{1, 2, 3, 4, 5, 6\}$

$< 6 > = \{1, 6\}$

Groups and Subgroups (XI)

Definition: For a finite group $(S, \oplus)$, the order of element $a \in S$, denoted by $\text{ord}(a)$, is the smallest k, such that $a^{(k)} = e$ (the identity element).

Theorem: For any finite group $(S, \oplus)$ and any element $a \in S$, the order of a is equal to the size of $< a >$, that is $\text{ord}(a) = | < a > |$.

Proof:

- If $\text{ord}(a) = k$, then for all $t \geq 0$, we have $a^{(k+t)} = a^{(t)}$ (since $a^{(k)} = e$). This implies that $| < a > | \leq k = \text{ord}(a)$.
- If $k = \text{ord}(a) > | < a > |$, then there are $1 \leq i < j \leq k$, such that $a^{(i)} = a^{(j)}$. This means $a^{(j-i)} = e$. But $j-i < k$ is contradiction. Thus, $\text{ord}(a) \leq | < a > |$.

Groups and Subgroups (XI)

Corollary: The sequence $a^{(1)}, a^{(2)}, \ldots$ is periodic with period $k = \text{ord}(a)$. That is, $a^{(i)} = a^{(j)}$ if and only if $i = j \pmod k$.

Definition: We can define for all $a \in S$:

- $a^{(0)} = e$ (the identity element), and
- $a^{(i)} = a^{(i \bmod k)}$ (where $k = \text{ord}(a)$).

Theorem: If $(S, \oplus)$ is a finite group with identity e, then for all $a \in S$, we have $a^{(|S|)} = e$.

Proof:

- Lagrange's Theorem implies that $\text{ord}(a)$ divides $|S|$.
- Therefore, $|S| = 0 \pmod k$, where $k = \text{ord}(a)$.

Solving Modular Equations (I)

Observation: A linear equation of the form $ax = b \pmod n$, where $a > 0$, $b \geq 0$, and $n > 0$, may have zero, one, or more solutions.

Observation: Since we have $< a > = \{ ax \pmod n : x > 0 \}$, then the equation $ax = b \pmod n$ has a solution if and only if $b \in < a >$. Lagrange's Theorem implies that $| < a > |$ must be a divisor of n.

Theorem: For any positive integers a and n, if $d = \gcd(a, n)$, then $< a > = < d > = \{ 0, d, 2d, \ldots, ((n/d)-1)d \}$ in $\mathbf{Z}_n$, and thus $| < a > | = n/d$.

Solving Modular Equations (II)

Proof:

- Since $d = \gcd(a, n)$, there are integers x and y such that $ax + ny = d$. This means that $ax = d \pmod n$, which implies that $d \in < a >$.
- Since d is in $< a >$, all the multiples of d should also be in $< a >$. That is, $< a >$ contains all the elements $\{ 0, d, 2d, \ldots, ((n/d)-1)d \}$, which means that $< d > \subseteq < a >$.
- Now, if $m \in < a >$, then $m = ax \pmod n$ for some integer x, and so $m = ax + ny$ for some integer y.
- Now, because $d \mid a$ and $d \mid n$, we also have $d \mid m$. Thus, $m \in < d >$. This means that $< a > \subseteq < d >$.
- Combining the above, we have $< d > = < a >$.
- The set $< d > = < a >$ contains n/d elements, which is the number of multiples of d between 0 and $n-1$.

Solving Modular Equations (III)

Corollary: The equation $ax = b \pmod n$ is solvable for x if and only if $\gcd(a, n) \mid b$.

Corollary: The equation $ax = b \pmod n$ either has d distinct solutions modulo n, where $d = \gcd(a, n)$, or it has no solutions.

Proof:

- If equation $ax = b \pmod n$ has a solution, then $b \in < a >$.
- The sequence $ai \pmod n$ is periodic with period $| < a > | = n/d$.
- If $b \in < a >$, then b appears exactly d times in the sequence $ai \pmod n$, for $i=0, 1, \ldots, n-1$. This is because the block of the n/d values in $< a >$ repeats exactly d times as i increases from 0 to $n-1$.
- The d times that b appears in the sequence $ai \pmod n$ give d solutions.

Solving Modular Equations (IV)

Theorem: Let $d = \gcd(a, n)$, and suppose that $d = ax' + ny'$ for integers x' and y'. If $d \mid b$, then equation $ax = b \pmod n$ has as one of its solutions the value $x_0 = x'(b/d) \pmod n$.

Proof:

We have

$$\begin{aligned} ax_0 &= ax'(b/d) \pmod n && \text{// checking for } x_0 \\ &= d(b/d) \pmod n && \text{// since } ax' = d \pmod n \\ &= b \pmod n && \text{// QED} \end{aligned}$$

Solving Modular Equations (V)

Theorem: Suppose that the equation $ax = b \pmod n$ is solvable (that is, $d = \gcd(a, n)$ divides b) and that x_0 is any solution to this equation. Then, this equation has exactly d distinct solutions, modulo n, given by $x_i = x_0 + i(n/d)$, for $i=0, 1, \ldots, d-1$.

Proof:

- The d values x_i are all distinct modulo n (since $0 \le i(n/d) < n$, and since $n/d > 0$).
- For all $i=0, 1, \ldots, d-1$, we have $ax_i = a(x_0 + i(n/d)) = ax_0 + ain/d = b \pmod n$, because d is a divisor of n.
- Each x_i is a solution, and the d values of x_i are distinct, which means that the d values of x_i are the only d solutions.

Solving Modular Equations (VI)

Algorithm:

```
Modular-Linear-Equation-Solver(a, b, n)
1   (d, x', y') ← Extended-Euclid(a, n)
2   if ( d | b)
3      then x_0 ← x'(b/d) mod n
4           for i ← 0 to d-1 do
5               print ((x_0 + i(n/d)) mod n)
6      else print "no solutions"
```


Solving Modular Equations (VII)

Algorithm Example:

- Consider the equation $14x = 30 \pmod{100}$
- In this equation $a=14$, $b=30$, and $n=100$
- Compute $d = \gcd(14, 100) = 2$
- Compute $x' = -7$
- Compute $x_0 = x'(b/d) \pmod n = 95$
- Compute $x_{i=1} = x_0 + i(n/d) \pmod n = 45$

Solving Modular Equations (VIII)

Corollary: For any $n > 1$, if $\gcd(a, n) = 1$, then the equation $ax = b \pmod n$ has a unique solution modulo n.

Corollary: For any $n > 1$, if $\gcd(a, n) = 1$, then a has a unique inverse modulo n. Otherwise, it has no inverse modulo n.

Proof: In the equation $ax = 1 \pmod n$, the only solution for x is the inverse of a modulo n.

Algorithm: For any $n > 1$, if $\gcd(a, n) = 1$, to compute a^{-1} in modulo n, one can call the algorithm Extended-Euclid(a, n) to find $d = \gcd(a, n) = 1$, and x and y with $ax + ny = 1 \pmod n$. The value of x is $x = a^{-1} \pmod n$.

Chinese Remainders (I)

Problem: Find integers that leave remainders 2, 3, and 2 when divided by 3, 5, and 7, respectively.

Solution: All integers of the form $23+105k$, for integer k.

Chinese Remainder Theorem provides a correspondence between a system of equations modulo a set of pairwise relatively prime moduli (such as 3, 5, and 7 in the example above) and an equation modulo their product (such as 105 in the example above).

Chinese Remainders (II)

Setting: Let the integer n be the product of k pairwise relatively prime factors $n_1, n_2, \ldots, n_k$. Represent an integer a either by its residue modulo n (that is, by $(a \bmod n)$), or by its k residues modulo the factors $n_1, n_2, \ldots, n_k$ (that is, by the k residues $a_1 = (a \bmod n_1), a_2 = (a \bmod n_2), \ldots, a_k = (a \bmod n_k)$).

Use: Chinese Remainders Theorem is useful in establishing an isomorphism between $\mathbf{Z}_n$ and $\mathbf{Z}_{n_1} \times \mathbf{Z}_{n_2} \times \cdots \times \mathbf{Z}_{n_k}$.

Use: Chinese Remainders Theorem allows finding results of arithmetic operations (such as addition and multiplication) in $\mathbf{Z}_n$ by doing the arithmetic in each of the $\mathbf{Z}_{n_i}$.

Chinese Remainders (III)

Chinese Remainders Theorem:

Let $n = n_1 n_2 \cdots n_k$, where the n_i are pairwise relatively prime.

We correspond $a \leftrightarrow (a_1, a_2, \ldots, a_k)$, where $a \in \mathbf{Z}_n$, and $a_i \in \mathbf{Z}_{n_i}$, and $a_i = (a \bmod n_i)$ for $i = 1, 2, \ldots, k$.

If $a \leftrightarrow (a_1, a_2, \ldots, a_k)$ and $b \leftrightarrow (b_1, b_2, \ldots, b_k)$, then:

$(a + b) \bmod n \leftrightarrow ((a_1 + b_1) \bmod n_1, ((a_2 + b_2) \bmod n_2, \ldots, ((a_k + b_k) \bmod n_k)$

$(a - b) \bmod n \leftrightarrow ((a_1 - b_1) \bmod n_1, ((a_2 - b_2) \bmod n_2, \ldots, ((a_k - b_k) \bmod n_k)$

$(a \bullet b) \bmod n \leftrightarrow ((a_1 \bullet b_1) \bmod n_1, ((a_2 \bullet b_2) \bmod n_2, \ldots, ((a_k \bullet b_k) \bmod n_k)$

Chinese Remainders (IV)

Proof:

- Going from the representation $a \in \mathbf{Z}_n$ to the representation of the k-tuple of the $a_i \in \mathbf{Z}_{n_i}$ is straightforward (by doing k divisions).
- Going from the k-tuple of the $a_i \in \mathbf{Z}_{n_i}$ to the representation $a \in \mathbf{Z}_n$ is done as follows:
  - Define $m_i = n / n_i$, for $i = 1, 2, \ldots, k$.
    (Thus, m_i is the product of all the n_j's other than n_i.)
  - This definition provides $m_j = 0 \pmod{n_i}$, for $i \neq j$.
    (Because m_j contains n_i as one of its factors.)
  - This definition provides $m_i \neq 0 \pmod{n_i}$.
    (Because m_i contains only factors that are relatively prime to n_i.)

Chinese Remainders (V)

Proof (continued):

- Define $c_i = m_i \bullet (m_i^{-1} \bmod n_i)$, for $i = 1, 2, \ldots, k$.
  (Value of m_i^{-1} is well defined since m_i and n_i are relatively prime.)
- These definitions provide $c_i = 1 \pmod{n_i}$.
- These definitions provide $c_i = 0 \pmod{n_j}$, for $j \neq i$.
- Compute $a \in \mathbf{Z}_n$ as a function of the $a_i \in \mathbf{Z}_{n_i}$ as follows:
  $a = (a_1 \bullet c_1 + a_2 \bullet c_2 + \ldots + a_k \bullet c_k) \pmod{n}$.
- This definition provides $a = a_i \pmod{n_i}$, for $i = 1, 2, \ldots, k$.

- The correspondence of the operations (addition, subtraction, and multiplication) follows from the above one-to-one mapping.

Chinese Remainders (VI)

Corollary: If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \cdots n_k$, then for any integers $a_1, a_2, \ldots, a_k$, the set of simultaneous equations $x = a_i \pmod{n_i}$, for $i = 1, 2, \ldots, k$, has a unique solution modulo n for the unknown x.

Corollary: If $n_1, n_2, \ldots, n_k$ are pairwise relatively prime and $n = n_1 n_2 \cdots n_k$, then for all integers x and a, we have $x = a \pmod{n_i}$, for $i = 1, 2, \ldots, k$, if and only if $x = a \pmod{n}$.

Chinese Remainders (VII)

Example: Find a that solves the two equations:

$a = 2 \pmod 5$ and

$a = 3 \pmod{13}$.

Solution:

$a_1 = 2$, $n_1 = 5$, and $m_1 = 13$.

$a_2 = 3$, $n_2 = 13$, and $m_2 = 5$.

We need to find $(a \bmod 65)$, since $n = n_1 n_2 = 65$.

$m_1^{-1} \pmod{n_1} = 13^{-1} \pmod 5 = 2$ and $c_1 = 13 \bullet (2 \bmod 5) = 26$.

$m_2^{-1} \pmod{n_2} = 5^{-1} \pmod{13} = 8$ and $c_2 = 5 \bullet (8 \bmod 13) = 40$.

$a = 2 \bullet 26 + 3 \bullet 40 \pmod{65} = 42$.

Chinese Remainders (VIII)

Chinese Residue Table: For $n_1 = 5$ and $n_2 = 13$.

The number in the row with index i and column j has residue $(i \bmod 5)$ and $(j \bmod 13)$.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	40	15	55	30	5	45	20	60	35	10	50	25
1	26	1	41	16	56	31	6	46	21	61	36	11	51
2	52	27	2	42	17	57	32	7	47	22	62	37	12
3	13	53	28	3	43	18	58	33	8	48	23	63	38
4	39	14	54	29	4	44	19	59	34	9	49	24	64

Modular Powers (I)

Problem: Computing powers of an element modulo n.

Example: Powers of 3 modulo 7

i	0	1	2	3	4	5	6	7	8	9
3^i	1	3	2	6	4	5	1	3	2	6

Example: Powers of 2 modulo 7

i	0	1	2	3	4	5	6	7	8	9
2^i	1	2	4	1	2	4	1	2	4	1

Modular Powers (II)

Notation: Let $< a >$ denote the subgroup of $\mathbf{Z}_n^*$ generated by a by repeated multiplication, and let $\mathrm{ord}_n(a)$ denote the order of a in $\mathbf{Z}_n^*$.

Definition: If for $g \in \mathbf{Z}_n^*$, we have $\mathrm{ord}_n(g) = |\mathbf{Z}_n^*|$ (that is, every element in $\mathbf{Z}_n^*$ is a power of g), then g is called a generator or a primitive root of $\mathbf{Z}_n^*$.

Examples: 3 is a generator of $\mathbf{Z}_7^*$.
2 is not a generator of $\mathbf{Z}_7^*$.

Definition: If $\mathbf{Z}_n^*$ has a generator, then the group $\mathbf{Z}_n^*$ is called cyclic.

Modular Powers (III)

Euler's Theorem: For any integer $n > 1$ and any $a \in \mathbf{Z}_n^*$, we have $a^{\Phi(n)} = 1 \pmod n$.

Fermat's Theorem: If p is prime, then for any $a \in \mathbf{Z}_p^*$ we have $a^{p-1} = 1 \pmod p$.

Corollary: If p is prime, then for any $a \in \mathbf{Z}_p$ we have $a^p = a \pmod p$.

Theorem (Niven-Zuckerman): The values of $n > 1$ for which $\mathbf{Z}_n^*$ is cyclic are 2, 4, p^e, and $2p^e$, for all primes $p > 2$ and all positive integers e.

Modular Powers (IV)

Definition: If g is a generator of $\mathbf{Z}_n^*$ and a is any element in $\mathbf{Z}_n^*$, then there exists a z such that $g^z = a \pmod n$. This z is called the discrete logarithm or index of a, modulo n, to the base g. We denote this value as $\mathrm{ind}_{n,g}(a)$.

Examples:

$\mathrm{ind}_{7,3}(1) = 0$ $\quad$ $\mathrm{ind}_{7,3}(2) = 2$

$\mathrm{ind}_{7,3}(3) = 1$ $\quad$ $\mathrm{ind}_{7,3}(4) = 4$

$\mathrm{ind}_{7,3}(5) = 5$ $\quad$ $\mathrm{ind}_{7,3}(6) = 3$

Discrete Logarithm Theorem: If g is a generator of $\mathbf{Z}_n^*$, then the equation $g^x = g^y \pmod n$ holds if and only if the equation $x = y \pmod{\Phi(n)}$ holds.

Modular Powers (V)

Proof:

- Suppose $x = y \pmod{\Phi(n)}$.
  Then $x = y + k\Phi(n)$, for some integer k.
  Therefore, $g^x = g^{y+k\Phi(n)} \pmod n$
  $= g^y (g^{\Phi(n)})^k \pmod n$
  $= g^y (1)^k \pmod n$
  $= g^y$.
- Suppose $g^x = g^y \pmod n$.
  Because the sequence of powers of g generates every element of $< g >$, and because $| < g > | = \Phi(n)$, then the sequence of powers of g is periodic with period $\Phi(n)$. Therefore, if $g^x = g^y \pmod n$, we must have $x = y \pmod{\Phi(n)}$.

Modular Powers (VI)

Theorem: If $p > 2$ is a prime and $e \geq 1$, then the equation $x^2 = 1 \pmod{p^e}$ has only two solutions: $x = 1$ or $x = -1$.

Proof:

- Let $n = p^e$. Niven-Zucekrman Theorem implies that $\mathbf{Z}_n^*$ is cyclic. Let g be a generator of $\mathbf{Z}_n^*$.
- Equation $x^2 = 1 \pmod n$ is written – $(g^{\text{ind}_{n,g}(x)})^2 = g^{\text{ind}_{n,g}(1)} \pmod n$.
- The above equation is equivalent to $(2 \cdot \text{ind}_{n,g}(x)) = 0 \pmod{\Phi(n)}$.
- Solving the linear equation in $\text{ind}_{n,g}(x)$, we note that $\Phi(n) = p^e (1 - 1/p) = (p - 1) p^{e-1}$, which is an even number.
- Therefore, $\gcd(2, \Phi(n)) = 2$, and equation $(2 \cdot \text{ind}_{n,g}(x)) = 0 \pmod{\Phi(n)}$ has exactly two solutions – $\text{ind}_{n,g}(x) = 0$ or $\text{ind}_{n,g}(x) = \Phi(n)/2$.
- The two solutions for $\text{ind}_{n,g}(x)$ result in solutions $x = 1$ or $x = -1$.

Modular Powers (VII)

Definition: A number x is a nontrivial square root of 1, modulo n, if it satisfies the equation $x^2 = 1 \pmod n$ but x is neither 1 nor -1 modulo n.

Example: 6 is a nontrivial square root of 1 modulo 35.

Theorem: if there exists a nontrivial square root of 1, modulo n, then n is composite.

Proof:

- From previous Theorem – n cannot be a prime greater than 2.
- If $n = 2$, then all the square root are trivial.
- Since n must be greater than 2, we get that n must be composite.

Modular Powers (VIII)

Modular Exponentiation: Raising a number a to the power of another number b, modulo n. That is, computing $a^b \pmod n$.

Repeated Multiplication: Successive multiplications by the value of a takes linear time in the size of b.

Repeated Squaring: Successive squaring of the number a leads to an algorithm that takes logarithmic time in the size of b.

Notation: Let $< b_k, b_{k-1}, \ldots, b_1, b_0 >$ be the binary representation of the exponent b.

Modular Powers (IX)

Algorithm:

```
Modular-Exponentiation(a, b, n)
1   c ← 0
2   d ← 1
3   for i ← k downto 0 do
4       c ← 2 • c
5       d ← (d • d) mod n
6       if b_i = 1
7           then   c ← c + 1
8                  d ← (d • a) mod n
9   return d
```


Modular Powers (X)

Example: a=7 b=560 n=561

i	9	8	7	6	5	4	3	2	1	0
b_i	1	0	0	0	1	1	0	0	0	0
c	1	2	4	8	17	35	70	140	280	560
d	7	49	157	526	160	241	298	166	67	1

Observation: If the inputs a, b, and n are k-bit long, then the total number of arithmetic operations is O(k) and the total number of bit operations is $O(k^3)$.

Fields (I)

Definition: A Field $F = (S, +, \bullet)$ is a set S with two binary operators + and •, for which the following properties hold:

1. Additive Group: The structure $(S, +)$ is an additive group with an identity element denoted by 0.

2. Multiplicative Group: The structure $(S-\{0\}, \bullet)$ is a multiplicative group with an identity element denoted by 1.

3. Commutativity: Operators + and • are commutative.

4. Distributivity: For any three elements $a, b, c \in S$, we have $a \bullet (b + c) = a \bullet b + a \bullet c$.

Fields (II)

Examples:

- $\mathbf{Z}_p$ – the integers modulo prime p
- $\mathbf{Q}$ – the rational numbers
- $\mathbf{R}$ – the real numbers
- $\mathbf{C}$ – the complex numbers

Fields (III)

Z_p – the integers modulo prime p:

- It is a commutative group with addition modulo p
- It is a commutative group with multiplication modulo p
- It obeys the distributive law

Example:

- $GF(5) = \mathbf{Z}_5 = (\{0, 1, 2, 3, 4\}, +_5, \bullet_5)$
- $GF(2) = \mathbf{Z}_2 = (\{0, 1\}, \text{XOR}, \text{AND})$

Fields (VI)

$GF(2^2)$

Elements – $\{ 0, 1, a, b \}$

Addition –

0+0=0	1+0=1	a+0=a	b+0=b
0+1=1	1+1=0	a+1=b	b+1=a
0+a=a	1+a=b	a+a=0	b+a=1
0+b=b	1+b=a	a+b=1	b+b=0

Multiplication –

0•0=0	1•0=0	a•0=0	b•0=0
0•1=0	1•1=1	a•1=a	b•1=b
0•a=0	1•a=a	a•a=b	b•a=1
0•b=0	1•b=b	a•b=1	b•b=a

Polynomials (I)

Definition: Polynomial $c(x) = \Sigma\, c_i x^i$, where the coefficients c_i are from some field.

Definition: The degree of polynomial is the highest exponent of x having a non-zero coefficient.

Definition: A polynomial with leading coefficient of 1 is called a monic.

Operations on Polynomials:

- Addition: $\sum_i c_i x^i + \sum_i d_i x^i = \sum_i (c_i + d_i) x^i$
- Multiplication: $\sum_i c_i x^i \bullet \sum_j d_i x^i = \sum_i \left(\sum_{j=0}^{i} c_{i-j} d_j \right) x^i$

Polynomials (II)

Properties:

- There is a zero polynomial – $z(x) = 0$.
- There is a unity polynomial – $u(x) = 1$.
- Polynomial addition and multiplication are associative, commutative, and distributive.
- The additive inverse of a polynomial is the polynomial with inverse coefficients in the field.
- What is the multiplicative inverse of a polynomial ???
- There is a notion of polynomial division with remainders.

Polynomials (III)

Polynomial Division:

- For any two polynomials:
  $p(x)$ – the dividend polynomial, and
  $d(x)$ – the divisor polynomial,
  there are two unique polynomials:
  $q(x)$ – the quotient polynomial, and
  $r(x)$ – the remainder polynomial,
  such that – $p(x) = q(x)\, d(x) + r(x)$
  and the degree of $r(x)$ is smaller than the degree of $d(x)$.
- If $r(x)$ is zero, then we say that $d(x)$ divides $p(x)$.
- Division algorithm is very similar to long division of numbers.

Polynomials (IV)

Example of Polynomial Division:

- Let the polynomials' coefficients be over $\mathbf{Z}_5$
- Let $p(x) = x^5 + x^4 + x^3 + x^2 + x + 1$
  and $d(x) = 2x^3 + 3x + 1$
- Then $q(x) = 3x^2 + 3x + 1$
  and $r(x) = 4x^2$

Polynomials (V)

Polynomial Modular Operations:

- Given a non-zero polynomial $m(x)$, we can represent any polynomial $p(x)$ as its remainder $r(x)$ when divided by $m(x)$. We write $r(x) = p(x) \bmod m(x)$.
- We can add, subtract, and multiply polynomials $a(x)$ and $b(x)$ modulo $m(x)$.
- We can divide polynomial $a(x)$ by polynomial $b(x)$, in modulo $m(x)$, if the multiplicative inverse of polynomial $b(x)$ exists in modulo $m(x)$.
- The greatest common divisor of two polynomials is the highest-degree monic polynomial dividing both of them.

Polynomials (VI)

Polynomial Greatest Common Divisor Example:

Let the polynomials' coefficients be over $\mathbf{Z}_5$

$\gcd(\, 2x^4+3x^3+4x^2+2x+1 \,,\, 4x^3+1 \,) =$

$\gcd(\, 4x^3+1 \,,\, 4x^2+4x+4 \,) =$

$\gcd(\, 4x^2+4x+4 \,,\, x^2+x+1 \,) =$

$\gcd(\, x^2+x+1 \,,\, 0 \,) =$

x^2+x+1

Polynomials (VII)

Irreducible Polynomial:

- A polynomial $p(x)$ with coefficients from a field F is called irreducible if and only if it can only be divided by $u(x)=1$ and by itself $p(x)$.
- Examples of polynomials with coefficients from $\mathbf{Z}_2$:

$x^2 = x \cdot x$	composite
$x^2 + 1 = (x+1) \cdot (x+1)$	composite
$x^2 + x = x \cdot (x+1)$	composite
$x^2 + x + 1$	irreducible

Polynomials (VIII)

Evaluating Polynomials:

- A polynomial $p(x)$ with coefficients from a field F can be evaluated for any value $v \in F$. The result $p(v)$ is a value in the field F.
- A root of a polynomial $p(x)$ with coefficients from field F is a value $v \in F$ such that $p(v) = 0$.
- **Claim**: Element $v \in F$ is a root of polynomial $p(x)$ if and only if the degree-1 polynomial $(x-v)$ is a factor of $p(x)$.
- **Conclusion**: Polynomial $p(x)$ can be uniquely written as a product $\prod (x-v_i)^{k_i}$ where v_i are different roots of $p(x)$.

Galois Fields(I)

Finite Fields:

- The characteristics of a finite field is the minimal number of times that 1 can be added to itself until the result becomes 0.
- The characteristics of a finite field must be a prime number.
- If the characteristics of a finite field F is $p \geq 2$, then the number of elements in F must be a power of p – that is – p^n for some $n \geq 1$.
- For a given prime $p \geq 2$ and an integer $n \geq 1$, there is only one field of size $q = p^n$. This field is called the Galois Field of order q and is denoted $GF(q)$.
- There could be many ways to represent $GF(q)$.

Galois Fields(II)

Useful Representation of GF(q):

- A useful representation of $GF(q)$, for $q = p^n$, is as all the p^n vectors of length n, where each component of a vector is a number in $\mathbf{Z}_p$.
- Addition of two elements in $GF(q)$ is done by component-wise addition in $\mathbf{Z}_p$ of the two n-component vectors, which correspond to the two elements in $GF(q)$.
- Multiplication of two elements in $GF(q)$ is done by multiplying the two polynomials, which are derived from the corresponding two n-component vectors, by treating the vector components as the coefficients of the polynomials. The result need to be taken modulo a fixed irreducible polynomial of degree n.
- A polynomial is irreducible if its only divisors are 1 and itself.

Galois Fields(III)

Example of $GF(2^8)$:

- The field $GF(2^8)$ has $2^8 = 256$ elements.
- The underlying field of characteristics 2 is $\mathbf{Z}_p = (\{0,1\}, \text{XOR}, \text{AND})$.
- Each element in $GF(2^8)$ can be represented by an octet (8 bits).
- Addition of two elements in $GF(2^8)$ (octets) is done by XORing the two octets bit-by-bit. Each element in is its own additive inverse.
- Multiplication of two elements in $GF(2^8)$ (octets) is done by first multiplying the two polynomials (each of degree at most 7), which are obtained by treating the 8 bits of each octet as the coefficients of the polynomials. Then, the resulting polynomial is taken modulo the irreducible polynomial $m(x) = x^8+x^4+x^3+x+1$.
- Example: $\{57\} + \{83\} = \{D4\}$ and $\{57\} \bullet \{83\} = \{C1\}$.

Mathematics of AES

Fields used in AES:

- AES uses arithmetic over the field $GF(2^8)$:
  - Each byte (octet) in AES represents an element in $GF(2^8)$.
  - Addition of two octets is bit-wise XOR of the two octets.
  - Multiplication of two octets is like polynomial multiplication modulo the irreducible polynomial $x^8+x^4+x^3+x+1$ over Z_2.
- AES also uses polynomials (of degree up to 3) with coefficients in $GF(2^8)$, that is, with 8-bit vectors as coefficients:
  - Addition of two polynomials is bit-wise XOR of the four 8-bit vectors.
  - Multiplication of two polynomials is polynomial multiplication modulo the irreducible polynomial x^4+1 over $GF(2^8)$.
- AES also uses polynomials (of degree up to 7) with coefficients in Z_2.
  - For these operations, the irreducible polynomial x^8+1 over Z_2 is used.

Public-Key Cryptography I

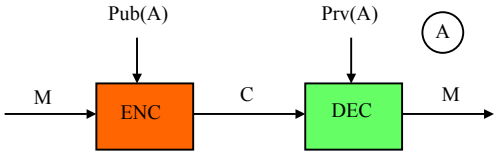
Prof. Shlomo Kipnis
November 28, 2007

Public-Key Model

- Entity A has two keys
  - Prv(A) – Private Key of A, kept secretly only by A
  - Pub(A) – Public Key of A, made public to the world
- Two functions used for encryption/decryption:
  - ENC – uses the public key
  - DEC – uses the private key
  - Such that DEC [ENC (M, Pub(A)) , Prv(A)] = M
- Two functions used for signature/verification:
  - SIGN – uses the private key
  - VER – uses the public key
  - Such that VER [SIGN (M, Prv(A)) , Pub(A)] = yes / no

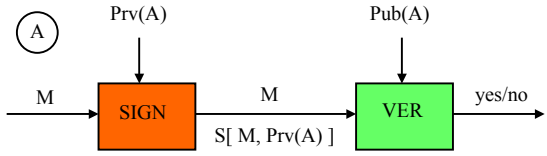
Public-Key Encryption

- Anyone knowing Pub(A) can encrypt message M for A
- Only A can decrypt, using Prv(A), message M that was encrypted for him



Public-Key Signatures

- Only A can sign, using Prv(A), some message M
- Anyone knowing Pub(A) can verify that message M is signed properly by A



History of Public-Key Schemes (I)

- 1976 – Diffie & Hellman suggested the public-key model for encryption and signatures
- 1976 – Diffie & Hellman developed public-key protocol for key-exchange based on the Discrete Log Problem
- 1976 – Merkle suggested a public-key encryption method based on the Knapsack Problem
- 1977 – Rivest, Shamir, Adelman developed the RSA public-key scheme for encryption and signatures based on the Number Factoring Problem

History of Public-Key Schemes (II)

- 1980's – Lamport suggested a one-time-signature scheme based on Hash Functions
- 1980's – El-Gamal developed public-key protocols for encryption and signatures based on the Discrete Log Problem
- 1980's – Various researchers developed public-key schemes based on Elliptic Curves

Public-Key Schemes – Requirements (I)

- The encryption function, the decryption function, the signature function, and the verification function should all be known functions
- The encryption function, the decryption function, the signature function, and the verification function should all be easy to compute when the relevant keys (public or private) are given
- Given the encryption and decryption functions (or the signature and verification function), and given the public key – it should be hard to find the private key

Public-Key Schemes – Requirements (II)

- Given a cipehrtext and the public key – it should be hard to find the plaintext
- Given many pairs of (plaintext, ciphertext) and given the public key – it should be hard to find the private key
- Given many pairs of (message, signature) and given the public key – it should be hard to find the private key
- Given many pairs of (message, signature) and given the public key – it should be hard to find the signature of a new (unseen) message

Knapsack Public-Key Encryption (I)

- Simple Integer Knapsack Problem:
  - Given weight vector $K = (k_1, k_2, \ldots, k_n)$, where each k_i is integer
  - Given selection vector $X = (x_1, x_2, \ldots, x_n)$, where each x_i is 0 or 1
  - Capacity of the selected items is $C = K \bullet X = \Sigma\, k_i x_i$
  - Given the weight vector K and the selection vector X, it is easy to compute the capacity C
  - Given the weight vector K and the capacity C, it is an NP-hard problem to find the selection vector X

Knapsack Public-Key Encryption (II)

- The encryption scheme:
  - Public-Key is vector $K = (k_1, k_2, \ldots, k_n)$, where each k_i is integer
  - Message is vector $M = (x_1, x_2, \ldots, x_n)$, each x_i is 0 or 1
  - Cipher is number $C = K \bullet M = \Sigma\, k_i x_i$
- Problem:
  - There could be more than one decryption for a given cipher C
- Example:
  - $K = (1, 2, 3, 5)$
  - $C = 6$ decrypts to $M = (1, 1, 1, 0)$ and also to $M' = (1, 0, 0, 1)$

Knapsack Public-Key Encryption (III)

- Super-Increasing Integer Knapsack Problem:
  - Public-Key will be of the form $K = (k_1, k_2, \ldots, k_n)$, where each k_i is integer, and where for each i we have $k_i > k_1 + k_2 + \ldots + k_{i-1}$
  - For message $M = (x_1, x_2, \ldots, x_n)$, where each x_i is 0 or 1 – the cipher $C = K \bullet M = \Sigma\, k_i x_i$ can be easily computed
  - For a given cipher C, decryption is unique and easy:
    - Find the largest index i for which $C > k_i$
    - The element k_i must be in the knapsack – meaning $x_i = 1$
    - Repeat this process with new capacity $C - k_i$
- Problem:
  - Anyone who knows Public-Key K can decrypt a cipher C

Knapsack Public-Key Encryption (IV)

- Scrambled Super-Increasing Integer Knapsack Problem:
  - Choose random super-increasing vector $K = (k_1, k_2, \ldots, k_n)$, where each k_i is integer and for each i we have $k_i > k_1 + k_2 + \ldots + k_{i-1}$
  - Choose integer m such that $m > k_1 + k_2 + \ldots + k_n$
  - Choose integer w relatively prime to m – that is – $\gcd(m,w) = 1$
  - Calculate w^{-1} – that is a number such that $ww^{-1} = 1 \pmod m$
  - Public-Key is defined as:
    $K^* = (k^*_1, k^*_2, \ldots, k^*_n) = (wk_1 \pmod m, wk_2 \pmod m, \ldots, wk_n \pmod m)$,
    which is a scrambled (and not super-increasing) form of K
  - Private-Key consists of three things:
    m, w^{-1}, and the selected super-increasing vector K

Knapsack Public-Key Encryption (V)

- Scrambled Super-Increasing Integer Knapsack Problem:
  - Encrypting a message $M = (x_1, x_2, \ldots, x_n)$ with the Public-Key $K^* = (k^*_1, k^*_2, \ldots, k^*_n)$ is done by computing $C^* = K^* \bullet M = \Sigma\, k^*_i x_i$
  - Decrypting a cipher C^* with the Private-Key is done as follows:
    - Calculate $C = w^{-1}C^* \pmod m$
    - Use C and the super-increasing vector K to find message M
  - Decryption works since:

$$\begin{aligned} C &= w^{-1}C^* \pmod m \\ &= w^{-1} \Sigma\, k^*_i x_i \pmod m \\ &= \Sigma\, w^{-1}k^*_i x_i \pmod m \\ &= \Sigma\, k_i x_i \pmod m \\ &= \Sigma\, k_i x_i \end{aligned}$$

  and from C and the vector K, one can decrypt M

Knapsack Public-Key Encryption (VI)

- In 1976, Merkle suggested the Integer Knapsack Problem as the basis for a Public-Key encryption scheme
- Although the general Integer Knapsack Problem is indeed an NP-Hard Problem, the Super-Increasing Integer Knapsack Problem is not NP-Hard
- In 1977, Shamir broke the Public-Key encryption scheme that was based on the Super-Increasing Integer Knapsack Problem

One-Time Signatures with Hash (I)

Private and Public Keys

- Entity A invents a one-time-private-key $Prv(A)$ suitable for signing one message M of length n bits: $Prv(A)$ is n ordered pairs of values (X_i, Y_i) for $i = 1, 2, \ldots N$, where X_i and Y_i are very large random numbers
- Entity A publishes matching one-time-public-key $Pub(A)$: $Pub(A)$ is n ordered pairs of values $(H(X_i), H(Y_i))$ for $i = 1, 2, \ldots N$, where H is a good and known hash function

One-Time Signatures with Hash (II)

Signing Message M

- M is a message of length n bits
- If i-th bit of M is 0 – then entity A discloses X_i
- If i-th bit of M is 1 – then entity A discloses Y_i
- The signature of M is the collection of n values of either X_i or Y_i for each of the bits of M

Verifying Signature of Message M

- If i-th bit of M is 0 – check if $H(X_i)$ appears in $Pub(A)$
- If i-th bit of M is 1 – check if $H(Y_i)$ appears in $Pub(A)$

One-Time Signatures with Hash (III)

- Question: Why is the signature good?
- Answer-1: An opponent cannot change any bit of M, because he does not know the other value (X_i or Y_i), and because he cannot find collisions of the hash function.
- Answer-2: An opponent that sees M and its signature cannot find the private key $Prv(A)$, because he cannot invert the hash function.

One-Time Signatures with Hash (IV)

- Question: Why is this a one-time signature scheme?
- Answer: Seeing two different messages and their signatures allows creating signatures for other messages even without the knowledge of the private key $Prv(A)$.
- Practicality: The scheme is not practical since the keys are huge and one-time.

Hard Number Theory Problems

- Observation: It is easy to compute g^x modulo prime p to obtain $y = g^x \pmod p$. But, given p, g, and y, it is difficult to compute the discrete logarithm $x = \text{ind}_{p,g}(y)$.
  - This is the basis of the Diffie-Hellman, El-Gamal, and DSS Public-Key schemes.
- Observation: It is easy to compute the product of two primes p and q to obtain $n = pq$. But it is difficult to factor the composite number n into its two prime factors p and q.
  - This is the basis of the RSA Public-Key scheme.

Discrete Logarithm (I)

- Definition: If g is a generator of $\mathbf{Z}_n^*$ and a is any element in $\mathbf{Z}_n^*$, then there exists a z such that $g^z = a \pmod n$. This z is called the discrete logarithm or index of a, modulo n, to the base g. We denote this value as $\text{ind}_{n,g}(a)$.
- Examples:

$\text{ind}_{7,3}(1) = 0$ $\quad$ $\text{ind}_{7,3}(2) = 2$
$\text{ind}_{7,3}(3) = 1$ $\quad$ $\text{ind}_{7,3}(4) = 4$
$\text{ind}_{7,3}(5) = 5$ $\quad$ $\text{ind}_{7,3}(6) = 3$

Discrete Logarithm (II)

- Discrete Exponentiation Problem:
  Given g a generator of $\mathbf{Z}_n^*$ and given a value $x \in \mathbf{Z}_n^*$, find the $y \in \mathbf{Z}_n^*$ such that $y = g^x \pmod n$.
- Observation: Computing discrete exponents can be done very efficiently.
- Discrete Logarithm Problem:
  Given g a generator of $\mathbf{Z}_n^*$ and given a value $y \in \mathbf{Z}_n^*$, find the $x \in \mathbf{Z}_n^*$ such that $x = \text{ind}_{n,g}(y)$.
- Observation: We don't have an efficient algorithm for computing discrete logarithms.

Discrete Logarithm (III)

- Known Algorithms:
  - Fastest known algorithm can find discrete logarithm in modulo p in about $O(e^{(1.923 (\ln p)^{1/3} (\ln \ln p)^{2/3})})$ steps
  - Algorithm is based on the Generalized Number Sieve Algorithm for number factoring

Number Factoring (I)

- Observation: Factoring a large composite number n is considered to be a very difficult task (although detecting whether a large number is prime or composite had proven to be a relatively simple task).
- Observation: Trying to divide a large number n by all the "small" integers successively will yield an $O(\sqrt{n})$ algorithm.
- Observation: Best known methods for factoring a large number n still require time exponential in the length of n.
- Observation: It is infeasible with today's computers and algorithms to factor an arbitrary 1024-bit number.

Number Factoring (II)

- Pollard-Rho Heuristic:

```
Pollard-Rho(n)
1   i ← 1
2   x1 ← Random(0, n-1)
3   y ← x1
4   k ← 2
5   while TRUE do
6       i ← i + 1
7       xi ← (x(i-1))^2 – 1  mod n
8       d ← gcd( y – xi , n)
9       if (d ≠ 1) and (d ≠ n)
10          then print d
11      if i = k
12          then y ← xi
13               k ← 2k
```

Number Factoring (III)

- Ideas of the Pollard-Rho Heuristic:
  - The algorithm generates a sequence of $\{x_i\}$ by $x_i \leftarrow f(x_{i-1}) \bmod n$. This sequence happens to be "randomly distributed" modulo n.
  - The sequence $\{x_i\}$ is very likely to repeat after $O(\sqrt{n})$ steps.
  - If p is a factor of n, then the sequence is also very likely to repeat after $O(\sqrt{p})$ steps.
  - Once the sequence enters a cycle modulo p, it stays within the cycle forever.
  - Inside the cycle modulo p, the sequence will find a the value of p (or a multiple of p) in about $O(\sqrt{p})$ steps.
  - The algorithm is not guaranteed to terminate…

Number Factoring (IV)

- Known Algorithms:
  - Pollard-Rho heuristic can find small factors p of n in about $O(n^{1/4})$ steps
  - Generalized Number Sieve Algorithm can factor a number n in order $L(1/3, n)^{1.923+o(1)}$ steps, where $L(\alpha, n) = e^{((\ln n)^{\alpha}(\ln \ln n)^{1-\alpha})}$

Primality Testing (I)

- Definition: The prime distribution function $\pi(n)$ specifies the number of primes that are less than or equal to n.
- Example: $\pi(10) = 4$
- Prime Number Theorem: $\lim\left[\frac{\pi(n)}{n / \ln n}\right] = 1$
- Observation: The approximation $n / \ln n$ gives reasonably accurate estimates of $\pi(n)$ even for small n.
- Example: $\pi(10^9) = 50{,}847{,}534$ where $n / \ln n \approx 48{,}254{,}942$.

Primality Testing (II)

- Observation: We can estimate the probability that a randomly chosen integer n will turn out to be prime as $1 / \ln n$.
- Observation: We would need to examine approximately $\ln n$ integers chosen randomly near n in order to find a prime that is the same length as n.
- Example: To find a 512-bit prime might require testing approximately $\ln 2^{512} \approx 355$ randomly chosen 512-bit numbers for primality. (Actually, the number of tests can be cut in half, since we don't need to test even integers.)

Primality Testing (III)

- Observation: A simple approach of testing whether a number n is prime or not involves trying to divide n by all the numbers $2, 3, \ldots \sqrt{n}$. This approach takes time that is proportional to $\sqrt{n}$, which is exponential in the length of n.
- Conclusion: Trial division can be used only for small values of n, or if n happens to have a small prime factor. If this is the case, there is a further advantage of being able to identify one of the factors of n.
- Conclusion: For large n, we need to develop another algorithm for detecting whether n is prime or composite.

Primality Testing (IV)

- Definition: $\mathbf{Z}_n^+ = \{1, 2, \ldots, n-1\}$.
- Observation: If n is prime, then $\mathbf{Z}_n^+ = \mathbf{Z}_n^*$.
- Definition: We say that n is base-a pseudoprime if n is composite and $a^{n-1} = 1 \pmod n$.
- Observation: If n is prime, then n is base-a pseudoprime for every $a \in \mathbf{Z}_n^+$ (because of Fermat's theorem).
- Observation: If for some $a \in \mathbf{Z}_n^+$, we detect that n is not base-a pseudoprime, then n must be composite.

Primality Testing (V)

- Idea: Test whether n is prime or composite, by checking whether n is base-a pseudoprime for $a = 2$.
- Observation: A randomly chosen 1024-bit number that is base-2 pseudoprime has a chance of less than 10^{-41} of actually being composite.
- Idea: Test whether n is prime or composite, by checking whether n is base-a pseudoprime for many $a \in \mathbf{Z}_n^+$.
- Problem: There are some (very rare) composite n, such that for all $a \in \mathbf{Z}_n^+$, we have that n is base-a pseudoprime. These numbers are called Carmichael numbers.

Primality Testing (VI)

- Miller-Rabin Primality-Test:
  1. Try $s \geq 1$ randomly chosen base values $a \in \mathbf{Z}_n^+$.
  2. While computing the modular powers $a^{n-1} \pmod n$, check whether a nontrivial square root of 1 is detected.
- Observation: The Miller-Rabin procedure uses the representation $n-1 = 2^t u$, where $t \geq 1$ and u is odd. Therefore, $a^{n-1} = (a^u)^{2^t} \pmod n$, so that $a^{n-1} \pmod n$ is computed by first computing $a^u \pmod n$, and then squaring the result t times successively.

Primality Testing (VII)

- Miller-Rabin Algorithm:

```
Witness(a, n)
1  Let n-1 = 2^t u , where t ≥ 1 and u is odd
2  x_0 ← Modular-Exponentiation(a, u, n)
3  for i ← 1 to t do
4      x_i ← (x_{i-1})^2 mod n
5      if (x_i = 1) and (x_{i-1} ≠ 1) and (x_{i-1} ≠ n-1)
6          then return TRUE
7  if (x_t ≠ 1)
8     then return TRUE
9  return FALSE
```


Primality Testing (VIII)

- Observation: Procedure Witness(a, n) returns TRUE if it clearly detects that a is a witness to non-primality of n. It returns FALSE if it were not able to use a to find a proof to non-primality of n (which doesn't mean that n is prime).

```
Miller-Rabin(n, s)
1  for j ← 1 to s do
2      a ← Random(1, n-1)
3      if Witness(a, n)
4          then return COMPOSITE    /// definitely
5  return PRIME                     /// almost surely
```


Primality Testing (IX)

- Theorem: If n is an odd composite number, then the number of witnesses to the compositeness of n is at least $(n-1)/2$.
- Proof: (omitted)
- Theorem: For any odd integer $n \geq 2$ and any positive integer s, the probability that Miller-Rabin(n, s) errs is at most 2^{-s}.
- Proof: Based on the previous theorem and the trial of s independent tests of witnesses.

Primality Testing (X)

- Observation: Running time of one call to Witness(a, n) is $O(\log n)$ word-steps (multiplications, additions, etc). Typical range of n is between 512-bit numbers to 2048-bit numbers.
- Observation: Running time of one call to Miller-Rabin(n, s) is $O(s)$ steps of getting random base a and calling Witness(a, n). Typical values of s are between 10 and 100.
- Observation: Finding a prime number (with a very small probability of error) near a given number n requires $O(\ln n)$ trials of calling the Miller-Rabin(x, s) routine with different values of x near n, until one call returns the result PRIME.

Public-Key Cryptography II

Prof. Shlomo Kipnis
December 3, 2007

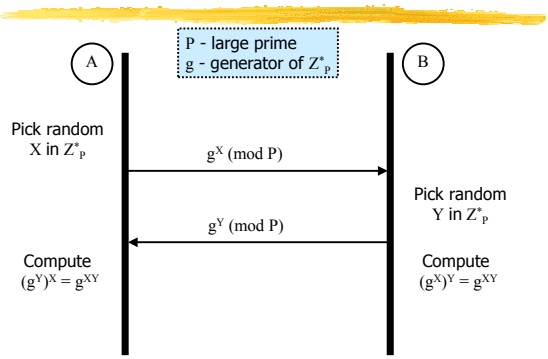
Public-Key Schemes

- Diffie-Hellman Key Exchange
- El-Gamal Encryption
- El-Gamal Signature
- Digital Signature Standard
- RSA encryption and signature

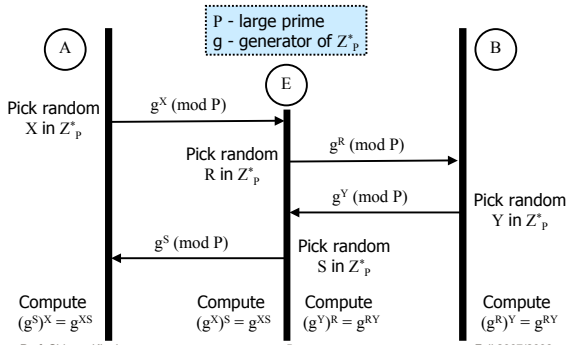
Diffie-Hellman Key Exchange (I)

- Large prime P is known to the world
- Generator g of Z^*_P is known to the world
- A and B do not share any secret value
- The D-H Protocol:
  - A picks at random a number $X \in \{1, 2, \ldots, P-1\}$ and sends to B the value $g^X \pmod P$
  - B picks at random a number $Y \in \{1, 2, \ldots, P-1\}$ and sends to A the value $g^Y \pmod P$
  - A computes $(g^Y)^X = g^{XY} \pmod P$
  - B computes $(g^X)^Y = g^{XY} \pmod P$
  - A and B now share the secret value $g^{XY} \pmod P$

Diffie-Hellman Key Exchange (II)



Diffie-Hellman – Man-in-the-Middle



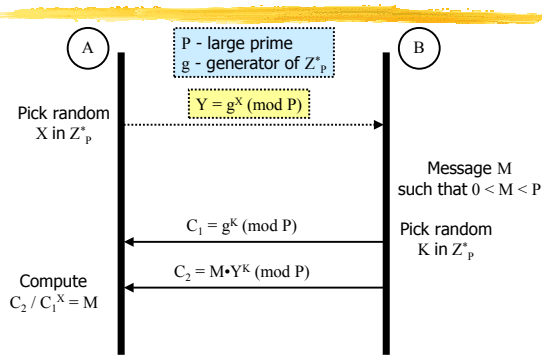
Working with Diffie-Hellman Protocol

- When opponent is assumed to be passive (it can only eavesdrop but it cannot change messages):
  - Protected line
  - Eye-sight line
- With Additional authentication between A and B:
  - Authenticated D-H using digital signatures

El-Gamal Encryption (I)

- Large prime P is known to the world
- Generator g of Z^*_P is known to the world
- A picks random Private Key: $X \in \{1, 2, \ldots, P-1\}$
  A publishes Public Key: $Y = g^X \pmod P$
- B wishes to encrypt for A message M such that $0<M<P$:
  - B picks at random $K \in \{1, 2, \ldots, P-1\}$
  - B sends to A two values:
  - $C_1 = g^K \pmod P$ and $C_2 = M \bullet Y^K \pmod P$
- A computes $C_2 / C_1^X = M \pmod P$

El-Gamal Encryption (II)



El-Gamal Signatures (I)

- Large prime P is known to the world
- Generator g of Z^*_P is known to the world
- A picks random Private Key: $X \in \{1, 2, \ldots, P-1\}$
  A publishes Public Key: $Y = g^X \pmod P$
- A wishes to sign message M such that $0<M<P-1$:
  - A picks random $K \in Z^*_P$ that is relatively prime to $P-1$
  - A computes $a = g^K \pmod P$
  - A computes $b = K^{-1} \bullet (M - X \bullet a) \pmod{(P-1)}$
  - A sends the message M and the signature $S = (a, b)$
- B verifies signature by checking if $Y^a \bullet a^b = g^M \pmod P$

El-Gamal Signatures (II)

- Signature is valid since:
  $Y^a \bullet a^b = g^M \pmod P$ iff $b = K^{-1} \bullet (M - X \bullet a) \pmod{(P-1)}$
- Signature is hard to forge since key K is random and unknown to the attacker.
- The scheme generates a per-message randomized signature (using the random key K)

Digital Signature Standard (I)

- DSS – Digital Signature Standard (mid 1990's)
- Variation of El-Gamal and Schnorr signature schemes
- DSS can be used only for signatures and not for encryption or key-exchange
- Uses the SHA-1 hash function
- Uses a random per-message mask (key)

Digital Signature Standard (II)

- DSS parameters:
  - Random prime q (160 bits)
  - Prime p (between 512 and 1024 bits), such that q divides $p-1$
  - Number $g = h^{(p-1)/q} \pmod p$, where h is a number between 1 and $p-1$, such that g is greater than 1
- Private Key: Random X such that $1< X < q$
- Public Key: $Y = g^X \pmod P$
- Random Per-Message Mask: $1 < K < q$

Digital Signature Standard (III)

- Signature:
  - Given message M (any length)
  - Compute $H(M)$, where H is the hash function SHA-1
  - Compute $r = (g^K \pmod p) \pmod q$
  - Compute $s = (K^{-1} \bullet (H(M) + X \bullet r)) \pmod q$
  - Signature of M is the pair (r, s)
  - Send message M of any length and the signature (r, s) of length 320 bits

Digital Signature Standard (IV)

- Verification:
  - Received message M'
  - Received signature (r', s')
  - Compute $H(M')$, where H is the hash function SHA-1
  - Compute $w = (s')^{-1} \pmod q$
  - Compute $u_1 = H(M') \bullet w \pmod q$
  - Compute $u_2 = (r') \bullet w \pmod q$
  - Compute $v = ((g^{u_1} \bullet Y^{u_2}) \pmod p) \pmod q$
  - Test whether $v = r'$

RSA Key Generation

- Pick two large primes – p and q – about 500 bits each
- Compute – $N = p \bullet q$ – result will be about 1000 bits
- Compute $\Phi(N) = (p-1)(q-1)$
- Pick e relatively prime to $\Phi(N)$ – that is $GCD(e,\Phi(N)) = 1$
- Calculate d inverse of e modulo $\Phi(N)$:
  (that is – calculate d such that $d \bullet e = 1 + k \bullet \Phi(N)$)
- Public Key: (e,N)
- Private Key: (d,N)

RSA Encryption & Decryption

- Encryption:
  - Use public-key (e,N) to encrypt message $M<N$:
    $C = M^e \pmod N$
  - Send only C
- Decryption:
  - Use private-key (d,N) to decrypt cipher $C<N$:
    $M = C^d \pmod N$

RSA Signature & Verification

- Signature:
  - Use private-key (d,N) to sign message $M<N$:
    $S = M^d \pmod N$
  - Send both M and S
- Verification:
  - Use public-key (e,N) to check signature S on message M:
    check whether $S^e = M \pmod N$

RSA Key-Exchange

- A has private key (d_A,N_A) and public key (e_A,N_A)
- B has private key (d_B,N_B) and public key (e_B,N_B)
- A and B wish to create a new symmetric key K:
  - A sends an encrypted random R_1 to B: $(R_1)^{e_B} \pmod{N_B}$
  - B sends an encrypted random R_2 to A: $(R_2)^{e_A} \pmod{N_A}$
  - A recovers R_2 and computes: $K = H(R_1 \oplus R_2)$
  - B recovers R_1 and computes: $K = H(R_1 \oplus R_2)$

Why RSA Works

- Need to show that $(M^e)^d = M^{ed} = M \pmod N$
- Consider modulo P:
  - $M^{ed} = M^{k\Phi(N)+1} \pmod P$
  - $= M \cdot M^{k\Phi(N)} \pmod P$
  - $= M \cdot M^{k(p-1)(q-1)} \pmod P$
  - $= M \cdot (M^{(p-1)})^{k(q-1)} \pmod P$
  - $= M \cdot (1)^{k(q-1)} \pmod P$
  - $= M \pmod p$
- Same argument holds modulo q
- Chinese Remainders Theorem – argument holds for N

RSA Key Generation Considerations

- Primes p and q should be of about the same length (about half the length of N).
- Primes p and q should be unrelated.
- Public exponent e can be small. This will increase the efficiency of encryption and of signature verification:
  - with $e = 3$ encryption will be 2 multiplications
  - with $e = 2^{16} + 1$ encryption will be 17 multiplications
- Private exponent d must be large to disallow searching on small values.
- Use different p, q, N, e, d for different users.

RSA Encryption Considerations

- Message should not be small:
  - M=1 gives C=1
  - Small M and small e can produce C that is less than the modulo N. In this case the attacker can compute the e-th root of C to reveal M
- Message should be random:
  - To avoid identifying old messages from their ciphers
  - To encrypt differently for different users
  - To overcome message size issues

RSA Multiplicative Nature

- RSA has a multiplicative nature:
  - If messages M_1 and M_2 encrypt to ciphers C_1 and C_2, then the message $M_1^i \cdot M_2^j \pmod N$ encrypts to the cipher $C_1^i \cdot C_2^j \pmod N$
  - If the signatures of messages M_1 and M_2 are S_1 and S_2, then the signature of message $M_1^i \cdot M_2^j \pmod N$ is $S_1^i \cdot S_2^j \pmod N$
- The multiplicative nature of RSA is problematic for applications such as encryption and signature
- The multiplicative nature of RSA is exploited by some applications such as anonymous cash, voting, etc.

RSA Blinding

- Signing an unknown message:
  - Having someone sign some message M without letting him know what he signs:
    - pick a random number r
    - compute $b = r^e \pmod N$
    - compose new (random) message $X = M \cdot b \pmod N$
    - let the signature of X be $Y = X^d \pmod N$
    - compute signature S of M as $S = Y / r \pmod N$
- Useful in anonymous cash, voting, etc.
- BUT - can also get the decryption of some message by blindly signing a "random" text

RSA Padding

- Pad the message M is a particular frame with:
  - fixed fields to identify the message
  - random fields to randomize the operation
- Padding the message:
  - passes distinguishability tests
  - prevents detection of repeated messages
  - allows using small messages M
  - allows using small public exponents e

PKCS #1 Version 1.5

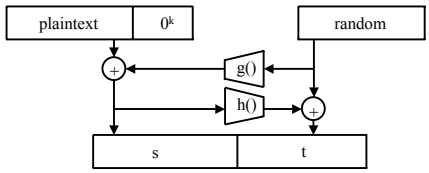
- PKCS – Public Key Cryptographic Standards – define how to use the RSA schemes properly
- PKCS #1 version 1.5:

00	02	padding	00	message

- First two bytes are 0002
- Next eight bytes are random pad (not 0)
- Used in SSL and SSH
- Still subject to chosen plaintext attack (about 10^6 texts)

PKCS #1 Version 2.0

- PKCS #1 version 2.0:
  - Uses OAEP (Optimal Asymmetric Encryption Padding)
  - Use two cryptographic functions g() and h()
  - Cipher contains "proof" of knowledge of plaintext



Efficiency of Signature Schemes

- All Schemes are computationally intensive
- Always sign (short) hash of the message
- RSA:
  - Key generation is very slow (primes p, q, etc.)
  - Signing is slow (large d)
  - Verifying can be fast (small e)
- DSS:
  - Key generation is reasonable (primes p, q, etc.)
  - Signing is fast (most work can be done in advance)
  - Verifying is slow (many exponentiations)

Summary – Required Key Lengths (est.)

Year	AES	RSA, DH, EG	EC
1990	62	768	120
2000	70	1028	139
2010	78	1369	160
2020	86	1881	188
2030	93	2493	215
2040	101	3214	244

Public-Key Cryptography III

Prof. Shlomo Kipnis
December 5, 2007

Optimizing Cryptographic Algorithms

- Symmetric cryptography is about 100 to 1000 times faster than public-key cryptography
- DH, El-Gamal, DSS, RSA require modular exponentiation (that is, many modular multiplications) of big numbers
- Brute-force approach – use specialized cryptographic hardware accelerators
- Software approach – optimize the algorithms used to perform the arithmetic

Modular Multiplication of Big Numbers

- Need to calculate $r = A \cdot B \pmod N$
- Done in two steps:
  $x = A \cdot B$
  $r = x \bmod N$
- A, B, N may be numbers with thousands of bits
- Computers do not support operations on such big numbers – so we need to develop routines for multiplying and dividing big numbers

Multiplication of Big Numbers

- To calculate $A \cdot B$ – view A and B as arrays of cells (e.g., bytes, shorts, longs)
- Cell multiplication is done using the computer's cell operations – using base $b=2^c$ (where c is the cell size):
  bytes – base 2^8
  shorts – base 2^{16}
  longs – base 2^{32}
- Time: n^2 cell multiplications, where n is the number of cells in A and B. (Note: additions are not counted.)

Karatsuba's Fast Multiplication (I)

- Divide A into high half A_H and low half A_L
- Divide B into high half B_H and low half B_L
- Now, we have:
  $C = A \cdot B = A_H \cdot B_H \cdot 2^k + A_H \cdot B_L \cdot 2^{k/2} + A_L \cdot B_H \cdot 2^{k/2} + A_L \cdot B_L$
  $= A_H \cdot B_H \cdot 2^k + (A_H \cdot B_L + A_L \cdot B_H) \cdot 2^{k/2} + A_L \cdot B_L$
  (which use four half-size multiplications and some additions)
- Note that multiplication of a number by 2^k is only a shift left of the number by k positions

Karatsuba's Fast Multiplication (II)

- But, we also have:
  $A_H \cdot B_L + A_L \cdot B_H = (A_H + A_L) \cdot (B_H + B_L) - A_H \cdot B_H - A_L \cdot B_L$
- Therefore we get:
  $C = A \cdot B = A_H \cdot B_H \cdot 2^k + A_H \cdot B_L \cdot 2^{k/2} + A_L \cdot B_H \cdot 2^{k/2} + A_L \cdot B_L$
  $= A_H \cdot B_H \cdot 2^k + [(A_H + A_L) \cdot (B_H + B_L) - A_H \cdot B_H - A_L \cdot B_L] \cdot 2^{k/2} + A_L \cdot B_L$
  (which use three half-size multiplications and some additions)
- Applying this scheme recursively 3 to 4 times can save between 60% to 70% of the multiplications

Division of Big Numbers (I)

- We need to calculate $Q = U / V$ with remainder r. (Actually, we need to find r and not Q.)
- We shall use cells in base b (as in the multiplication):
  $U = U_1 U_2 \ldots U_n$
  $V = V_1 V_2 \ldots V_n$
  $Q = Q_1 Q_2 \ldots Q_n$
- We will normalize U and V (by left shifts) until we will have $V_1 \geq b/2$ (where b is the base).
- At the end of the division, we will de-normalize as needed.

Division of Big Numbers (II)

- First verify that U (or whatever is left of U) $\geq V$
- Initial assessment for Q_i (marked q below)
  if $(U_1 = V_1)$ $q = b - 1$
  else $q = (U_1 \bullet b + U_2) / V_1$
- Initial correction for q:
  decrement q while $V_2 \bullet q > (U_1 \bullet b + U_2 - q \bullet V_1) \bullet b + U_3$
- Multiply $q \bullet V$ and decrement from U
- A final (-1) correction for q might be needed

Division of Big Numbers (III)

- Time complexity of long division of cells:
  - n^2 operations of cell multiplication
  - n operations of cell division

Optimizing Modular Multiplication

- Use largest cell possible on the computer:
  doubling the cell size ➔ 4 times faster.
- But remember – we need to obtain the full result of the cell multiplication.
  Example: If cell size = 32 bit, we need 64 bits of result.
- Choose a convenient divisor (V_1=1 and V_2=0) if possible.
- There is motivation to code certain parts in assembler.
- Montgomery reduction (not explained here).

Modular Exponentiation (I)

- We want to compute $d = A^B \bmod N$
- Denote the bits of B as $b[k-1]\ b[k-2]\ \ldots\ b[1]\ b[0]$
- Algorithm 1 – Scanning the bits of the exponent from least to most (from right to left).
- Intuition – generate increasing powers of A, and multiply by the relevant powers.
  (Because the bits of B – $b[k-1]\ b[k-2]\ \ldots b[2]\ b[1]\ b[0]$ correspond to the following powers of A – $A^{2^{k-1}}\ A^{2^{k-2}} \ldots A^4\ \ A^2\ \ A$)

Modular Exponentiation (II)

- Code for Algorithm 1:

```
P = A;
d = 1;
for i = 0 to k-1 do
{
   if (b[i] == 1) d = d * P mod N;
   P = P * P mod N;
}
return(d);
```

Modular Exponentiation (III)

- We want to compute $d = A^B \bmod N$
- Denote the bits of B as $b[k-1]\ b[k-2]\ \ldots\ b[1]\ b[0]$
- Algorithm 2 – Scanning the bits of the exponent from most to least (from left to right).
- Intuition – if $d = A^B \bmod N$, then

  $A^{B0} \bmod N = d^2 \bmod N$

  $A^{B1} \bmod N = d^2 \cdot A \bmod N$

Modular Exponentiation (IV)

- Code for Algorithm 2:

```
d = 1;
for i = k-1 downto 0 do
{
   d = d * d mod N;
   if (b[i] == 1) d = d * A mod N;
}
return(d);
```


Modular Exponentiation (V)

- Time analysis of Algorithm 1 and Algorithm 2 shows that both take, on the average, $3k/2$ modular multiplications:
  - There is one modular multiplication in every round of the loop (and there are k iterations of the loop)
  - There is one modular multiplication for every bit of 1 in B (and, on the average, B will have $k/2$ bits of 1)

Optimizing Modular Exponentiation

- If possible – choose convenient parameters:
  - Small exponent
  - Sparse exponent (few bits of 1)
- Two scenarios:
  - Fixed base
  - Fixed exponent

Fixed-Base Exponentiation (I)

- Fixed-base exponentiation – compute $g^B \pmod p$
- Used in Diffie-Hellman, in El-Gamal, and in DSS
- Idea: pre-compute powers of g once and use them later
- Pre-computation (only once):

  For each $i = 0 \ldots k-1$, compute $g[i] = g^{2^i} \bmod p$
- To compute $g^B \pmod p$, where $B = b[k-1]\ b[k-2]\ \ldots\ b[1]\ b[0]$, multiply (in modulo p) all the $g[i]$ for which $b[i]=1$
- Space: k powers of g
- Time: $k/2$ modular multiplications (on average)

Fixed-Base Exponentiation (II)

- One can reduce the number of computations even more at the expense of using more space
- Idea: pre-compute powers of g for k/w windows of w bits
- Pre-computation (only once):

  For each $i = 0 \ldots k/w-1$ compute 2^w powers of g:

  $(g^{2^{iw}})^0, (g^{2^{iw}})^1, (g^{2^{iw}})^2, (g^{2^{iw}})^3, \ldots, (g^{2^{iw}})^{(2^w-1)}$
- To compute $g^B \pmod p$, multiply the relevant powers
- Space: $(k/w) \cdot 2^w$ powers of g
- Time: $(k/w) \cdot (2^w-1)/2^w$ modular multiplications (on average)

Fixed-Exponent Exponentiation (I)

- Fixed-exponent exponentiation – compute $M^B \pmod N$
- Used in RSA
- Idea: Divide B into k/w windows of w bits, and compute the 2^w powers of M once at the beginning:
  For each $i = 0 \ldots 2^w-1$, compute $MP[i] = M^i \bmod N$
- Compute $M^B \pmod p$, where $B = b[k-1]\, b[k-2] \ldots b[1]\, b[0]$, by calculating values of successive w-bit windows in B, and multiplying by the appropriate values of MP

Fixed-Exponent Exponentiation (II)

```
MP[0] = 1;
MP[1] = M;
for i = 2 to 2^w-1 do
    MP[i] = (MP[i-1] * M) mod N;
d = 1;
j = w;
val = 0;
for i = k-1 downto 0 do
{
    d = (d * d) mod N;
    val = (2 * val) + b[i];
    j = j - 1;
    if (j == 0 || i == 0)
    {
          d = d * MP[val] mod N;
          j = w;
          val = 0;
    }
}
```


Fixed-Exponent Exponentiation (III)

- Time: 2^w-2 modular multiplications for pre-computation, and $k + (k/w) \cdot (2^w-1)/2^w$ modular multiplications (on the average) during the exponentiation
- Optimal window size minimizes the expression:
  $(2^w-2) + (k/w) \cdot (2^w-1)/2^w$ (for a given k)
- Example: $k = 1024$
  - $w=4$ => 254 modular multiplications
  - $w=5$ => 229 modular multiplications
  - $w=6$ => 230 modular multiplications
  - $w=7$ => 272 modular multiplications

Fixed-Exponent Exponentiation (IV)

- Improvement: Sliding window method with window of size w-bits
- Idea: Pre-compute the following powers of M:
  $M^3, M^5, M^7 \ldots M^{2^w-1}$
  and process the exponent in windows that start and end with bits of 1

People Authentication I

Prof. Shlomo Kipnis
December 10, 2007

Authentication Objectives

- User identification (name, id, etc.)
- User validation (proof of identity)
- Resource identification (name, address, etc.)
- Resource validation (proof of identity)
- Access control (permission lists)
- Monitoring (online, offline)
- Auditing (logs, books)
- Other . . .

Authentication Considerations

- Accuracy Level
- Validation Time
- Processing Power
- Operating Costs
- System Reliability
- Ease of Forging
- Ease of Use
- Portability
- Transferability
- Revocability

Authentication Methods

1. Memorable Data – something the user knows:
   PIN, passwords, pass phrases, personal information
2. Biometrics – something the user is/has:
   finger prints, palm geometry, retina scan, iris scan, face recognition, signature characteristics, typing characteristics, DNA recognition
3. Tokens – something the user holds:
   ID card, paper card, key, smart card, crypto token

Passwords (I)

- Desirable Properties:
  - Password should be easy to remember
  - Password should be hard to guess
- Problem:
  - Cannot achieve desirable two properties above with (current) humans

Passwords (II)

- Password length at a university (study in early 1990's):

Length	Number	Percentage
1	55	0.4
2	87	0.6
3	212	2
4	449	3
5	1260	9
6	3035	22
7	2917	21
8	5772	42

Passwords (III)

- Common Situation:
  - Short passwords (between 4 and 8 characters)
  - English words (few thousands)
  - Names (few hundreds)
  - Personal data (easy to obtain)
  - Combinations of above (easy to compute)
  - Passwords are written somewhere (easy to find)
  - Some passwords are weak (find weakest point)
  - Password in many systems (break one – break all)

Passwords (IV)

- Attacks on Passwords:
  - Sniffing communication lines
  - Searching for passwords
  - Password guessing
  - Online dictionary attacks
  - Offline password cracking
  - Social engineering

Passwords (V)

- Internet Worm:
  - Created by student in Cornell in 1987
  - Broke to 1000's of computers in several hours
  - Used holes in the email protocol
  - Searched passwords from a 400-word dictionary
  - Used same login on multiple computers
  - Had a bug which brought computers down – it infected same computer many times

Passwords (VI)

- How / Where Passwords are Stored:
  - Not hidden and not access-protected (some PC's)
  - Hidden but not access-protected (some appliances)
  - Access-protected (Unix, NT, other OS)
  - Encrypted (but where is the key stored)
  - Hashed (Unix, NT, other OS)
  - Protected server (distributed systems)

Passwords (VII)

- Unix "salt" Mechanism:
  - When user A opens an account, a password pw_A is defined, and a random 12-bit $salt_A$ is selected
  - The Unix password file stores both $salt_A$ and the value of $h(pw_A, salt_A)$ at the entry for user A
  - When a user attempts to log on as A and types some pw, the system takes $salt_A$ from A's entry in the file, computes $h(pw, salt_A)$, and compares the result to what is stored in the file. A match allows logging on.
  - This scheme makes online dictionary attacks infeasible (since each password can be hashed in $4096 = 2^{12}$ ways). This scheme is not resilient against offline password cracking.

Passwords (VIII)

- Password breaking recipes:
  - Try default passwords used in standard system accounts
  - Exhaustively search all short passwords
  - Try words from online dictionaries
  - Collect and try data that is related to users (user names, family member names, birth dates, identification numbers, etc.)
  - Try common combinations of user data (e.g., reverse writing, adding digits at end of passwords, etc.)
  - Look for written passwords
  - Observe password typing patterns

Passwords (IX)

- Password breaking recipes (continued):
  - Use a Trojan horse to steal users' passwords
  - Eavesdrop to communication lines
  - Get access to passwords files
  - Analyze the (hashed / encrypted) passwords file
  - Get from machine to machine with OS facilities and/or with known passwords
  - Pretend to be a legitimate user and ask the administrator to issue you a new password
  - Pretend to be a legitimate administrator and ask the user to disclose the password

Passwords (X)

- System recommendations:
  - Educate users of importance of password security
  - Monitor user accounts for suspicious behavior
  - Lock account after a number of unsuccessful login attempts
  - Keep password file encrypted or hashed
  - Use password strengthening mechanisms (e.g. Unix salt)
  - Keep password files in secure locations (directories in the file system, special servers, etc.)
  - Request users to change passwords frequently
  - Run password cracking tests and disallow weak passwords
  - Use passwords only as one factor in authentication process

Passwords (XI)

- Password selection recommendations:
  - Use combinations of letters, upper-case, lower-case, digits, other characters
  - Change passwords frequently
  - Use different passwords in different systems
  - Use random passwords (8-10 characters long)
  - Use readable passwords (16-20 characters)
  - Use pass-phrases (30-40 character sentences)

Passwords (XII)

- Password Administration Scenario:
  - Admin: Passwords must be changed every 90 days
  - User: Changes the password to the same password
  - Admin: Check that password is changed to a new one
  - User: Changes the password and changes again to the old one
  - Admin: Tracks last n passwords and checks that password is new
  - User: Changes the password $n+1$ times and returns to old one
  - Admin: Disallows more than one password change per day
  - User: Changes to the same password with $1, 2, 3, \ldots$ at the end
  - Admin: Disallows passwords that are "too similar" to old ones
  - User: Invents a random password and writes it down on paper

Passwords (XIII)

- Summary:
  - Accurate (positive & negative)
  - Secret user data
  - Secret server data
  - Easy to break
  - Simple to operate
  - Portable
  - Transferable
  - Not easily revocable

Biometrics (I)

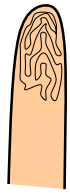
- Biometrics consists of checking physical, biological or physiological properties of a person
- Certain properties are highly unique to each person
- Need to select properties that are:
  - Easy to detect
  - Provide high levels of accuracy
- Need to maintain database of biometrics parameters

Biometrics (II)

- Finger prints
- Palm geometry
- Retina scan
- Iris scan
- Face recognition
- Signature characteristics
- Typing characteristics
- DNA recognition

Biometrics (III)

- Finger Prints:
  - 2-D geometry
  - 3-D geometry
  - Liveliness checks (pulse, temperature, etc)
  - Capacitance / resistance checks



- Relatively accurate
- Needs maintenance
- Acceptance level is increasing

Biometrics (IV)

- Palm Geometry:
  - 2-D geometry
  - Finger lengths
  - Finger widths
  - Gaps between fingers

- Good accuracy
- Low maintenance
- High acceptance

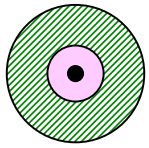
Biometrics (V)

- Retina Scan:
  - 2-D map of blood vessels
  - Blood vessels are warmer than surrounding tissues
  - Detected by IR radiation

- Highly accurate
- Expensive (special equipment)
- Invasive (low acceptance)

Biometrics (VI)

- Iris Scan:
  - 2-D map of iris texture
  - Detection by camera



- Relatively accurate
- Inexpensive
- Non-invasive (high acceptance)

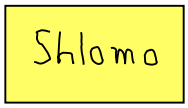
Biometrics (VII)

- Face Recognition:
  - 2-D image
  - Bone-muscle model
  - Under-skin thermal radiation

- Detection by camera / IR
- Good accuracy
- Non-invasive

Biometrics (VIII)

- Signature Characteristics:
  - 2-D image
  - Dynamic signature
  - Online test
  - Speed, pressure, angles, etc.



- Highly accurate
- Low maintenance
- High acceptance

Biometrics (IX)

- Typing Characteristics:
  - Dynamic typing parameters
  - Speed, gaps, letter patterns
  - Online test
  - Could be tacit

- Highly accurate
- Low maintenance
- High acceptance

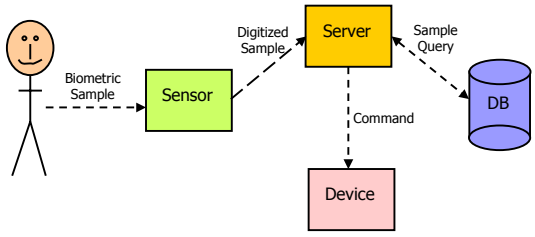
Biometrics (X)

- DNA Recognition:
  - DNA matching against known patterns
  - Sample could be taken from external tissues

- Highly accurate
- Expensive (equipment)
- Conceived as invasive
- Not widely used (yet)

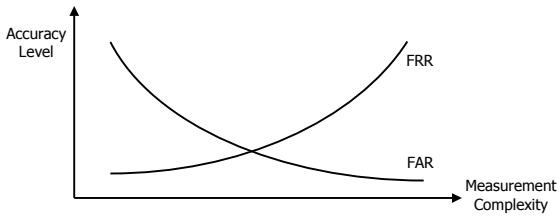
Biometrics (XI)

- Biometric System Components:



Biometrics (XII)

- Accuracy levels:
  - FAR – False Accept Ratio
  - FRR – False Reject Ratio



Biometrics (XIII)

- Biometrics – Summary:
  - Not totally accurate (positive & negative)
  - Private data (not secret data)
  - Easy to steal data
  - Costly to operate
  - Portable
  - Non-transferable
  - Non-revocable

People Authentication II

Prof. Shlomo Kipnis
December 12, 2007

Authentication Tokens

- Authentication Tokens:
  - Mechanical tokens
  - Cryptographic tokens
- Assist in opening / remembering / computing

Mechanical Tokens

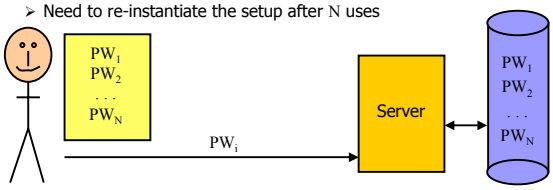
- Mechanical Tokens:
  - Keys (to physical locks)
  - Paper cards (passports, letters, documents, etc.)
  - Plastic cards (ID cards, bank cards, pass cards, etc.)

Cryptographic Tokens

- Cryptographic Tokens:
  - Password list cards
  - Password chain cards
  - Challenge-Response cards
  - One-Time-Password (OTP) cards

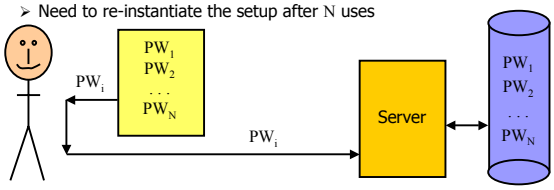
Password List Cards (I)

- Password List – Paper Card:
  - User has paper list of N one-time passwords
  - User sends one password at a time
  - User erases used passwords
  - Server stores password list and compares to received passwords
  - Server disallows passwords that were used before
  - Need to re-instantiate the setup after N uses



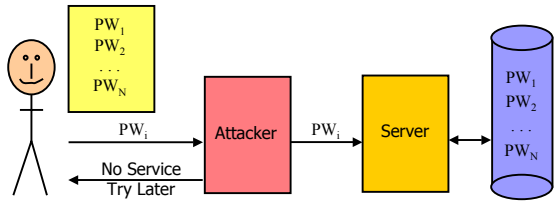
Password List Cards (II)

- Password List – Memory Card:
  - Card keeps N one-time passwords
  - User+Card send one password at a time
  - Card erases used passwords
  - Server stores password list and compares to received passwords
  - Server disallows passwords that were used before
  - Need to re-instantiate the setup after N uses



Password List Cards (III)

- Active "man-in-the-middle" attack:
  - User sends password PW_i to the server
  - Attacker pretends to be the server towards the user – it gets the password PW_i and informs the user that there is no service
  - Attacker later uses password PW_i and pretends to be the user towards the server

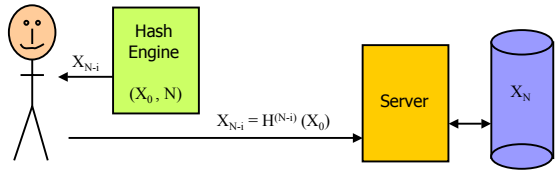


Password List Cards (IV)

- Advantages:
  - Passwords can be arbitrarily strong
  - Passwords are one-time
  - Simple and inexpensive
- Disadvantages:
  - Large storage at the user side
  - Large storage at the server side (for each user)
  - Method is sensitive to the user losing the card
  - Method is sensitive to breaking into the server
  - A "man-in-the-middle" attacker can steal and use passwords

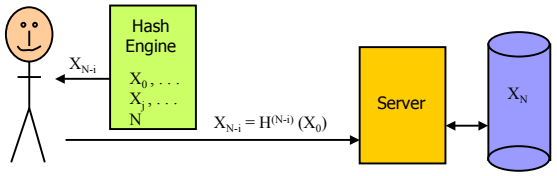
Password Chain Cards (I)

- Password Chain – S/Key:
  - User card stores seed X_0 and length N of a hash chain
  - Server stores end of the hash chain $X_N = H^{(N)}(X_0)$
  - At use i, user gets and submits password $X_{N-i} = H^{(N-i)}(X_0)$
  - Server accepts the user if $X_{N-i+1} = H(X_{N-i})$
  - User card replaces N with $N-1$
  - Server updates end of chain to be X_{N-i} instead of X_{N-i+1}



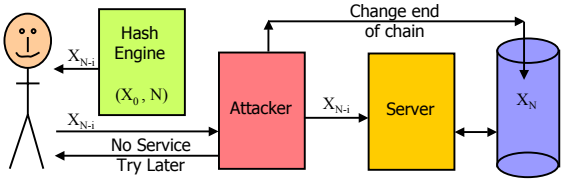
Password Chain Cards (II)

- Password Chain – Improvements:
  - Card can store several intermediate values between seed X_0 and end X_N in order to accelerate the computations.
  - Card can start from X_0 and make N computations to find X_N. While doing so, card can record value of X_j, where $j = 9N/10$, and so about 10% of future computations can start from X_j. This scheme uses less storage and amortizes computation time.



Password Chain Cards (III)

- Active attacks:
  - Man-in-the-middle:
    User sends a password to the server;
    Attacker pretends to be the server and gets the password;
    Attacker later uses the password to impersonate the user.
  - Server penetration:
    Attacker creates a new hash chain from a seed known to it;
    Attacker attempts to "plant" the end of the chain at the server.

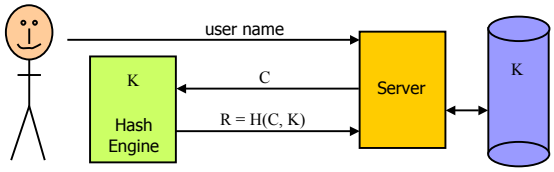


Password Chain Cards (IV)

- Advantages:
  - Passwords are strong
  - Passwords are one-time
  - Small storage at the user side
  - Small storage at the server side (for each user)
  - Method is insensitive to stealing data from the server
- Disadvantages:
  - Requires hardware at the user side
  - Method is sensitive to the user losing the card
  - Method is sensitive to stealing and using passwords
  - Method is sensitive to injecting data at the server

Challenge-Response Cards (I)

- Challenge-Response Card:
  - User activates card
  - User identifies to the server
  - Server sends challenge C to user
  - Card computes response R based on challenge C and secret K
  - Server verifies the response R using challenge C and secret K

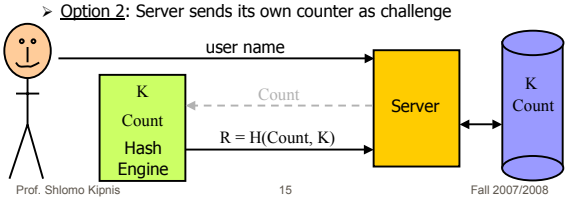


Challenge-Response Cards (II)

- Types of challenges:
  - Sequence number (by user / by server)
  - Time stamp (by user / by server)
  - Random value (by server)
- Properties:
  - User card with simple CPU and small storage
  - Server with strong CPU and small storage per user
  - Server needs to "remember" its challenge to each user

Sequence Challenge Cards (I)

- Sequence Challenge Card:
  - User activates card
  - User identifies to the server
  - Card computes value based on counter C and secret K
  - Server verifies the value using counter C and secret K
  - Option 1: Server maintains counter synchronization with each user
  - Option 2: Server sends its own counter as challenge

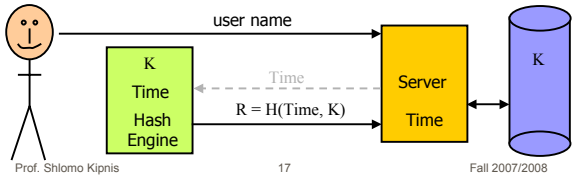


Sequence Challenge Cards (II)

- Advantages:
  - Passwords are strong and one-time
  - Small storage at the user side
  - Small storage at the server side (for each user)
  - Some options may reduce communication load on the server
- Disadvantages:
  - Requires hardware at the user side
  - Method is sensitive to the user losing the card
  - Some options require counter synchronization
  - Some options are sensitive to stealing and using passwords
  - Method is sensitive to injecting data at the server

Time Challenge Cards (I)

- Time Challenge Card:
  - User activates card
  - User identifies to the server
  - Card computes value based on current time T and secret K
  - Server verifies the value using current time T and secret K
  - Option 1: Server maintains clock synchronization with each user
  - Option 2: Server sends its own clock reading as challenge

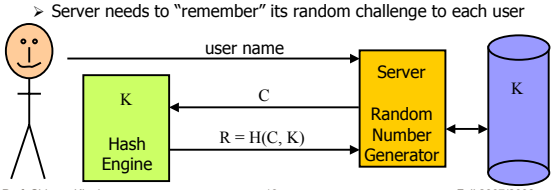


Time Challenge Cards (II)

- Advantages:
  - Passwords are strong and one-time
  - Small storage at the user side
  - Small storage at the server side (for each user)
  - Some options may reduce communication load on the server
- Disadvantages:
  - Requires hardware at the user side
  - Method is sensitive to the user losing the card
  - Some options require time synchronization
  - Some options are sensitive to stealing and using passwords
  - Method is sensitive to injecting data at the server

Random Challenge Cards (I)

- Random Challenge Card:
  - User activates card
  - User identifies to the server
  - Server sends random challenge C to user
  - Card computes response R based on random C and secret K
  - Server verifies the response R using random C and secret K
  - Server needs to "remember" its random challenge to each user

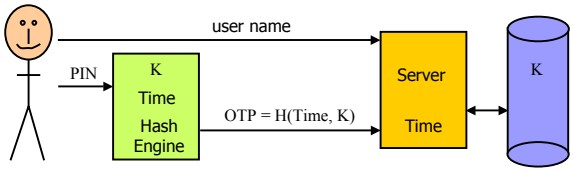


Random Challenge Cards (II)

- Advantages:
  - Passwords are strong and one-time
  - Small storage at the user side
  - Small storage at the server side (for each user)
  - No need to maintain synchronization between server and users
  - Method is resilient against stealing and using passwords
- Disadvantages:
  - Requires hardware at the user side
  - Requires involvement of the server in any authentication
  - Method is sensitive to the user losing the card
  - Method is sensitive to injecting data at the server

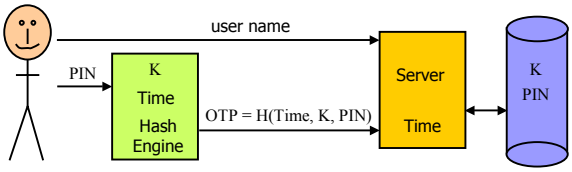
One-Time-Password Cards (I)

- OTP Card – Option 1:
  - User identifies to the server
  - User activates card by entering PIN
  - Card locks after several erroneous attempts of PIN
  - Card computes OTP based on time T and secret K
  - OTP is good only for 60 seconds
  - Server verifies OTP using time T and secret K



One-Time-Password Cards (II)

- OTP Card – Option 2:
  - User identifies to the server
  - User activates card and enters PIN
  - Card computes OTP based on time T, secret K, and user's PIN
  - OTP is good only for 60 seconds
  - Server verifies OTP using time T, secret K, and user's PIN
  - Server disqualifies card after several erroneous attempts



One-Time-Password Cards (III)

- Advantages:
  - Passwords are strong and one-time
  - Small storage at the user side
  - Small storage at the server side (for each user)
  - Method reduces communication load on the server
  - Method is insensitive to the user losing the card
  - Method is resilient against stealing and using passwords
- Disadvantages:
  - Requires expensive hardware at the user side
  - Method requires time synchronization between server and users
  - Method is sensitive to injecting data at the server

Crypto Tokens

- Summary:
  - Highly accurate (positive & negative)
  - Secret data / code
  - Some tokens are hard to forge
  - Relatively easy to use
  - Operation may be costly
  - Portable
  - Some are transferable
  - Some are revocable

Authentication Protocols I

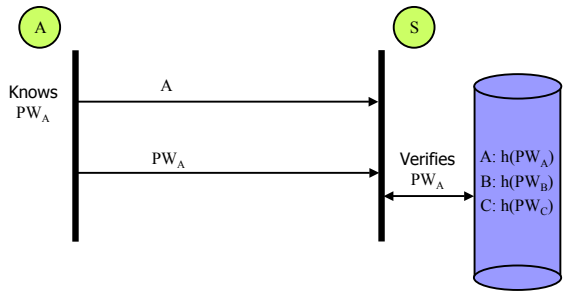
Prof. Shlomo Kipnis
December 17, 2007

One-Sided Authentication Protocols

- Typically, clients authenticate to servers with:
  - Password protocols
  - Secret-key protocols
- Typically, servers authenticate to clients or to other servers with:
  - Secret-key protocols
  - Public-key protocols
  - Zero-Knowledge protocols

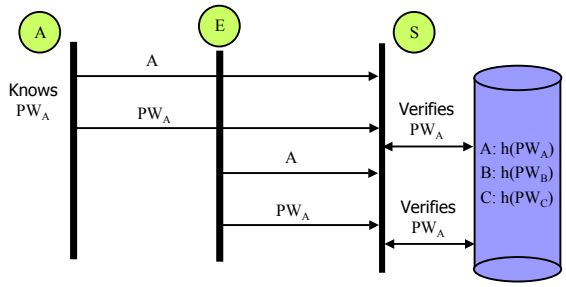
Password Protocols (I)

- A typical password authentication protocol:



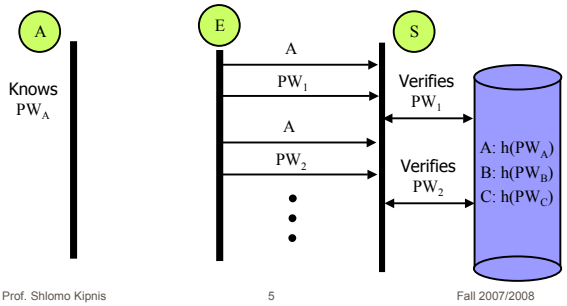
Password Protocols (II)

- Password sniffing and replay attack:



Password Protocols (III)

- Password guessing / cracking attack:

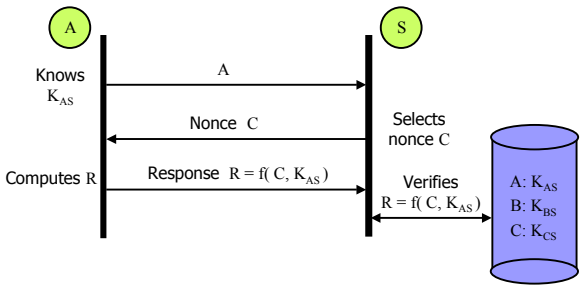


Password Protocols (IV)

- Advantages:
  - Simple implementations
  - No need for hardware at the client
  - Work with humans
  - Useful over secure channels
- Disadvantages:
  - Password sniffing and replay attacks
  - Password guessing and cracking attacks
  - Dictionary attacks on password DB

Secret-Key Protocols (I)

- A typical secret-key protocol:

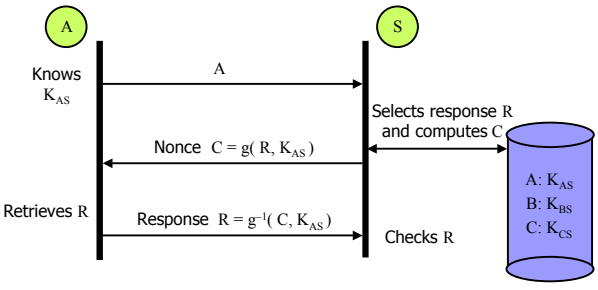


Secret-Key Protocols (II)

- Nonce C can be based on:
  - Counter
  - Time stamp
  - Random value
- Function f can be:
  - Hash function (usually fast)
  - MAC function (usually fast)
  - Encryption function (usually slower than hash or MAC)

Secret-Key Protocols (III)

- Another secret-key protocol:

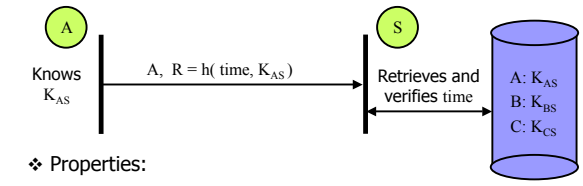


Secret-Key Protocols (IV)

- Function g should be invertible – an encryption function might be the preferred choice
- Nonce C is computed from a response R that can be recognized by the client A and by the server S. The response R should have a particular format and a server-challenge field (such as a time stamp)
- Protocol can achieve bi-directional authentication, since the client can check that the response R indeed has the right format and that the server-challenge is acceptable (such as that the time stamp is correct)

Secret-Key Protocols (V)

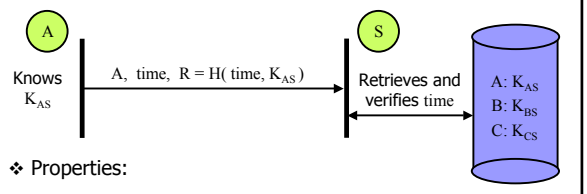
- A 1-message authentication protocol:



- Properties:
  - Uses an invertible function h
  - Requires good clock synchronization
  - Allows attacks within time window
  - Server does not need to keep state for authentication

Secret-Key Protocols (VI)

- Another 1-message authentication protocol:



- Properties:
  - Uses a hash function H
  - Requires good clock synchronization
  - Can use very fine time window to prevent attacks
  - Server does not need to keep state for authentication

Secret-Key Protocols (VII)

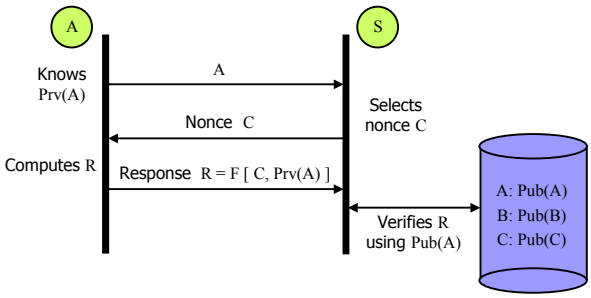
- Advantages:
  - Resilient against sniffing and replay attacks
  - Resilient against key guessing and cracking
  - Efficient implementations of symmetric crypto
  - Some versions can achieve bi-directional authentication
- Disadvantages:
  - Requires hardware at client and server
  - Might not be strong if keys are derived from passwords
  - Allows getting encryptions or decryptions of chosen texts
  - Possibility of attacks on key DB at the server

Secret-Key Protocols (VIII)

- Usage Notes:
  - Need to establish long-term secret keys K_{AS} at the client and the server
  - A random number is best as nonce, but they are expensive to generate (especially at busy servers)
  - A time stamp as nonce is a good choice if clocks are highly synchronized and if there are secure ways of re-synchronizing clients and servers
  - A counter as nonce requires a non-volatile memory at the server (so that it does not re-start its counter every time it is shut down and restarted)

Public-Key Protocols (I)

- A typical public-key protocol:

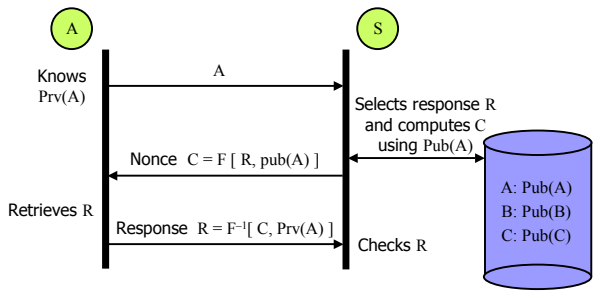


Public-Key Protocols (II)

- Nonce C can be based on:
  - Counter
  - Time stamp
  - Random value
- Function F should be similar to a "signature" (as it uses the client's private key on some text)
- The server verifies the "signature" using the client's public key

Public-Key Protocols (III)

- Another public-key protocol:



Public-Key Protocols (IV)

- Function F should be similar to an "encryption" function (as it uses the client's public key on some text)
- The client decrypts the "encrypted" challenge using its private key
- There is no short way of achieving a bi-directional authentication in these protocols since everyone knows the public key of the client

Public-Key Protocols (V)

- Advantages:
  - Resilient against sniffing and replay attacks
  - Resilient against key guessing and cracking
  - Servers do not store sensitive data
- Disadvantages:
  - Slower implementations of asymmetric crypto
  - May require specialized hardware
  - Allows getting "decryptions" or "signatures" of chosen texts
  - Require using special frames (such as PKCS)
  - Possibility of learning something about private keys

Public-Key Protocols (VI)

- Problems:
  - Computing with the private key on values supplied by another party may be interpreted as a "signature"
  - Public-key system may leak some information on the private key in the computation:
    - Example: The Discrete Log Problem may leak low order bits of the private key
  - Inefficient for authentication purposes due to the overhead needed in padding and wrapping the data to be "signed"

Zero-Knowledge Protocols (I)

- Party A has private-key $Prv(A)$
- World knows public-key $Pub(A)$
- Authentication protocol is based on $Prv(A)$ and $Pub(A)$
- Requirements:
  - Completeness – A will always succeed in proving its identity
  - Soundness – an imposter will fail with probability approaching 1
  - Zero-Knowledge – an eavesdropper will not gain any knowledge about $Prv(A)$

Zero-Knowledge Protocols (II)

- Zero-Knowledge Protocol based on Discrete-Log Problem
- Infrastructure:
  - Known to the World: large prime p
  - Known to the World: generator g of Z^*_p
  - Private Key: A picks random $x \in \{1, 2, \ldots, p-1\}$
  - Public Key: A publishes $y = g^x \pmod p$

Zero-Knowledge Protocols (III)

- Every time that A wants to prove its identity to B:

  Repeat the following process for t rounds:

  1. Commit: A picks random value $k \in \{1, 2, \ldots, p-1\}$ and sends to B the value $s = g^k \pmod p$
  2. Challenge: B sends to A random challenge $c \in \{0, 1\}$
  3. Response: A sends to B response $r = k + c \cdot x \pmod{(p-1)}$
  4. Verify: B checks whether $g^r = s \cdot y^c \pmod p$

Zero-Knowledge Protocols (IV)

- Proof of the "completeness" property:
  - A knows the private key x. So, it can always commit on some random value k, compute the random value s, get a random challenge c, and respond with correct value r that will convince B that A passes the verification test.
  - Note: In the case of a challenge $c = 0$, the response r does not involve the private key x.
  - Note: In the case of a challenge $c = 1$, the response r involves the private key x in a way that an eavesdropper cannot learn the value of x since a random k is added to x to produce the response r.

Zero-Knowledge Protocols (V)

- Proof of the "soundness" property:
  - Imposter E does not know x. The protocol requires E to commit to some value s before getting a challenge c. E has two options.
  - Option 1: E prepares to a challenge of $c = 0$.
    - E chooses random k and commits to the value $s = g^k \pmod p$.
    - If c is indeed 0, then E replies with $r = k$ and passes the test.
    - But if c is 1, then E does not know the correct value $r = k + x$.
  - Option 2: E prepares to a challenge of $c = 1$.
    - E chooses random k and commits to the value $s = g^k / y \pmod p$.
    - If c is indeed 1, then E replies with $r = k$ and passes the test.
    - But if c is 0, then E does not know the correct value $r = k - x$.
  - In each of the t verification rounds, E will fail with probability ½. Therefore, E will fail in the verification with probability $1 - (½)^t$.

Zero-Knowledge Protocols (VI)

- Proof of the "zero knowledge" property:
  - The three values $< s, c, r >$, which are exchanged in each round of the protocol, are uniformly distributed over the three spaces $< Z^*_p, \{0,1\}, Z^*_p >$. In addition, these three values satisfy the relationship $g^r = s \cdot y^c \pmod p$ for fixed p, g, and y.
  - However, one can generate any number of such triplets without knowing the value of x as follows:
    - pick a random r in Z^*_p
    - pick a random c in $\{0,1\}$
    - compute $s = g^r / y^c \pmod p$
  - This means that seeing any number of "correct" such triplets $< s, c, r >$ does not leak any knowledge of x.

Zero-Knowledge Protocols (VII)

- Improvements to the Zero-Knowledge Protocol based on the Discrete-Log Problem:
  - Selecting the value of c in the protocol to be between 0 and 2^n-1 will reduce the success probability of an imposter from ½ to $(½)^n$. This allows cutting the number of rounds by a factor of n while attaining the same success probability of a n imposter.
  - In each round, the party wishing to prove its identity picks n independent random values: $k_1, k_2, \ldots, k_n$ and commits to n independent values: $s_1, s_2, \ldots, s_n$. The verifying party provides n independent challenges: $c_1, c_2, \ldots, c_n$, and then the proving party responds with n responses: $r_1, r_2, \ldots, r_n$. This allows cutting the number of rounds by a factor of n.

Zero-Knowledge Protocols (VIII)

- Transforming a ZK Proof into a ZK Signature:
  - Pick t random values $k_1, k_2, \ldots, k_t$ and commit to t random values $s_1, s_2, \ldots, s_t$ (where $s_i = g^{k_i} \pmod p$)
  - Concatenate the t commitment values $s_1, s_2, \ldots, s_t$ to the message M and call the result M^*
  - Take a hash of M^* and use the bits of $H(M^*)$ as the challenge bits $c_1, c_2, \ldots, c_t$
  - The signature will consist of:
    - the message M
    - the t commitments $s_1, s_2, \ldots, s_t$
    - the t responses $r_1, r_2, \ldots, r_t$, for $r_i = k_i + c_i \cdot x \pmod{(p-1)}$

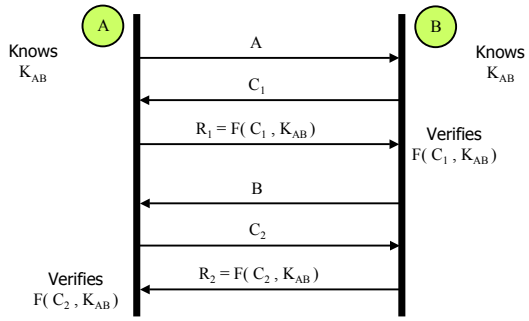
Authentication Protocols II

Prof. Shlomo Kipnis
December 19, 2007

Mutual Authentication Protocols

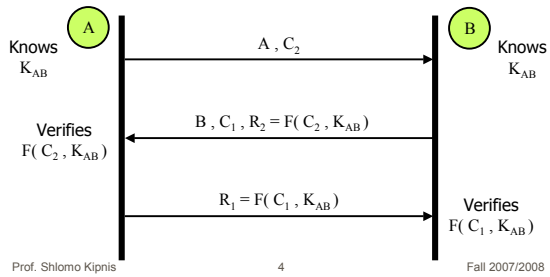
- Mutual authentication is done when both the client needs to authenticate to the server and the server needs to authenticate to the client
- Each side can use a different method to authenticate:
  - Password protocols
  - Secret-key protocols
  - Public-key protocols
- Usually, the more sensitive party will request that the less sensitive party will authenticate first

Symmetric Mutual Authentication (I)



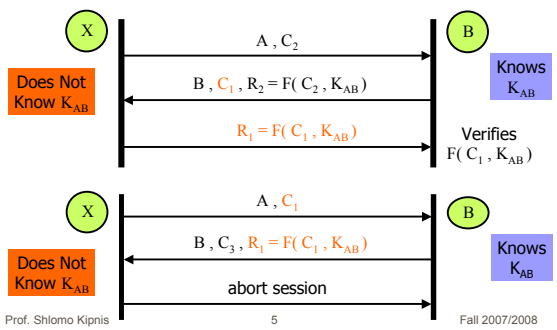
Symmetric Mutual Authentication (II)

- Optimized (3 messages instead of 6 messages), while reversing the order of the authentication:



Symmetric Mutual Authentication (III)

- Reflection attack (with 2 sessions) on optimized protocol:

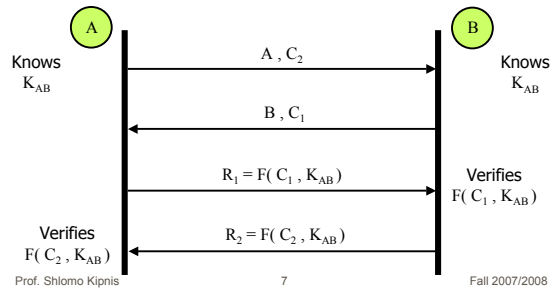


Symmetric Mutual Authentication (IV)

- Reflection attack (with 2 sessions) on optimized protocol:
  - A and B know K_{AB} and X tries to impersonate as A
  - In session #1:
    - X sends to B challenge C_2
    - B sends to X challenge C_1
  - In session #2:
    - X sends to B challenge C_1
    - B replies to X with $R_1 = F(C_1, K_{AB})$
  - Back in session #1:
    - X sends to B the value $R_1 = F(C_1, K_{AB})$ from session #2
    - B verifies $F(C_1, K_{AB})$ and believes X to be A

Symmetric Mutual Authentication (V)

❖ Optimized (4 messages instead of 6 messages), while preserving the order of the authentication:

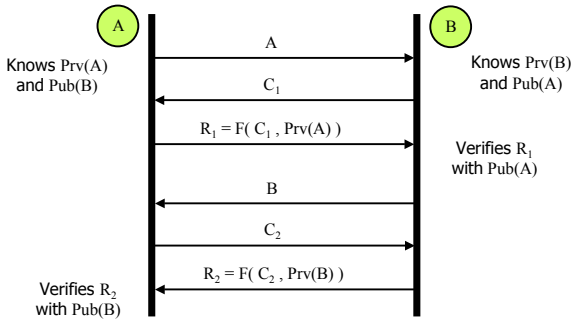


Symmetric Mutual Authentication (VI)

❖ Ways to overcome reflection attacks:

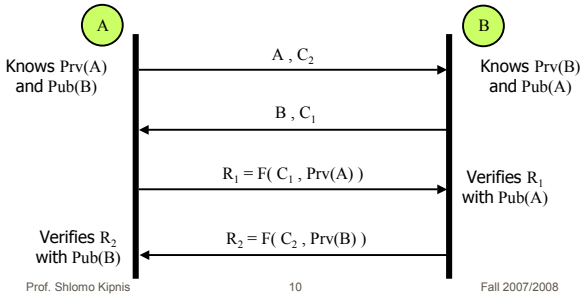
- Preserve order in the authentication process: the client should finish its authentication before the server finishes its authentication
- Use different types of messages: "from client to server" and "from server to client"
- Introduce an additional self-nonce before applying the authentication function:
  - A sends to B nonce C_1
  - B replies to A with N_1 and with $R_1 = F(C_1 , N_1, K_{AB})$, where N_1 is a new nonce introduced by B

Asymmetric Mutual Authentication (I)



Asymmetric Mutual Authentication (II)

❖ Optimized (4 messages instead of 6 messages), while preserving the order of authentication:



Session Key Establishment (I)

❖ Principles:

- Use the long-term (symmetric or asymmetric) keys only infrequently to authenticate the parties and to establish session keys
- Session keys will be used to encrypt and authenticate data transmitted between the two parties after initial authentication of the parties
- The establishment of a session key should be tied to the authentication protocol between the parties

Session Key Establishment (II)

❖ Creating a session key from values exchanged in a secret-key authentication:

- Use the values of the nonce C_1, the nonce C_2, and the long-term key K_{AB} to create the session key K
- Use a function $K = g(K_{AB} , C_1 , C_2)$ that cannot be easily predicted or injected into the protocols
- Such schemes do not protect conversations that were held in the past and/or conversations that will be held in the future (if the attacker manages to break into party A or party B and steal the key K_{AB})

Session Key Establishment (III)

- Protecting past conversations with secret keys:
  - Change the secret key shared between party A and party B on a regular basis at predetermined time periods
  - Let $K_{AB}[i]$ be the i-th secret key
  - Option 1: At the end of time period i, send $K_{AB}[i+1]$ encrypted with $K_{AB}[i]$ from one party to the other
  - Option 2: At the end of time period i, compute $K_{AB}[i+1]$ from $K_{AB}[i]$ using a one-way function at both parties
  - If key $K_{AB}[i]$ is revealed by an attacker, past conversations that were protected with $K_{AB}[j]$, for $j < i$, cannot be broken
  - If key $K_{AB}[i]$ is revealed by an attacker, future conversations that will be protected with $K_{AB}[j]$, for $j \geq i$, can be broken

Session Key Establishment (IV)

- Protecting past & future conversations with secret keys:
  - Change the secret key shared between party A and party B on a regular basis at predetermined time periods
  - Let $K_{AB}[i]$ be the i-th secret key
  - There is no relation between keys $K_{AB}[i]$ and $K_{AB}[j]$, for $j \neq i$
  - Option: At the end of time period i, physically install new key $K_{AB}[i+1]$ at parties A and B
  - If key $K_{AB}[i]$ is revealed by an attacker, past conversations that were protected with $K_{AB}[j]$, for $j < i$, cannot be broken
  - If key $K_{AB}[i]$ is revealed by an attacker, future conversations that will be protected with $K_{AB}[j]$, for $j \geq i$, cannot be broken

Session Key Establishment (V)

- Creating a session key from values exchanged in a public-key authentication:
  - Option 1: Party A invents a session key K and sends it to party B encrypted with B's public key. After the session, both parties erase K.

    Problem: Attacker E can impersonate party A.
  - Option 2: Party A invents a session key K and sends it to party B encrypted with B's public key and signed with A's private key. After the session, both parties erase K.

    Problem: Attacker E that breaks into party B will be able to understand past conversations.

Session Key Establishment (VI)

- Creating a session key from values exchanged in a public-key authentication (continued):
  - Option 3: Party A invents a random number C_1 and sends it to party B encrypted with B's public key. Party B invents a random number C_2 and sends it to party A encrypted with A's public key. Both parties compute session key $K = C_1 \oplus C_2$. In this scheme there is no need to sign the messages. After the session, both parties erase K.

    Problem: Attacker E that breaks into both party A and party B will be able to understand past conversations.

Session Key Establishment (VII)

- Creating a session key from values exchanged in a public-key authentication (continued):
  - Option 4: Authenticated Diffie-Hellman Key Exchange.
    There is a large prime p and a generator g of Z^*_p.
    Party A picks a random X and sends to B the value $g^X \pmod p$.
    Party B picks a random Y and sends to A the value $g^Y \pmod p$.
    Both parties compute session key $K = g^{XY} \pmod p$.
    After the session, both parties erase X, Y and K.
    Note: The D-H key shares may need to be signed.

    Property: Session keys are always new. Attacker E that breaks into both party A and party B cannot learn anything about past conversations and about future conversations.

Session Key Establishment (VIII)

- Creating a session key from values exchanged in a one-sided public-key authentication:
  - Assume that only party B has public/private keys.
  - Option 1: Party A invents a session key K and sends it to party B encrypted with B's public key. After the session both parties erase K.
  - Option 2: Party A and party B perform a D-H key exchange. Only party B signs its D-H key share.
  - Property: In both options above, only party B authenticates to party A. However, party B is assured that it is speaking with the same party all the time. Authentication of party A can be done later (for instance, by using a password protocol).

Protecting User Passwords (I)

- Schemes for protecting user passwords between user A and server B:
  - Option 1: Party A and party B perform an unauthenticated D-H key exchange. Party A then sends its password encrypted with the key $K = g^{XY} \pmod p$ to party B.

    Problem: Attacker can impersonate B towards Party A.
  - Option 2: Party A and party B perform a D-H key exchange in which only party B authenticates itself (or its D-H key share). Party A then sends its password to party B encrypted with the key $K = g^{XY} \pmod p$.

    Problem: Party A needs to be able to authenticate party B.

Protecting User Passwords (II)

- Schemes for protecting user passwords between user A and server B (continued):
  - Option 3: Party B sends nonce C to party A. Party A encrypts nonce C under a key derived from its password.

    Problem: Attacker can launch off-line dictionary attacks on the encrypted password.
  - Option 4: Party A uses one-time passwords derived from a password chain with a seed $x_1 = h(pw)$. Party B keeps the end of the hash chain.

    Problem: Attacker can steal passwords by impersonating the server (party B).

Protecting User Passwords (III)

- Schemes for protecting user passwords between user A and server B (continued):
  - Option 5: Strong Password Protocol.
    Party A and party B both know $w = h(pw)$.
    Party A sends $E_w(g^X \pmod p)$ to party B.
    Party B sends $E_w(g^Y \pmod p)$ to party A.
    Both parties compute key $K = g^{XY} \pmod p$.
    Party A sends $E_K(C_1)$ to party B.
    Party B sends $E_K(C_2\ C_1)$ to party A.
    Party A sends $E_K(C_2)$ to party B.
  - Attacker cannot identify password w since, for each value of w yields possible values of $g^X \pmod p$ and $g^Y \pmod p$.

Authentication Principles (I)

- Use message frames to distinguish directionality and to distinguish functionality
- Add type and direction information to packets
- Use different functions for party authentication, message authentication, encryption, signatures, etc.
- Use only strong and well tested functions
- Use only proven (tested) authentication protocols
- Authenticate each party separately in the protocols
- Derive communication keys immediately following the authentication

Authentication Principles (II)

- Expose master key minimally
  - Space – disk, main memory, etc.
  - Time – only for short durations
  - Use – only to derive session keys
  - Authority – only to authorized programs
- Derive session keys from master key
  - Session keys to be used in communication
  - Master key only to generate / protect session keys
- Change session keys at pre-defined frequency
  - Based on time
  - Based on amount of data

Authentication Principles (III)

- Use different keys for different security functions (authentication, encryption, signatures, etc.)
- Use different keys for different directions of communication
- Use good nonce in protocols to defuse replay attacks
- Add own random values to messages before hashing or signing them
- Anchor initial authentication in nonce that cannot be predicted or derived in the protocols

Authentication Principles (IV)

- Use passwords rarely and carefully
- Protect passwords in the communication protocols
- Add characteristics information to each session (session ID, sequence numbers, etc.)

Nonce Selection

- Nonce – unique identifier that does not repeat
- Nonce = sequence number:
  - Requires counter synchronization
  - Might be predicted from protocols
- Nonce = time stamp:
  - Requires time synchronization
  - Might be predicted from protocols
- Nonce = random number:
  - Cannot be predicted from protocols
  - Requires good random number generators

Symmetric Key Distribution

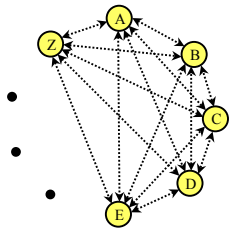
Prof. Shlomo Kipnis
December 24, 2007

Symmetric Key Distribution

- Symmetric Key Distribution between Two Parties:
  1. Party A physically gives party B the key in a secure manner
  2. Parties A and B deliver the key in a secure manner (under the protection of another key)
  3. A trusted third party physically gives parties A and B the key in a secure manner
  4. A trusted third party delivers the key to parties A and B in a secure manner (under the protection of other keys)

Symmetric Key Systems

- N entities – $O(N^2)$ symmetric keys
- $K_{A,B}$ – symmetric key for A and B
- Administration Problems:
  - Adding new entities
  - Removing existing entities
  - Changing keys

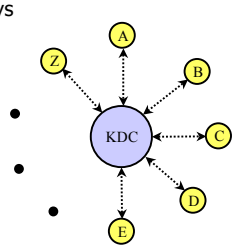


Trusted Third Parties

- Trusted Third Party (TTP) – another party (besides the communicating parties A and B) that is responsible for managing the "trust" in the system
- Types of TTP:
  - Key Distribution Centers (in symmetric systems)
  - Certificates Authorities (in asymmetric systems)

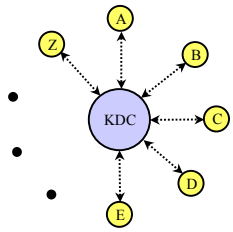
Key Distribution Centers (I)

- Key Distribution Center (KDC)
- N entities – O(N) symmetric keys
- K_A – symmetric master key between A and KDC
- Administration Issues:
  - Easy to add new entities
  - Easy to remove existing entities
  - Easy to change entities keys



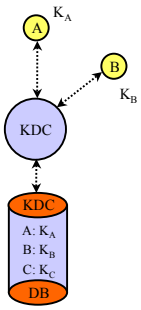
Key Distribution Centers (II)

- Advantages:
  - Centralized administration
  - One point of high security
  - Easy system extendibility
  - Easy function extendibility
- Disadvantages:
  - KDC must be online
  - KDC is point of high security
  - Much communication through KDC
  - KDC may be performance bottleneck



KDC Operation

- A wishes to communicate securely with B
- A requests from KDC a session-key with B
- KDC creates a symmetric session-key K_{AB}
- KDC communicates K_{AB} securely to A (protected with K_A)
- KDC communicates K_{AB} securely to B (protected with K_B)
- A and B can now communicate securely

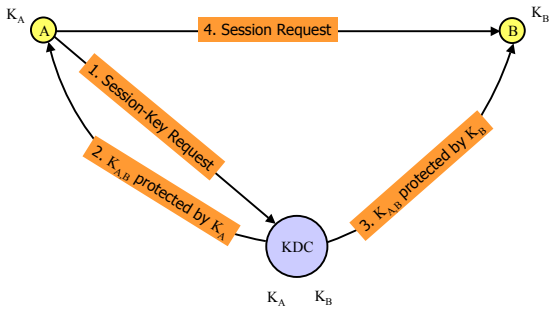


Intranet KDC Scenario (I)

A wishes to communicate securely with B:

1. A contacts KDC and requests a session-key with B
2. KDC sends to A a symmetric session-key K_{AB} (protected with K_A)
3. KDC also sends to B the symmetric session-key K_{AB} (protected with K_B)
4. A contacts B and requests to initiate a secure session

Intranet KDC Scenario (II)



Intranet KDC Scenario (III)

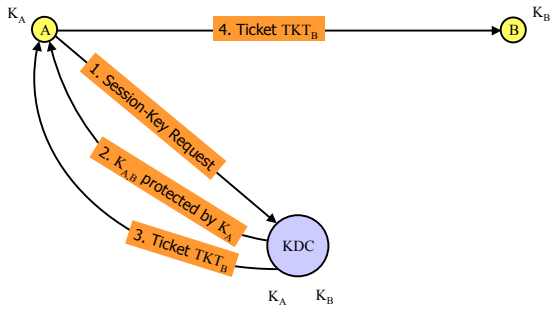
- Scenario Properties:
  - KDC needs to contact B
  - B may be offline or busy
  - KDC is loaded with entity requests
  - A needs to be notified when B is ready
- Scenario is good in tightly connected systems, and when KDC can be loaded

Internet KDC Scenario (I)

A wishes to communicate securely with B:

1. A contacts KDC and requests a session-key with B
2. KDC sends to A a symmetric session-key K_{AB} (protected with K_A)
3. KDC also sends to A the symmetric session-key K_{AB} protected with K_B – this is called ticket TKT_B
4. A provides to B the ticket – TKT_B – in order to initiate a secure session

Internet KDC Scenario (II)



Internet KDC Scenario (III)

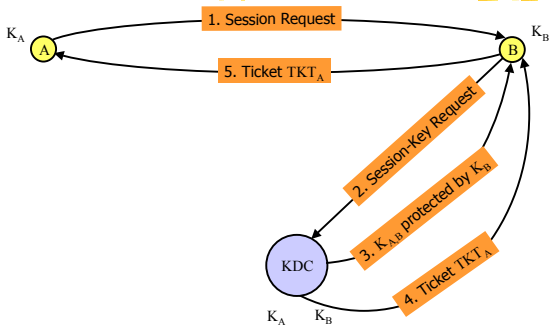
- Scenario Properties:
  - A needs to deliver all data to B
  - KDC does not need to contact B
  - KDC does not need to keep state of requests
  - B is only contacted if and when A wishes to talk
- Scenario is good in loosely connected systems, where contact responsibility is with A, and where KDC cannot be loaded

Extranet KDC Scenario (I)

A wishes to communicate securely with B:

1. A contacts B and requests to hold a secure session
2. B contacts KDC and requests a session-key with A
3. KDC sends to B a symmetric session-key K_{AB} (protected with K_B)
4. KDC also sends to B the symmetric session-key K_{AB} protected with K_A – this is called ticket TKT_A
5. B provides to A the ticket – TKT_A – in order to have A initiate the secure session

Extranet KDC Scenario (II)



Extranet KDC Scenario (III)

- Scenario Properties:
  - B needs to deliver all data to A
  - KDC cannot / does not need to contact A
  - KDC does not need to keep state of requests
  - B is the contact point for the communication
- Scenario is good in extranet-type systems, where contact responsibility is with specific terminals, and where KDC cannot be reached from outside

Sample KDC Protocol (I)

- Sample KDC Protocol with 4 Messages:
  1. $A \rightarrow KDC: \quad A, B, N_1$
     A contacts the KDC with a request to establish a secure session with B. The nonce N_1 is used to disallow replays of the protocol.
  2. $KDC \rightarrow A: \quad E_{K_A}(K_{AB}, A, B, N_1), E_{K_B}(K_{AB}, B, A)$
     KDC sends A the session key K_{AB} and the session identification information A, B, and N_1 encrypted under key K_A. In addition, KDC sends A a ticket for B that contains the session key K_{AB} and the session identification information B and A.

Sample KDC Protocol (II)

- Sample KDC Protocol with 4 Messages (continued):
  3. $A \rightarrow B: \quad A, B, N_2, E_{K_{AB}}(A, B, N_2), E_{K_B}(K_{AB}, B, A)$
     A sends B the session identification information A and B, the nonce N_2, the authenticator $E_{K_{AB}}(A, B, N_2)$, and the ticket $E_{K_B}(K_{AB}, B, A)$ that it obtained from the KDC.
  4. $B \rightarrow A: \quad B, A, E_{K_{AB}}(B, A, N_2+1)$
     B sends A the session identification information B and A, and the authenticator $E_{K_{AB}}(B, A, N_2+1)$.

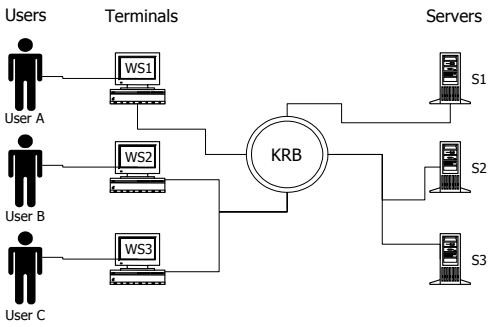
Kerberos

Prof. Shlomo Kipnis
December 26, 2007

Kerberos History

- Developed at MIT in early 1980's
- Computing shift from mainframes to workstations
- Pools of distributed workstations connected to servers
- Concept of "Network Credentials"
- Two commercial and non-compatible versions V4 and V5
- Principles and systems are relevant until today
- Concepts incorporated in DCE, AFS, NFS, NT, etc.

Kerberos Environment (I)



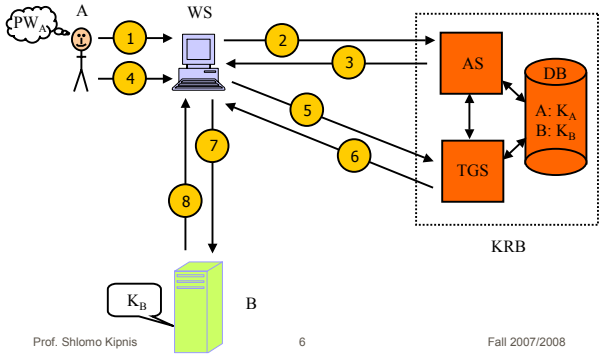
Kerberos Environment (II)

- KRB consists of:
  - AS – Authentication Server
  - TGS – Ticket Granting Server
  - DB – Data Base of entity keys
- Separation between two actions:
  - Authentication – logging into the "network"
  - Communication – holding a session between two parties

Kerberos V4 Architecture (I)

- Kerberos implements the "Internet Scenario"
  - User A has password PW_A to authenticate to KRB
  - KRB stores key K_A that is derived from password PW_A
  - Server B has key K_B to authenticate to KRB
  - KRB stores key K_B that is identical to the server's key
  - Workstations are stateless – they don't know the users and their passwords, and they don't have keys
  - Kerberos provides tickets to the source party (A+WS) that requested the session, and it does not bother the destination party (B)

Kerberos V4 Architecture (II)



Kerberos V4 Protocols (I)

Acquiring Network Credentials:

1. User A sits at workstation WS and enters his name "A".
   At this point the new entity "A+WS" is created.

2. Workstation WS contacts the Authentication Server (AS) on behalf of user A and requests "Network Credentials" to the entity A+WS.

   Workstation WS sends to AS the following data in the clear:
   $< A, WS, TGS, Timestamp_1 >$.

   Workstation WS does not yet know password PW_A of user A, so there is no way to authenticate user A to Kerberos at this point.

Kerberos V4 Protocols (II)

Acquiring Network Credentials (continued):

3. Authentication Server AS checks whether user A is permitted to log in to the network from workstation WS, and if so it replies to A+WS with the following encrypted "Network Credentials":
   $NC_{A+WS} = E_{K_A} (< K_{A,TGS}, TGS, Timestamp_2, Lifetime_2, TKT_{TGS} >)$, where
   $TKT_{TGS} = E_{K_{KRB}} (< K_{A,TGS}, A, WS, TGS, Timestamp_2, Lifetime_2 >)$.

4. User A now enters his password PW_A.

   Workstation WS computes key $K_A = h(PW_A)$ and erases password PW_A from its memory.

   Workstation WS decrypts the encrypted "Network Credentials" (NC_{A+WS}) with the key K_A and erases key K_A from its memory.

Kerberos V4 Protocols (III)

Establishing Connection with Server:

5. When user A wishes to get service from server B, workstation WS contacts the Ticket Granting Server (TGS) and requests a ticket for server B. The request consists of:
   $< B, TKT_{TGS}, Auth_3 >$, where
   $Auth_3 = E_{K_{A,TGS}} (< A, WS, Timestamp_3 >)$.

6. Ticket Granting Server TGS checks whether user A is permitted to get service to server B from workstation WS, and if so it replies to A+WS with the following encrypted "Service Credentials":
   $SC_{A+WS} = E_{K_{A,TGS}} (< K_{A,B}, B, Timestamp_4, Lifetime_4, TKT_B >)$, where
   $TKT_B = E_{K_B} (< K_{A,B}, A, WS, B, Timestamp_4, Lifetime_4 >)$.

Kerberos V4 Protocols (IV)

Establishing Connection with Server (continued):

7. Workstation WS decrypts the encrypted SC_{A+WS} with the key $K_{A,TGS}$.
   Workstation WS contacts the server B to request a secure session. The "Service Request" is:
   $SR_B = < TKT_B, Auth_5 >$, where
   $Auth_5 = E_{K_{A,B}} (< A, WS, Timestamp_5 >)$.

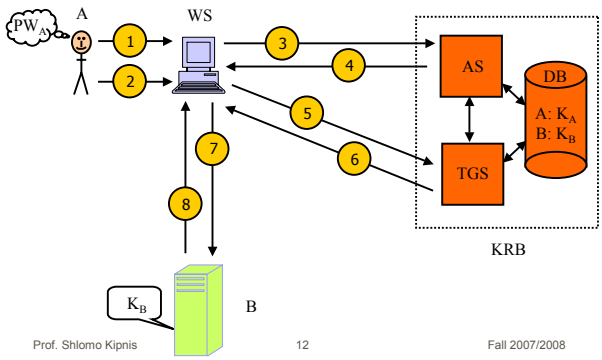
8. Server B decrypts the encrypted ticket TKT_B with the key K_B.

   Server B decides whether it wishes to provide service to A+WS, and if so it replies to A+WS with the following authentication data:
   $< Auth_6 > = E_{K_{A,B}} (< Timestamp_5 + 1 >)$.

Kerberos V5 Architecture(I)

- V5 improves over V4:
  - Standard message byte ordering
  - Multiple network address types
  - Multiple encryption algorithms
  - Arbitrary ticket lifetimes
  - Improved protocols
  - Authentication forwarding, proxying, postdating
  - Inter-realm authentication

Kerberos V5 Architecture (II)



Kerberos V5 Architecture (III)

- Standard message byte ordering:
  - V4 message structure is determined by the sender
  - V5 messages are defined with Abstract Syntax Notation (ASN.1)
- Multiple network address types:
  - V4 uses IP addresses
  - V5 allows other types of addresses
- Multiple encryption algorithms:
  - V4 uses DES for encryption
  - V5 allows specifying different kinds of encryption algorithms
- Arbitrary ticket lifetimes:
  - V4 ticket lifetimes is limited by 1280 minutes
  - V5 ticket lifetimes can be arbitrarily long

Kerberos V5 Architecture (IV)

- Authentication forwarding, proxying, and postdating:
  - V5 allows forwarding credentials from one client to another
  - V5 allows proxying of credentials
  - V5 allows issuing postdated tickets
- Inter-realm authentication:
  - V5 includes "realm" in the authentication protocols
- Improved protocols:
  - V5 introduces a pre-authentication step between client and AS
  - V5 eliminates double encryption in messages
  - V5 changes the encryption mode (from PCBC to CBC)
  - V5 negotiates sub-session keys by the client and the server

Kerberos V5 Protocols (I)

Acquiring Network Credentials:

1. User A sits at workstation WS and enters his name "A".
   At this point the new entity "A+WS" is created.

2. User A enters his password PW_A.
   Workstation WS computes key $K_A = h(PW_A)$ and erases password PW_A from its memory.
   Workstation can optionally compute some pre-authentication data from the user's password.

Kerberos V5 Protocols (II)

Acquiring Network Credentials (continued):

3. Workstation WS contacts the Authentication Server (AS) on behalf of user A and requests "Network Credentials" to the entity A+WS.
   Workstation WS sends to AS the following data in the clear:
   $< A, WS, Realm_A, TGS, Times_1, Nonce_1 >$, where
   $Times_1$ gives the time validity interval, and $Nonce_1$ is random value.
   There is also an option of sending the pre-authentication data.

4. Authentication Server AS checks whether user A is permitted to log in to the network from workstation WS, and if so it replies to A+WS with the following two items:
   (1) $< Realm_A, A, WS, TKT_{TGS} >$
   (2) $E_{K_A} (< K_{A,TGS}, Times_2, Nonce_1, Realm_{TGS}, TGS >)$
   where $TKT_{TGS} = E_{K_{KRB}} (< K_{A,TGS}, Realm_A, A, WS, Times_2 >)$.
   Workstation WS decrypts the encrypted item (2) with key K_A.

Kerberos V5 Protocols (III)

Establishing Connection with Server:

5. When user A wishes to get service from server B, workstation WS contacts the Ticket Granting Server (TGS) and requests a ticket for server B. The request consists of:
   $< B, Times_3, Nonce_3, TKT_{TGS}, Auth_3 >$, where
   $Auth_3 = E_{K_{A,TGS}} (< A, WS, Realm_A, Timestamp_3 >)$.

6. Ticket Granting Server TGS checks whether user A is permitted to get service to server B from workstation WS, and if so it replies to A+WS with the following two items:
   (1) $< Realm_A, A, WS, TKT_B >$
   (2) $E_{K_{A,TGS}} (< K_{A,B}, Times_4, Nonce_3, Realm_B, B >)$
   where $TKT_B = E_{K_B} (< K_{A,B}, Realm_A, A, WS, Times_4 >)$.

Kerberos V5 Protocols (IV)

Establishing Connection with Server (continued):

7. Workstation WS decrypts the encrypted item (2) sent by the TGS with key $K_{A,TGS}$.
   Workstation WS contacts server B and requests to hold a session by sending the following:
   $< TKT_B, Auth_5 >$, where
   $Auth_5 = E_{K_{A,B}} (< A, WS, Realm_A, Timestamp_5, Subkey, Seq\# >)$.
   The fields Subkey and Seq# are optional.

8. Server B decrypts the encrypted ticket TKT_B with the key K_B.
   Server B decides whether it wishes to provide service to A+WS, and if so it replies to A+WS with the following authentication data:
   $< Auth_6 > = E_{K_{A,B}} (< Timestamp_5, Subkey, Seq\# >)$.

Kerberos V5 Flags and Options (I)

- INITIAL:
  - Indicates that a ticket was issued by AS and not by a TGS.
- PRE-AUTHENT:
  - Indicates that the user was pre-authenticated by some means before a TGS ticket was issued.
- HW-AUTHENT:
  - Indicates that the user was authenticated with a hardware token before a TGS ticket was issued.
- RENEWABLE:
  - Tells TGS that this ticket can be used to obtain a replacement ticket that expires at a later date.
- MAY-POSTDATE:
  - Tells TGS that a post-dated ticket may be issued based on this ticket-granting ticket.
- POSTDATED:
  - Indicated that this ticket has been postdated.

Kerberos V5 Flags and Options (II)

- INVALID:
  - Indicates that this ticket is invalid and must be validated by the TGS before use.
- PROXYABLE:
  - Tells TGS that a new service-granting ticket with a different network address may be issued based on this ticket.
- PROXYABLE:
  - Indicates that this ticket is a proxy.
- FORWARDABLE:
  - Tells TGS that a new ticket-granting ticket with different network address may be issued based on this ticket-granting ticket.
- FORWARDED:
  - Indicates that this ticket has either been forwarded or that it was issued based on authentication involving a forwarded ticket.

Kerberos Keys (I)

- Kerberos holds one master Key DB
- Kerberos allows several read-only replicas of the Key DB
- Each replica Key DB should be updated regularly from the master Key DB in a secure manner (encrypted and authenticated)
- Key and password updates are done only on the master copy of the Key DB
- Updates of the replica Key DB may take some time
- Kerberos allows several simultaneous key versions

Kerberos Keys (II)

- Kerberos session key between client and server can be used for a long time
- Client and server may wish to use different versions of the session key to hold several communications in the same session
- Client may suggest usage of a "sub-key" and "seq-num" in a given session, in order to distinguish between different communications in the same session

Kerberos Database (I)

- name – name of the entity
- key – master key of the entity
- kvno – version number of the master key
- max_life – maximal time for tickets
- max_renewable_life – maximal time to renew tickets
- k_kvno – version number of the Kerberos key under which the key in this entry is encrypted
- expiration – time when this entry will expire

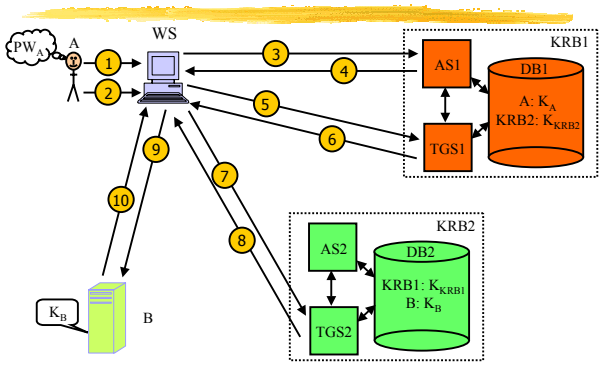
Kerberos Database (II)

- mod_date – date of last modification
- mod_name – name of last modifier
- flags – are tickets forwardable, proxyable, postdateable
- pw_expiration – time when password will expire
- last_pw_change – time of last change of password
- last_success – time of last successful login

Kerberos Realms (I)

- Each realm (domain) manages its own entities
- Inter-realm communication is enabled by having one KRB realm be a client of another KRB realm, and by issuing tickets to communicate between realms
- V4 allows communication only between entities in neighboring realms
- V5 allows communication between entities in realms that can be reached by any chain of KRB realms

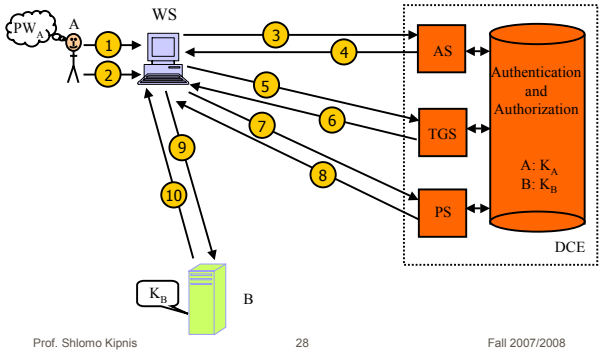
Kerberos Realms (II)



DCE

- DCE – Distributed Computing Environment
- Created by the OSF (Open Software Foundation) in the early 1990's.
- Extended Kerberos in several ways:
  - Addition of Privilege Server (PS) that defines authorizations of different principals (users, computers, other entities)
  - Inclusion of privilege data in tickets
  - Definition of access control lists to support privileges

DCE Architecture

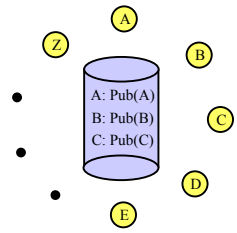


Public-Key Systems

Prof. Shlomo Kipnis
December 31, 2007

Public-Key Management (I)

- N entities – O(N) asymmetric keys:
  - Prv(A) – private key of A
  - Pub(A) – public key of A
- Management Issues:
  - Creating public/private keys
  - Distributing public keys
  - Revoking public keys



Public-Key Management (II)

- Creating Public/Private Keys:
  - Client creates the public/private keys. Client keeps the private key secret and publishes the public key.
  - One of many public servers creates the public/private key-pair. The server gives the client the private key (in a secure manner), and it publishes the public key.
  - A highly secure Trusted-Third-Party creates the public/private key-pair. It gives the client both the private key and the public key (in a secure manner).

Public-Key Management (III)

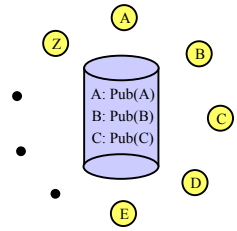
- Distributing Public Keys:
  - Physical delivery to destination entities (and possibly also physical installation at such entities)
  - Attachments to documents (such as E-mail docs, Word docs, etc.)
  - Publication on public directories (such as bulletin boards, LDAP servers, and web sites)
  - An official public-key authority that provides the public key upon request

Public-Key Management (IV)

- Revoking Public Keys:
  - Revocation reasons:
    - Entity left the organization
    - Private key of entity was compromised
  - Distribution of periodic Public-Key Revocation Lists
  - Distribution of periodic Public-Key Activation Lists
  - Clients need to check at public-key directories for the validity of public keys before using them

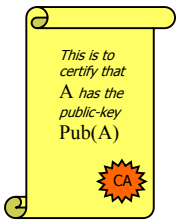
Public-Key Management (V)

- Problems:
  - Authenticating the public-key information (the name of the owner, the public key, the validity of the pubic key, etc.)
  - Man-in-the-middle attacks (an opponent replaces the correct public key with another public key, for which the private key is known to the opponent)
  - Revocation of public keys



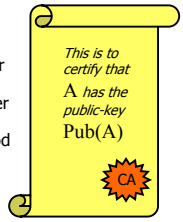
Public-Key Certificates (I)

- Certificate is an official document that stands to authenticate the data contained in it
- Public-Key Certificate is an official document that stands to authenticate the binding of a particular entity with the public-key of that entity:
  - Public-Key Certificate of A = < A, Pub(A) > authenticated



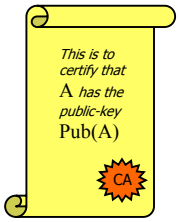
Public-Key Certificates (II)

- Certificate Properties:
  - It contains information that identifies its owner
  - It contains the certified public-key of the owner
  - It contains information about the validity period
  - It may contain some usage policies
  - It may contain additional extension fields
  - It is signed by a known authority (called a Certification Authority)



Public-Key Certificates (III)

- Certificate Properties (continued):
  - The certificate and the information in it are not secret
  - The certificate cannot be forged
  - The certificate can be issued for specific purposes and duration
  - The certificate can be revoked if and when required

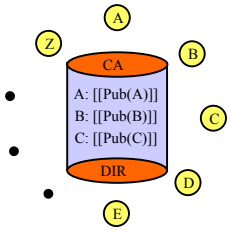


Certification Authorities (I)

- A Certification Authority is a body that authenticates data in certificates
  - Authentication is with Private-Key of the Certification Authority
  - Verification is with Public-Key of the Certification Authority
- Prv(CA) – must be highly confidential
- Pub(CA) – must be known and highly trusted

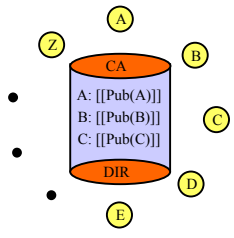
Certification Authorities (II)

- Certificate Authority (CA)
- N entities – O(N) asymmetric keys
- Prv(A) – stored only at A
- Pub(A) – signed by CA
- Administration Issues:
  - Creating public/private keys
  - Creating certificates
  - Distributing certificates
  - Revoking certificates
  - Distributing public key of CA



Certification Authorities (III)

- Advantages:
  - Communication does not involve CA
  - Certificates are not sensitive
  - Parties can have multiple CA's
  - CA can be offline
- Disadvantages:
  - CA needs to be highly secure
  - Problem of certificate revocation



Certification Authorities (IV)

- Certificate Issuance:
  - A wishes to certify its public key – Pub(A)
  - A communicates with some CA in a secure manner
  - The CA verifies the identity of A (by whichever means that are determined by the operational policies of CA)
  - The CA creates a PK-Certificate – containing "A" and "Pub(A)"
  - The CA returns the PK-Certificate to A
  - The CA keeps a record of A, of A's public key, of the time, and of the certificate issued to A
- Note: The CA does not need to know Prv(A)

Certification Authorities (V)

- Certificate Distribution:
  - A may post the PK-certificate on public directories and on web sites
  - A may attach the PK-certificate to documents that it sends to others
  - A may communicate the PK-certificate to parties demanding to identify A
- Note: It is A's responsibility to update or to renew the PK-certificate as necessary (change of data, extension of validity, lost private key, etc.)

Certification Authorities (VI)

- Certificate Revocation:
  - Party A may contact the CA to revoke its certificate
  - CA may initiate the revocation of A's certificate
- CA needs to issue CRL's – Certificate Revocation Lists on a regular basis
- CRL's should be published at "Revocation Servers"
- The Revocation Servers and the CRL's should be approached by clients wishing to check the validity of certain certificates

Certification Authorities (VII)

- Certificate Verification:
  - Party B, wishing to verify Pub(A), needs to check the integrity of the PK-certificate and the CA's signature on it. The public key of the CA is assumed to be known to all and is highly trusted.
  - Party B needs also to check that the PK-certificate has not been revoked (by checking relevant certificate-revocation directories)
- Note: It is B's responsibility to validate the data in the PK-certificate as necessary (name, information, validity time, etc.)

X.509 Certificates

- The X.509 standard is part of the X.500 series of standards for distributed directory services
- The X.509 standard defines a framework for the provision of public-key authentication
- Topics:
  - X.509 Certificate Structure
  - X.509 Certificate Fields
  - X.509 Certificate Extensions

X.509 Certificate Structure

X.509 Version
Serial Number
Signature Parameters
Certificate Issuer
Not Before
Not After
Subject Details
Subject Public Key
Extensions
Signature

X.509 Certificate Fields (I)

- X.509 Version – current version is 3, older versions are supported for compatibility
- Serial Number – of the certificate in the issuer's domain
- Signature Parameters – algorithm and parameters of the issuer's public key
- Certificate Issuer – name and details of issuer
- Not Before – start of certificate validity period
- Not After – end of certificate validity period

X.509 Certificate Fields (II)

- Subject Details – name and details of entity to which the certificate is issued
- Subject Public Key – algorithms, parameters, and values of the public key of the entity to which the certificate is issued
- Extensions – variety of options to include additional information in the authenticated certificate
- Signature – of the issuer on all the fields of the certificate

X.509 Certificate Extensions (I)

- Authority Key Identifier – which CA key was used
- Subject Key Identifier – which entity key is certified
- Key Usage – authentication, encryption, signatures, etc.
- Extended Key Usage – applications and protocols in which the key can be used
- CRL Distribution Point – location of server(s) where relevant Certificate Revocation Lists can be found
- Certificate Policies – additional policy on certificate usage

X.509 Certificate Extensions (II)

- Policy Mapping – between domains
- Subject Alternative Name – non-standard names
- Issuer Alternative Name – non-standard names
- Subject Directory Attributes – certificate owner attributes
- Basic Constraints – basic trust constraints
- Path Length Constraints – for path exploration
- Name Constraints – for CA certificates
- Policy Constraints – for CA certificates

Public-Key Infrastructure

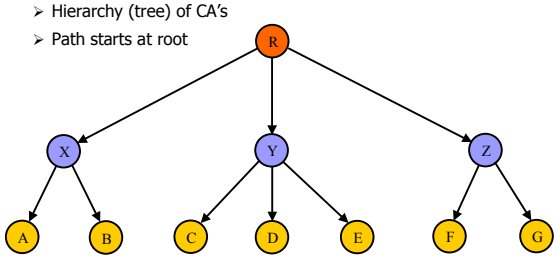
Prof. Shlomo Kipnis
January 9, 2008

CA Trust Models (I)

- Need for CA Trust Models:
  - Zillions of entities need certificates
  - Different non-cooperative CA's in different domains (financial, health, legal, business, military, government, etc.)
  - One CA cannot handle all entities
  - Need to handle multiple CA's and interconnections between them
- Notation: $CA << X >>$ stands for a certificate that was issued by CA for entity X.

CA Trust Models (II)

- Hierarchical Model:
  - Root CA is trusted by all entities
  - Hierarchy (tree) of CA's
  - Path starts at root

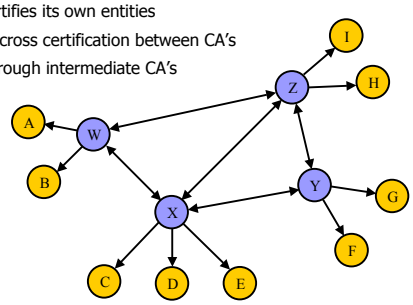


CA Trust Models (III)

- Hierarchical Model:
  - Proposed originally in late 1980's in the context of the PEM (Privacy Enhanced Mail) standard
  - Never really caught due to lack of compliance to one standard and one root CA
  - Used internally in hierarchical organizations (large corporations, financial institutions, military, etc.)
  - Party A, wishing to identify party D, needs to obtain a chain of certificates of the form $R << Y >>$ and $Y << D >>$
  - In the hierarchical model, each node in the tree needs to obtain a certificate from its parent in the tree

CA Trust Models (IV)

- Enterprise Model:
  - Each CA certifies its own entities
  - Network of cross certification between CA's
  - Paths go through intermediate CA's

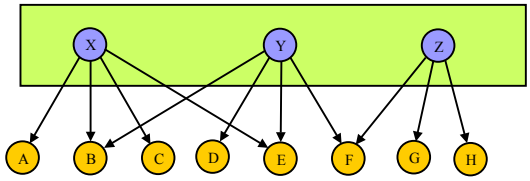


CA Trust Models (V)

- Enterprise Model:
  - Used since the early 1990's as a convenient way to interconnect organizations that maintain business connections
  - In wide use today, since it correlates well to inter-operational characteristics of organizations
  - Party A, wishing to identify party F, needs to obtain a chain of certificates of the form $W << X >>$, $X << Y >>$, and $Y << F >>$
  - In the enterprise model, each "leaf" node needs to obtain a certificate from its parent "enterprise" node
  - In addition, an "enterprise" node needs to issue and to obtain cross-certificates to (some) peer "enterprise" nodes

CA Trust Models (VI)

- Flat Model:
  - List of trusted CA's
  - Paths are immediate (one step)
  - Used in Internet Browsers

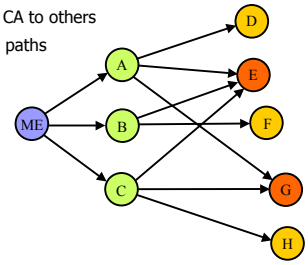


CA Trust Models (VII)

- Flat Model:
  - Used since the early 1990's in Internet Browsers and in other software packages
  - Correlates well with trust models that are used in the regular business world today (credit cards, id cards, driver license, etc.)
  - To identify party E, there is a need to obtain either a certificate of the form X << E >> or a certificate of the form Y << E >>
  - In the flat model, each "leaf" node needs to obtain a certificate from one of many possible CA's (Microsoft, GTE CyberTrust, Verisign, Thawte, Valicert, Certisign, etc.)

CA Trust Models (VIII)

- Personal Model:
  - Web-of-Trust (centered at ME)
  - Each user acts as CA to others
  - Trust threshold in paths
  - Used in PGP



CA Trust Models (IX)

- Personal Model:
  - Used since the early 1990's in the PGP (Pretty Good Privacy) package and in other utilities
  - Very popular with individuals wishing to define their own web-of-trust
  - In the personal model, trust is gained either by immediate (one level) certificate, or by several chains of certificates that "build up" the level of trust (using threshold functions)
  - To identify party E, there is a need to obtain one chain of certificates of the form (ME << A >> and A << E >>) AND/OR another chain of the form (ME << B >> and B << E >>) AND/OR another chain of the form (ME << C >> and C << E >>)

CA Trust Models (X)

- Summary:
  - No universally accepted trust model
  - Many models, policies, and interconnections
  - Existing solutions are incomplete
- Problems:
  - Certificate path exploration
  - Checking certificate policies
  - Certificate revocation
  - Certificate lifetime management

Public-Key Infrastructure

- Infrastructure
- PKI Technologies
- PKI Services
- Certification Services
- Certificate Management
- Certificate Revocation Lists
- Certificate Activation Lists

PK Infrastructure

- Public-Key Infrastructure (PKI):
  - General Architecture
  - Availability
  - Reliability
  - Predictability
  - Comprehensive
  - Transparency
  - Standard API
  - Application Enabling

PKI Technologies

- Symmetric Cryptography – DES, 3-DES, AES:
  - Advantages: Cost, Speed, Simplicity
  - Disadvantages: Scale, Key Distribution
- Hash Functions – MD5, SHA-1, HMAC:
  - Advantages: Cost, Speed, Simplicity, Size Reduction
- Asymmetric Cryptography – DH, El-Gamal, DSS, RSA:
  - Advantages: Scale, Key Distribution, Signatures, Key Exchange
  - Disadvantages: Cost, Speed, Complexity

PKI Services (I)

- Policy Authority – defining policies
- Certification Authority – certifying public keys
- Registration Authority – registering clients
- Certificate Repository – for holding certificates
- Certificate Revocation – for canceling certificates
- Key Backup – to overcome key losses
- Key Recovery – to overcome key losses
- Key Update – to renew keys automatically
- Key Archive – to hold old keys
- Cross Certification – to communicate between domains

PKI Services (II)

- Authentication – of parties
- Integrity – of data
- Confidentiality – of data
- Non Repudiation – of documents
- Secure Communication – between parties
- Time Stamping – providing authenticated time services
- Notarization – providing authenticated notary services
- Data Archive – storing important data securely
- Policy Management – enforcing policies

Certificate Management (I)

- Certificate Initialization:
  - Entity registration
  - Key-pair generation
  - Certificate creation
  - Key distribution
  - Certificate distribution
  - Key backup

Certificate Management (II)

- Certificate Lifetime:
  - Certificate storage
  - Certificate retrieval
  - Certificate validation
  - Path validation
  - Key recovery
  - Key update

Certificate Management (III)

- Certificate Cancellation:
  - Certificate expiration
  - Certificate revocation
  - Key history
  - Key archive
  - Data archive

Certificate Management (IV)

- Certificate Renewal:
  - Key-pair generation
  - Certificate creation
  - Key distribution
  - Certificate distribution
  - Key backup
  - Key history
  - Key archive
  - Old certificate archive

Certificate Revocations Lists (I)

- A Certificate Revocation List (CRL) is a signed data structure that contains a list of revoked certificates
  - Integrity and authenticity of CRL is provided through the digital signature on the CRL
  - Protocols that communicate the CRL do not need to authenticate it
  - To improve performance, CRL's can be cached and contain references to the certificates serial numbers rather than to the full certificates

Certificate Revocations Lists (II)

- Complete CRL – complete list of revoked certificates. List must be issued regularly and be signed by the CA.
- CRL Distribution Points – indication in the certificate where to find information about certificate status.
- Delta CRL – signed differential list of revoked certificates. List must be signed by the CA.
- Authority Revocation Lists (ARL) – list of CA's that are revoked. Must be signed by another CA.

CRL Structure (I)

Version
Signature Parameters
Issuer
Issue Date
Next CRL Update
..... List Of Revoked Certificates Serial Number and Revocation Date .....
Extensions
Signature

CRL Structure (II)

- Version – of CRL. current version is 1.
- Signature Parameters – algorithm and parameters
- Issuer – name and details of issuer
- Issue Date – of this CRL
- Next Issue Date – of next CRL
- List of Revoked Certificates – serial number and date of revocation for each revoked certificate
- Signature – of the CA on the CRL

CRL Schemes (I)

- Complete CRL:
  - All the revocation information of a certain CA is kept in a single CRL
- Problems:
  - Scalability – CRL postings and downloads become very large
  - Timeliness – CRL should have a long validity period, since CRL downloads requires much time

CRL Schemes (II)

- CRL Distribution Points:
  - Standard method to partition the CRL information among multiple distribution points
  - Client can find the distribution point in the "Distribution Point" field stated in the certificate extensions
  - Improves performance and scalability

CRL Schemes (III)

- Delta CRL:
  - Standard method to post timely revocation information without requiring a complete CRL or CRL Distribution Point update
  - Can be used in conjunction with CRL Distribution Points to enhance performance, scalability, and timeliness
  - Improves performance and scalability

CRL Schemes (IV)

- ARL:
  - Similar to a CRL, but instead of holding a list of revoked certificates belonging to clients, it holds a list of the revoked certificates belonging to CA's
  - It is shorter, more security-sensitive, and should be checked more often then CRLs
  - Usually ARL's should be empty

On-Line Revocation Service

- On-Line Revocation Service (OLRS):
  - On-line service that returns status ("revoked" or "valid") of one or more certificates
  - Similar to credit card verification mechanism
  - Also known as OCSP (Online Certificates Status Protocol)
  - Documented in RFC 2560

Certificate Activation Lists (CAL)

- Certificate Activation List (CAL):
  - List of all valid certificates (serial numbers and hashes of certificate for each serial number)
  - Should be signed by the CA
- Problems:
  - CAL is likely to be much larger than CRL
  - CAL might change more frequently

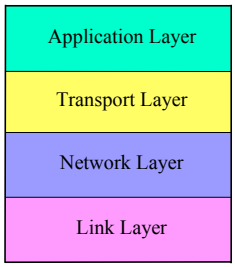
Network Layer Security I

Prof. Shlomo Kipnis
January 14, 2008

Networking Layers (I)

- Communication is handled in several layers:
  - Link Layer –protocols that communicate data between devices on the same physical media
  - Network Layer – protocols that route packets between devices on different physical media
  - Transport Layer – protocols that handle communication sessions between two stations
  - Application Layer – protocols that handle the communication of objects in applications

Networking Layers (II)



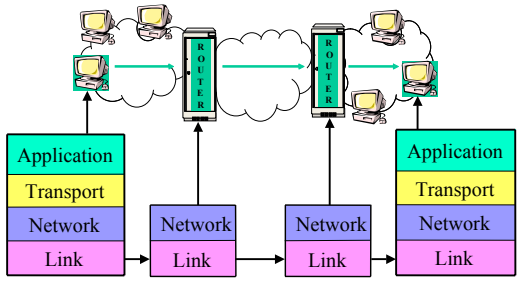
Applications such as browser, email, file transfer, etc.

Transport protocols and API such as TCP, UDP, X.25 etc.

Communication with IP (Internet Protocol)

Communication over Ethernet, Token-Ring, Wireless LAN, etc.

Layers Along Path

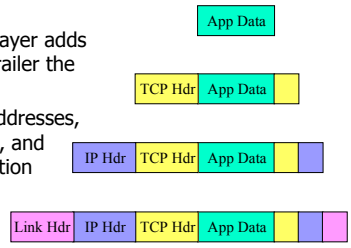


Addressing in Different Layers

- Link Layer Addressing – use of physical network addresses – Media Access Control (MAC) addresses consist of unique 48-bit identification of devices
- Network Layer – use Internet Protocol (IP) addresses that consist of 32 bits (in IPv4) or 128 bits (in IPv6)
- Transport Layer – use port numbers as addresses
- Application Layer – use domain names, host names, port numbers and other application data as addresses

Encapsulation by Layers

- Each layer passes payload and control information such as the destination address to the layer underneath it
- Encapsulation: each layer adds its own header and trailer the to payload
- Header and trailer: addresses, error detection codes, and other control information needed by the layer



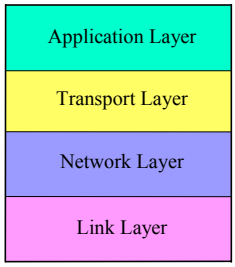
Services at Different Layers

- Datagram – IP, UDP
  - Uni-directional sending of a packet to the destination
  - Packet contains source, destination, error detection code, etc.
  - Detection of packet corruption (but not of loss or reorder)
- Connection – TCP
  - Establishment of a bi-directional session
  - Sequencing of packets
  - Retransmission of lost data
- Client/Server – FTP, HTTP
  - Client sends request, server responds
  - Reliable pairing of response and request

Adding Security at Which Layer?

Which layer is the most appropriate to add security services at?

Different layers provide different services, and it might be necessary to add security services at several layers



Link Layer Security

- Protects all traffic including the IP headers
- Path may contain multiple links (hops)
- Protecting each link is possible, but:
  - Requires cooperation of all routers
  - Communication is still exposed to routers
  - Large overhead – decrypt/encrypt at each hop
  - Provides attacker with encryptions of related plaintexts (same packet, different headers)

Network Layer Security

- IPSec provides following services:
  - Confidentiality, authentication, no-replay, etc.
  - Protection of all applications and data above it
  - Implementation in hardware, operating systems, and/or routers and firewalls
- Cons:
  - Hard to implement
  - Not always available at destination

Transport Layer Security

- SSL – Secure Sockets Layer (by Netscape)
- TLS – Transport Layer Security (by IETF)
- Easy to implement and use
- Widely available
- Cons:
  - Protects only if used
  - Protects only end-to-end communication
  - Headers are exposed
  - Requires changing each application

Application Layer Security

- Secure E-mail (PGP, S/MIME,…)
- XML security
- Implemented by each application
- Cons:
  - Changes each application and operating system
  - Hard to implement, error prone
  - Wasteful of resources
  - No protection for headers and lower-level data

IP Characteristics

- Characteristics of Internet Protocol:
  - Protocol is connectionless
  - Protocol is unreliable
  - Protocol does not guarantee order of packets
  - IP packets can be sniffed
  - IP addresses can be spoofed

IPSec – IP Security Protocol

- Designed by IETF committee in the 1990's
- Designed for IPv4 and IPv6
- Optional in IPv4 / mandatory in IPv6
- Consists of following components:
  - Arch – IPSec Architecture (RFC 2401)
  - AH – Authentication Header (RFC 2402)
  - ESP – Encapsulating Security Payload (RFC 2406)
  - IKE – Internet Key Exchange (RFC 2409)

IPSec Requirements

- Requirements from IPSec:
  - Data source authentication
  - Data integrity protection
  - Data confidentiality
  - Guaranteed order of packets
  - Protection against packet replay
  - Access control mechanisms
  - Limited traffic privacy
  - Full compatibility with IP

IPSec Operation

- IPSec provides unidirectional protection to each IP packet in a stream of IP packets
- IPSec enables authenticating the content of an IP packet using the AH protocol
- IPSec enables encrypting the content of an IP packet using the ESP protocol
- IP packets that undergo protection by IPSec are still standard IP packets at the outmost level
- The algorithms, keys, and other security parameters of IPSec are negotiated ahead of time between the sender and the receiver of the IP stream using the IKE protocol

IPSec SA (Security Association)

- The properties of the secure IPSec channel created between the sender and the receiver are stored in the Security Association (SA) structure
- The SA is unidirectional, and it contains all the data necessary for IPSec processing of outgoing and of incoming packets
- An IPSec device stores all the active SA's it currently has in a data base called the SAD (Security Association Database)
- Most implementations store incoming and outgoing SA's in separate databases

IPSec Modes of Operation

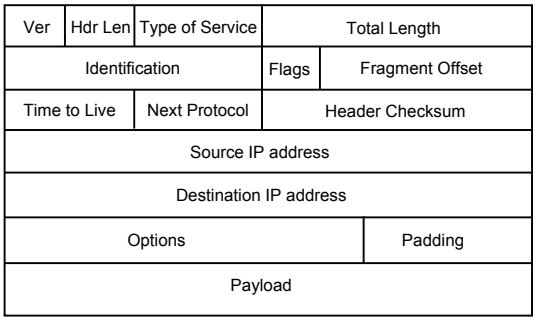
- Transport Mode:

  End-to-end – IPSec encapsulation is done by the source (sender) of the original IP packet, and de-encapsulation is done by the destination (receiver) of the IP packet
- Tunnel Mode:

  GW-to-GW – IPSec encapsulation and de-encapsulation are done by gateways along the route between source (sender) and destination (receiver)

  Special case: when one of the gateways is the sender or the receiver

IPv4 Packet (I)



IPv4 Packet (II)

- Ver – version number (4)
- Hdr Len – length of the IP Header (in units of 32 bits)
- Type of Service – used by routers and gateways for quality of service
- Total Length – total length of header and payload
- Identification – ID number of this IP packet to be used by all fragments of this packet
- Flags – Do-Not-Fragment and Last-Fragment
- Fragment Offset – number of fragment of this IP packet

IPv4 Packet (III)

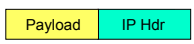
- Time to Live – number of hops this packet can make before it will be discarded
- Next Protocol –number of the protocol encapsulated in this IP packet
- Header Checksum – non-cryptographic checksum of the IP packet header
- Source IP Address – 32-bit IP address of the source
- Destination IP Address – 32-bit IP address of the destination

IPv4 Packet (IV)

- Options – extensions of the IP header with features such as "source routing"
- Padding – to align packet size to a multiple of 32 bits
- Payload – the data contained in this IP packet, which is typically a packet of another protocol (TCP, UDP, IP, ICMP, etc.)

Authentication Header (I)

Original IP packet:



Transport Mode AH packet:

Tunnel Mode AH packet:

Authentication Header (II)

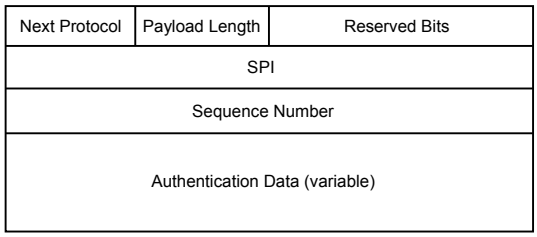
- Transport Mode AH:
  - The outer IP header is (almost) the same IP header of the original IP packet
  - The Authentication header is inserted between the original IP header and the original IP payload
- Tunnel Mode AH:
  - The outer IP header is a new IP header that may be different from the original IP header
  - The Authentication header is inserted between the new IP header and the original IP packet

Authentication Header (III)

- The AH Transformation:
  - When IPSec AH protocol is applied to an IP packet, the result is still an IP packet – the outer IP packet simply contains the AH packet
  - The AH packet contains either the payload of the original IP packet (in the case of Transport Mode) or the original IP packet (in the case of Tunnel Mode)
  - The outer IP packet refers to the AH packet by using the "next protocol" value of 51, and the AH packet refers to the internal packet with the appropriate value of "next protocol"

Authentication Header (IV)

The AH Header:



Authentication Header (V)

- Next Protocol – value that indicates the protocol contained in the AH packet
- Payload Length – length of the AH header in 32-bit words minus 2
- SPI – Security Parameter Index, points to the SA in the receiver's SA table
- Sequence Number – 32-bit counter that is used to prevent packet replays
- Authentication Data – result of the MAC (the length depends on the authentication algorithm)

Authentication Header (VI)

- The AH Authentication:
  - In AH, the MAC is computed over the all fixed fields in the IP packet (including fields in the IP header, fields in the AH header, and the payload)
  - Before the MAC calculation, all the unpredictable or mutable fields in the IP header are set to zero, while all the predictable fields are set to their arrival value
  - Before the MAC calculation, the "Authentication Data" field in the AH header is set to zero

Authentication Header (VII)

AH Encapsulation Process:

- Locate the outgoing SA
  - To be described later
  - If the relevant SA is not found – drop the packet
- Set the SPI in the AH header
- Increment the sequence number in the outgoing SA, and put the new sequence number in the AH header
- Calculate the authentication data

Authentication Header (VIII)

AH Decapsulation Process:

- Locate the incoming SA (according to the SPI)
  - If there is no SA that matches the SPI – drop the packet
- Check if the sequence number is legal
  - If sequence number is not legal – drop the packet
- Calculate the authentication data, and compare to the authentication data in the packet
  - If authentication data do not match – drop the packet
- Continue the processing according to the Next Protocol

Encapsulating Security Payload (I)

Original IP packet:

Transport Mode ESP packet:

Tunnel Mode ESP packet:

Encapsulating Security Payload (II)

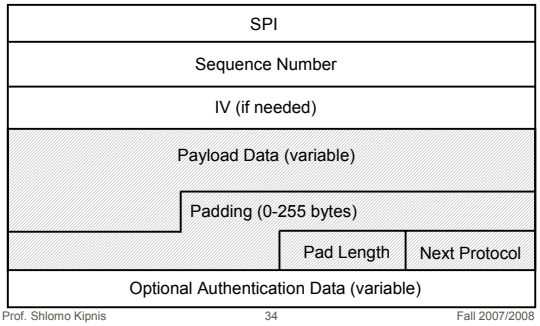
- Transport Mode ESP:
  - The outer IP header is (almost) the same IP header of the original IP packet
  - The ESP header is inserted between the original IP header and the original IP payload, and the ESP trailer is placed after the original IP payload
- Tunnel Mode ESP:
  - The outer IP header is a new IP header that may be different from the original IP header
  - The ESP header is inserted between the new IP header and the original IP packet, and the ESP trailer is placed after the original IP packet

Encapsulating Security Payload (III)

- The ESP Transformation:
  - When IPSec ESP protocol is applied to an IP packet, the result is still an IP packet – the outer IP packet simply contains the ESP packet
  - The ESP packet contains either the payload of the original IP packet (in the case of Transport Mode) or the original IP packet (in the case of Tunnel Mode)
  - The outer IP packet refers to the ESP packet by using the "next protocol" value of 50, and the ESP packet refers to the internal packet with the appropriate value of "next protocol"

Encapsulating Security Payload (IV)

The ESP Header + Payload + Trailer:



Encapsulating Security Payload (V)

- SPI – Security Parameter Index, points to the SA in the receiver's SA table
- Sequence Number – 32-bit counter that is used to prevent packet replays
- IV – Initial Vector used to randomize the encryption
- Payload Data (encrypted) – the data encrypted by the ESP protocol (either only the payload of the original IP packet or the whole original IP packet)
- Padding (encrypted) – to align to IPv4 frame size, to align to the encryption algorithm block size, and to hide the true size of the payload

Encapsulating Security Payload (VI)

- Pad Length (encrypted) – number of bytes of padding (between 0 and 255)
- Next Protocol (encrypted) – value that indicates the protocol contained in the ESP packet
- Optional Authentication Data – result of an optional MAC used in the ESP protocol (the length depends on the authentication algorithm)

Encapsulating Security Payload (VII)

- The ESP Encryption & Authentication:
  - In ESP, the encryption is done over the Payload Data, the Padding, the Pad Length and the Next Protocol
  - In ESP, there is an optional authentication (which does not cover as many fields as the AH authentication)
  - The optional ESP MAC is computed over all the fields in the ESP Header + Payload + ESP Trailer, starting from the SPI field and ending in the Next Protocol field

Encapsulating Security Payload (VIII)

ESP Encapsulation Process:

- Locate the outgoing SA
  - To be described later
  - If the relevant SA is not found – drop the packet
- Set the SPI in the ESP header
- Increment the sequence number in the outgoing SA, and put the new sequence number in the ESP header
- If SA includes encryption, and the encryption algorithm requires an IV – generate an IV and put the IV field in the ESP header

Encapsulating Security Payload (IX)

ESP Encapsulation Process (continued):

- Pad the Payload Data (based on the IP frame size, the encryption algorithm block size, and the required level of payload size hiding)
- Set the fields Pad Length (even if the padding is NULL) and Next Protocol
- If SA includes encryption – encrypt the Payload Data, Padding, Pad Length, and Next Protocol fields
- If SA includes authentication – compute the MAC on all the fields starting from the SPI and ending in the Next Protocol, and append the MAC to the packet

Encapsulating Security Payload (X)

ESP Decapsulation Process:

- Locate the incoming SA (according to the SPI)
  - If there is no SA that matches the SPI – drop the packet
- Check if the sequence number is legal
  - If sequence number is not legal – drop the packet
- If SA includes authentication, calculate the authentication data and compare to the value in the packet
  - If authentication data do not match – drop the packet
- If SA includes encryption, decrypt the data in the packet
- Continue the processing according to the Next Protocol

Network Layer Security II

Prof. Shlomo Kipnis
January 16, 2008

Security Association (SA)

- The properties of the secure IPsec channel created between Alice and Bob are stored in the Security Association (SA) structure
- The SA is unidirectional, and it contains all the data necessary for IPsec processing of incoming and outgoing packets
- An IPsec device stores all the active SAs it currently have in a data base called the SAD (Security Association Data)

IPsec SA

- The SA contains all the data required during IPsec processing of incoming and outgoing packets
  - SPI – Security Parameter Index
    - A 32 bits number that together with the receiver IP address, and the protocol (AH or ESP) uniquely identifies the SA
  - The IPsec sender IP address
  - The IPsec receiver IP address
  - Protocol – AH or ESP
  - Operation mode – Tunnel or Transport
  - Integrity and encryption (if relevant) algorithms and keys
  - Sequence number (incoming – replay window)
  - SA life time
    - For dynamic SAs only
    - Life time is defined either by traffic or by time

IPsec Replay Protection

- The sequence number is set to zero when the SA is established
- The sender increments the sequence number per outgoing packet
- If the receiver deploys replay protection then he extracts the sequence number from the header and checks if he had already encountered this number
  - To avoid dropping legal packets arriving out of order, the receiver maintains in the SA a sliding window (minimal size 32)

IPsec Replay Protection (cont.)

- The sequence number field is included in the MAC computation – an attacker that modifies this value, must also modify the authentication data
- The sequence number must not wrap during a single key lifetime
- AH Keys must be changed after less then 2^{32} packets

IPsec Data Structures

- IPsec is using two data structures (define in the IPsec architecture RFC)
- SAD – Security Association Data contains all the active SAs used by the IPsec implementation
- SPD – Security Policy Data contains user defined policy. The user defines which security services, at which level are offered to each IP datagram

The SPD

The SPD: Security Policy Data contains a list of rules, set by the user. Each rule contains a set of selectors, and an action. The rules are scanned by their order, and a packet is processed according to the first match

- Selectors can be IP addresses, IP addresses ranges, TCP/UDP ports and more
- An action
  - Discard
  - Bypass IPsec
  - Apply IPsec – security services, protocol, and algorithms must be specified for this action. An implementation can also store a pointer to the entry SAD containing a matching active SA (if such exists)

Incoming SAD

- An IPsec device maintains an incoming SAD per protocol, per interface
- The SA is accessed through the SPI that appears in the incoming packet IPsec header

SPI	
1	SA
2	SA
3	SA
…	…

Outgoing SAD

- An IPsec device maintains single outgoing SAD
- The SA is accessed through the SPD

Selectors					
Src IP	Dst IP	Protocol	Src Port	Dst Port	SA
a	b	TCP	*	Telnet	SA
net1	net2	*	*	*	SA
*	*	TCP	SMTP	*	SA

Building the SAD

The IPsec SAD is built either

- Manually – static SAs that use static key can be added manually to the SAD.
  Note replay protection cannot be applied with static SAs
- Automatically – by a key management service

A Secure IPsec Channel

- Alice and Bob communicate securely using IPsec
- Alice outgoing SAD and Bob incoming SAD contain a SA that used for traffic sent from Alice to Bob
  - The SPI is allocated by Bob
- Bob outgoing SAD and Alice incoming SAD contain an SA that is used for traffic sent from Bob to Alice
  - The SPI is allocated by Alice

Example – an IPsec Channel

Protocol = ESP,transport mode, enc=DES3, intg = HMAC-MD5
Life time = 10 Min
Bob's SPI = 17
Bob's encr key = 0x0b0b0b, Bob's intg key = 0x0a0a0a
(used by Bob for decryption and signature verification)

Protocol = ESP,transport mode, enc=AES, intg = HMAC-SHA1
Life time = 10 Min
Alice's SPI = 13
Alice's encr key = 0x0c0c0c, Alice's intg key = 0x0d0d0d
(used by Alice for decryption and signature verification)

Alice Outgoing SAD

Selectors	SA
…	
Alice,Bob	SPI=17,Alice, Bob, DES3, HMAC-MD5
…	

Bob Incoming SAD

SPI	SA
1	
2	
…	
17	SPI=17, Alice,Bob, DES3,HMAC-MD5

Alice Incoming SAD

SPI	SA
1	
2	
…	
13	Bob, Alice, AES, HMAC-SHA1
…	

Bob Outgoing SAD

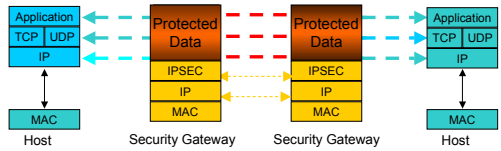
Selectors	SA
Bob, Chris	
Bob, Any, HTTP	
…	
Bob,Alice	SPI=13, AES, HMAC-SHA1

IPsec Transport Mode

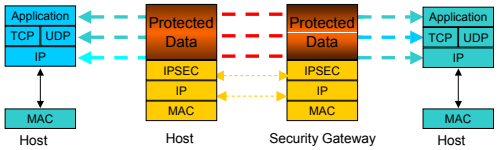
- Supplies end to end security services
- End devices must be modified - TCP/IP stack in the OS must support IPsec

IPsec Tunnel Mode

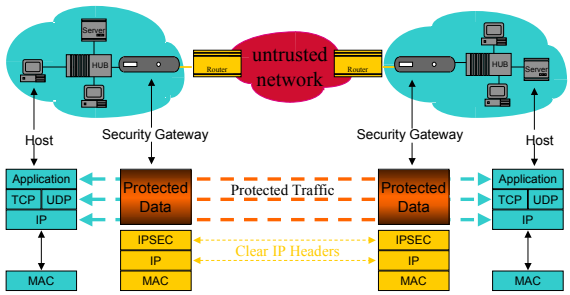
- Can be applied by Security gateways
- End-systems transparent
- End systems addresses are hidden



IPsec Channel – Host and SG



Secure VPN Implementation



Outbound Packet Processing

1. Scan the SPD rules list for selectors match
2. If action is drop, drop packet
3. If action is bypass, forward packet
4. If action is IPsec processing, and no matching SA, drop packet
5. Process packet according to SA
6. Forward packet

Inbound Packet Processing

1. Extract SPI from packet header, and search incoming SAD with SPI
2. If SAD doesn't contain a matching SA, drop packet
3. Otherwise, process packet according to SA
4. If packet is illegal, drop
5. Scan SPD with clear packet: check that packet arrived with the appropriate security services and levels
6. Forward packet to higher layers (transport mode), or to appropriate host (tunnel mode)

VPN – Virtual Private Network

Requirement I: Provide connectivity, reachability, security and QoS identical to private network over a public (or shared) infrastructure

Motivation : Enables users to access services residing in the private LAN from every VPN location in a secure and easy to use fashion

Requirement II: Many organizations use private address spaces (often non-routable). Addresses from the same address space must be assigned to all the VPN users

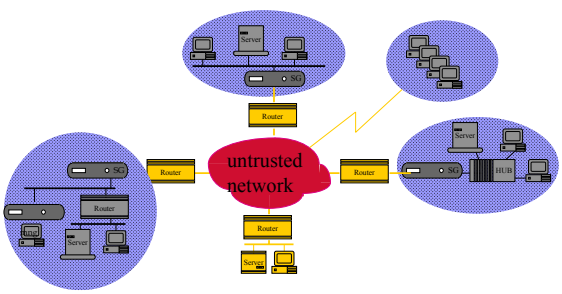
VPN Types

Site to Site VPN: two sites, residing in two different physical locations (two separate LANs), belong to the same VPN

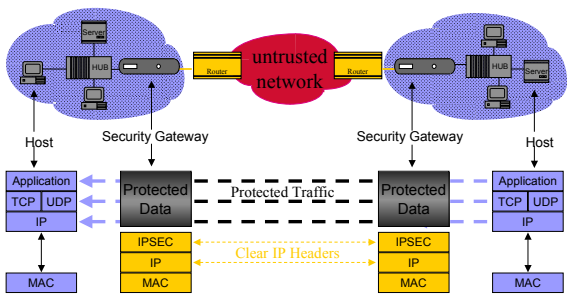
- If the sites belong to the same organization, then the VPN is an Intranet
- If the sites belong to different organizations, then the VPN is an Extranet

Remote Access VPN: roaming users, wish to connect through the Internet to services in their private LAN, and "feel like" they are at the office

Secure VPN



Site-to-Site Secure VPN Implementation



IKE

- IKE is a standard protocol for Security Association establishment and management
- IKE can be executed between
  - Two hosts
  - Two Security gateways
  - A host and a security gateway
- IKE is a merge of two protocols ISAKMP-OAKLEY
- The cryptographic authenticated key exchange is using the Diffie-Hellman protocol
- Work over UDP, port 500
- A generic protocol

IKE Design Requirements

- Secrecy and Authenticity
- Key refreshment (protected against replay)
- PFS
- Scalability
- Privacy and Anonymity (Identity Protection)
- Protection against DoS
- Efficiency (no. of messages and computational)
- Generic (independent of cryptographic algorithms)
- Minimize protocol complexity

IKE Overview

- IKE is a two phase protocol
  - In the first phase the parties establish an ISAKMP SA, that consists of cryptographic algorithms, authentication method and keys
  - The second phase messages are protected by the ISAKMP SA
  - In the second phase the parties establish IPsec SAs that will be stored in the SAD
  - A single IKE phase one, can protect many phase two exchanges

Outbound IPsec Packet Processing

- SA creation is triggered by IPsec, and done on demand
- When the SPD entry doesn't contain a pointer to a valid SA, IPsec triggers IKE, requesting a new SA for this SPD entry
- IKE tries to run a phase two exchange to create an SA
- If an ISAKMP SA required for this exchange doesn't exist, then IKE runs a phase one exchange, and phase two exchange
- When the next packets arrive, they are processed with the newly created SA

Why Two IKE Phases?

- To fulfill the PFS requirement, every phase I exchange, performs a DH exchange
- In Phase II, DH execution is optional – phase II and the IPsec keys can be derived from phase I exchange
- Phase II is more efficient; many two phase II exchanges can use the same set of phase I keys

IKE v2

http://www.ietf.org/internet-drafts/draft-ietf-ipsec-ikev2-04.txt

- Single exchange, authenticated with either public signature keys, or pre-shared secrets
- 4 message exchange
- Provides PFS and identity protection
- Dos protection can be requested by the responder
  - The responder calculates a cookie, and wait for the cookie return by the initiator
- IPsec SA establishment messages can be piggybacked on the initial IKE exchange

Diffie-Hellman Groups

- A group for the DH key exchange specifies the global parameters of DH.
- Each group includes the definition of 2 global parameters and the identity of the algorithm
- Three of these groups are classic DH algorithm using modular exponentiation

Diffie-Hellman Groups

All these three groups (id=1,2,5) have:

- Generator = 2

For group id=1:

- Prime = 2^768 - 2^704 – 1 + 2^64 * { [2^638 pi] + 149686}

For group id=2:

- Prime = 2^1024 - 2^960 – 1 + 2^64 * { [2^894 pi] + 129093}

For group id=5:

- Prime = 2^1536 - 2^1472 – 1 + 2^64 * { [2^1406 pi] + 741804}

All these three groups (id=3,4) have:

- Over Galois fields using elliptic curves.

IKE Message Flow

- IKE message flow always consists of a **request followed by a response**.
- It is the **responsibility of the requester** to ensure reliability.
- If the response is not received within a timeout interval, the requestor will **retransmit** the request or abandon the connection.

SA Terminology

- IKE-SA
  used to secure IKE communication.
- CHILD-SA
  Created by the IKE to be used in ESP or AH security.

The Two IKE Phases

- Phase 1
  Establishment of an **IKE-SA**
  Mutual authentication
  Creation of the first **CHILD-SA**.
  (An **expensive** process which is done rarely)
- Phase 2
  establishing additional CHILD-SA's
  and performing "housekeeping functions"

Phase 1

- Establishes an IKE-SA that includes shared secret information that can be used to efficiently establish Child-SA's.
- Performs mutual authentication
- Creates the first Child-SA
- Consists of **two stages**

Stage 1 - INIT

- Negotiates cryptographic algorithms
- Exchanges nonces
- Performs a Diffie-Hellman exchange

Note: messages are **not protected**

INIT Request

The initiator sends the following message:
→ HDR, SAi1, KEi, Ni

- HDR – header contains SPI, version, flags
- SAi1 – states the cryptographic algorithms the initiator supports
- KEi – initiator's DH value.
- Ni – initiator's nonce.

INIT Response

The responder sends the following message:
← HDR, SAr1, KEr, Nr [CERTREQ]

- SAr1 – states the cryptographic algorithms the responder selected.
- KEr – responder's Difie-Hellman value.
- Nr – responder's nonce.
- CERTREQ – optional certificate request

SKEYSEED Generation

- Each party generates the SKEYSEED which is used to derive the keys used in IKE-SA.

Notice: separate keys are computed for each direction

Authentication Stage

- Authenticates the previous messages
- Exchanges and proves identities
- Establishes the first CHILD-SA

- Messages are protected using the keys derived at the INIT stage.
- Identity protection is achieved

Authentication Request

→ HDR, SK{ IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi, TSr }

- IDi – initiator's identity
- CERT, CERTREQ – optionally presenting or requesting certificates
- IDr – specifiying the responder's desired identity

* SK{…} indicates encription and integrity protection using SK_e and SK_a

Authentication Request

→HDR, SK{ IDi, [CERT,] [CERTREQ,] [IDr,] AUTH, SAi2, TSi, TSr }

- AUTH – authenticates the previous message and the initiator's identity.
  (a digital signature over the first message)
- SAi2, TSi, TSr – needed for first CHILD-SA

Authentication Response

← HDR, SK{ IDr, [CERT,] AUTH, SAr2, TSi, TSr}

- IDr – responder's identity
- CERT – presenting certificates
- AUTH - authenticates the previous message and the resonder's identity.
- SAr2, TSi, TSr - needed for first CHILD-SA

End of Phase 1

- Signatures are verified
- First CHILD-SA is established

Possible DOS Attack

- The response to the first Phase 1 message involves expensive computations
- This can be exploited in a DOS attack where the attacker use IP Spoofing to flood the victim with requests to open connection.
- Can be solved using cookies

Cookies

- A Cookie is a field which is sent to the connection initiator.
- The initiator is expected to return the cookie.
- A returned cookie is a proof that the initiator can receive packets destined to the IP address he declared.

Use of Cookies

- When a responder detects many half-open connections, a **new policy** is used.
- Each new connection request that does not contain a valid cookie is rejected.
- The rejection message contains a cookie and a demand for this cookie in a future connection request.

Cookie Demands

- No one but the creator can construct a valid cookie
- Cheap to create
- No resources needs to be allocated to handle each cookie

Cookie Creation

- Cookie = HASH(<secret> | IPi | SPI | TS)

- <secret> - a secret known only to the responder
- IPi – IP address of the initiator
- SPI – included in the request
- TS – time stamp

(A possible problem …)

Phase 2

Creating a new CHILD-SA:

- Relatively cheap operation
- Can be initiated by either endpoints
- Provides optional perfect-forward-secrecy

All other information flow:

- Will be discussed later

CHILD-SA Request

A CREATE_CHILD_SA request:

→ HDR, SK{ SA, Ni, [KEi,] [TSi, TSr] }

- SA – The initiator's offers for the new SA
- Ni - Initiator's nonce
- KEi – a new DH value
- TSi, TSr – proposed traffic selectors

CHILD-SA Response

A CREATE_CHILD_SA response:

←HDR, SK{ SA, Nr, [KEr,] [TSi, TSr] }

- SA – The responder chosen SA offer
- Nr - Responder's nonce
- KEr – A new DH value (if the request had a Kei)
- TSi, TSr – selected traffic selectors

PFS in IKE

Reminder:

Perfect forward secrecy means that once a connection is closed and its keys are forgotten, even someone who recorded all the data from the connection, and gets access to the long term keys used to protect it, cannot reconstruct the keys of a CHILD-SA that was created during the connection.

Achieving CHILD-SA PFS

- The optional KE field in the establishment of a new CHILD-SA indicates that the keys created depend on a unique DH key exchange. Such keys achieve the PFS quality.
- Expensive procedure.
- Depends on two conditions …

Generating Keying Material

- In the establishment of the IKE-SA a **pseudo-random function** (prf) is negotiated.
- The keying material for all SA's is derived from the output string of the prf.

PRF Usage

- Usually the amount of keying material needed is greater then the prf output.
- The prf is used iteretively:

  Prf+ (K , S) = T1 | T2 | T3 | …

  where:

  T1 = prf (K , S | 0x01)

  T2 = prf (K , T1 | S | 0x02)

  T3 = prf (K , T2 | S | 0x03)

IKE-SA Keying Material

- After INIT (in Phase 1),
  SKEYSEED is calculated:
  SKEYSEED = prf (Ni | Nr , g^ir)
- Ni, Nr – nonces of both parties
- G^ir – the DH key

IKE-SA Keys Derived

Five keys are derived.

- SK_d – used in deriving CHILD-SA keys
- SK_ai, SK_ar – authentication keys
- SK_ei, SK_er – encryption keys

Notice: different keys for each direction

IKE-SA Keys Deriviation

{ SK_d , SK_ai , SK_ar , SK_ei , SK_er } =
Prf+ (SKEYSEED ,
g^ir | Ni | Nr | CKY-I | CKY-R)

- CKY-I , CKY-R – the cookie of both parties

CHILD-SA Keying Material

Creation without PFS:
KEYMAT = prf+ (SK_d, Ni | Nr)

Creation with PFS:
KEYMAT = prf+ (sk_d, **g^ir** | Ni | Nr)

g^ir is the DH key established for this CHILD_SA

CHILD-SA Keying Derivation

- All keys needed for the SA created are derived from the KEYMAT
  (depends on the needs of the algorithms using this SA)
- Multiple SA's can be created at a single CHILD_SA negotiation.

Transport Layer Security I

Prof. Shlomo Kipnis
January 21, 2008

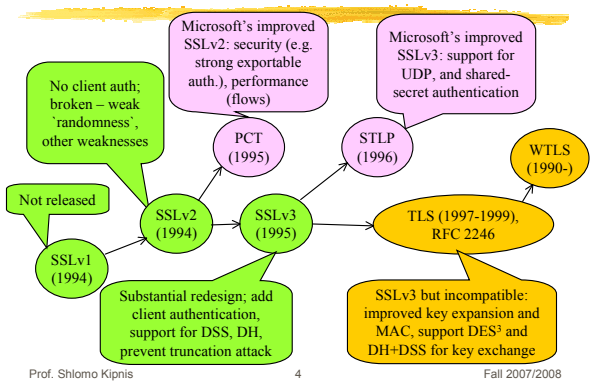
SSL History

- SSL was developed by Netscape in early 1990's
- Netscape identified need for "session layer" security for emerging Internet / Web applications
- Other competing proposals (PCT, HTTPS, etc) came at about the same time
- SSL 3.0 became de-facto standard
- Transport Layer Security (TLS) 1.0 standard is almost identical to SSL 3.0

SSL / TLS in a Nutshell

- SSL provides a `secure TCP tunnel from client to server`:
  - Confidentiality
  - Authentication of server, optionally also of client
  - Message and connection integrity
- SSL: Secure Socket Layer
  - Since SSL (& TLS) operate on top of `standard` Sockets API
- TLS: Transport Layer Security
  - Since TLS (& SSL) secure TCP (the transport layer)
  - IETF standard version of SSL
  - When we describe common aspects we usually say just SSL
- Many implementations, libraries, e.g. Open-SSL
- Original goal and still main use: secure transfer of credit card number…

SSL/TLS Evolution



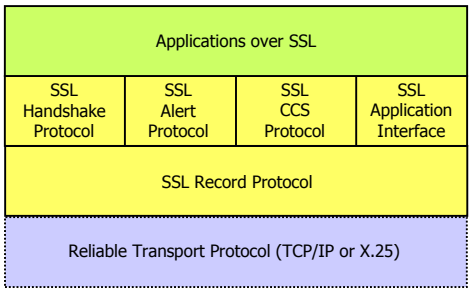
SSL Functionality

- Session-Layer Security:
  - Protection of (bi-directional) transport protocol
- Security Services:
  - Integrity, Authenticity, Confidentiality
- Client Security:
  - Server must be authenticated (using public-key certificates)
- Server Security:
  - Client may be authenticated (using public-key certificates)
- Security Suite:
  - Client and Server negotiate Algorithms and methods

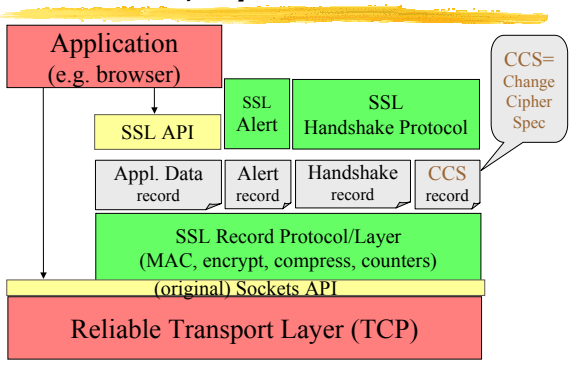
SSL Services

- Server Authentication (mandatory)
- Client Authentication (optional - if required by server)
- Secure connection:
  - Confidentiality (Encryption) – optional, possibly weak (export)
  - Message Authentication
  - Reliability: prevent re-ordering, truncating etc.
- Efficiency: allow resumption of SSL session in new connection (no need to re-do handshake)

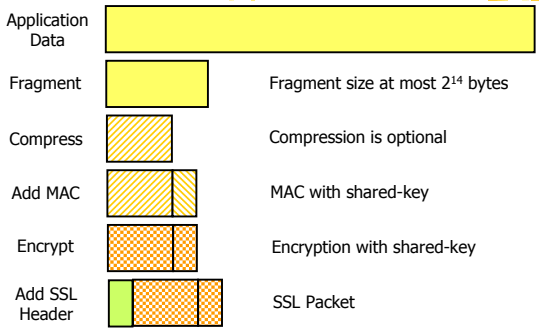
SSL Architecture



SSL Protocols, Layers and Records

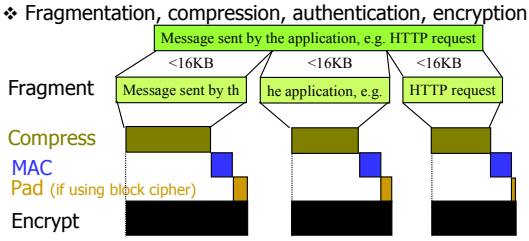


SSL Record Protocol



SSL Record Layer

- Assumes underlying reliable communication (TCP)
- Fragmentation, compression, authentication, encryption



Send each fragment via TCP

SSL Record Protocol

1. Fragments data – 16KB in a fragment
2. Compress each fragment; Compression must be lossless and never increase length (up to 1KB Ok)
3. Authenticate by appending MAC
   - Key: MAC_write_secret (from *master_secret)*
   - MAC computed over *counter* || *length* || *content*
   - Use *counter* (64 bits) to prevent replay in SSL session
   - The *counter* value is only input to MAC, not sent
     - Since we assume SSL is over TCP which ensures FIFO
     - So why SSL adds counter to MAC at all?
4. Padding to complete block (if using block cipher)
5. Encrypt fragment (including MAC)

SSL Alert & CCS Protocols

- SSL Alert Protocol:
  - Used to deliver error messages from side to side
  - Contains five fatal errors and seven non-fatal errors
- SSL Change Cipher Spec (CCS) Protocol:
  - Used to switch to agreed cipher-spec
  - Used in SSL Handshake Protocol
  - Can be used between connections

Alert Protocol and Record

- Signal state changes and indicate errors
- Invoked by:
  - Application - to close connection *(close_notify)*
    - Connection should close with *close_notify*
    - This allows detection of *truncation attack* (dropping of last messages)
    - Notice: *close_notify* is normal, not failure alert!
  - Handshake protocol – in case of problem
  - Record protocol – e.g. if MAC is not valid
    - Notice: easy to tear-down (denial of service)
- Alert record carries alerts

TLS

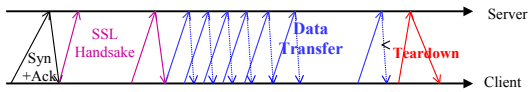
- TLS 1.0 is very similar to SSL 3.0 with some changes:
  - SSL uses an older version of HMAC while TLS uses the new version of HMAC
  - Use of Pseudo Random Function (PRF) in TLS
  - Additional Alert Codes in TLS
  - Some differences in the cipher suites
  - Some differences in client certificate types
  - Minor changes in some cryptographic computations
  - Variable padding length (before encryption) in TLS

Transport Layer Security II

Prof. Shlomo Kipnis
January 23, 2008

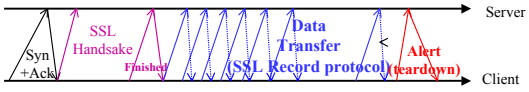
SSL Operation Phases (high level)

- TCP Connection
- Handshake
  - Negotiate (agree on) algorithms, methods
  - Authenticate server and optionally client
  - Establish keys
- Data transfer
- SSL Secure Teardown (why is this necessary?)



SSL Operation Phases

- Client uses SSL API to open connection
- SSL Handshake protocol:
  - For efficiency – resume `session` if possible
  - If not (session not kept, new connection, override)
    - Establish session - algorithms and master keys
  - Establish connection (keys, etc.)
- Data transfer (SSL Record protocol)
- Teardown – use Alert protocol:
  - By application closing connection
  - Or due to error (by handshake or record protocols)



SSL Sessions and Connections

- Connection:
  - TCP/IP connection – send/receive secure messages
  - Reliable: ensures Delivery, Matching, FIFO
  - Independent, different keys for each connection
- SSL Session:
  - May span multiple connections for efficiency
  - Agree on algorithms and options
    - Client specifies possibilities, server chooses or rejects
  - Use public keys to Establish shared *MasterSecret* key
  - Server sets `session_id` so connection can resume (use existing session, for efficiency)
    - Client, server may discard session
    - Recommended (in RFC): keep session at most 24 hours

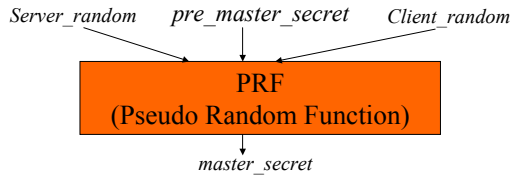
SSL Session State Variables

- Session ID: 32 bytes selected by server
- Peer certificate (X.509 v3)
- Compression method
- Cipher spec (encryption, MAC, etc.)
- Is Resumable: flag: allow new connections
- *master_secret*: 48 bytes, known to both
  - Derived from 48 bytes *pre_master_secret* (from DH key exchange / sent encrypted by RSA)
  - Using random numbers chosen by server and client at 1st connection of session
  - Using Pseudo-Random Function (PRF)
  - How?

Deriving *master_secret* Key

$$master\_secret = PRF_{pre\_master\_secret}(\text{“master secret”} \,||\, Client\_random \,||\, Server\_random)$$

PRF is based on MD5 and SHA-1;
design differs btw SSL & TLS, see later



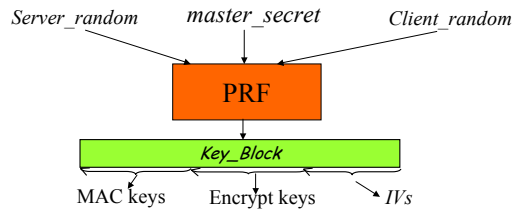
SSL Connection State Variables

- Session ID: 32 bytes selected by server
- Server and client sequence numbers
- *Server_random, client_random:* 32 bytes
  - Unique to each connection!
- Cryptographic keys and Initialization Vectors (IV)
  - Unique to each connection (why?)
  - Distinct encryption and authentication (MAC) keys (why?)
  - Distinct keys for client to server and server to client packets (why?)
  - How?

Deriving Connection Keys, IVs

Key_Block = PRF $_{master\_secret}$ *(master_secret || "key expansion" || Server_random ||Client_random)*

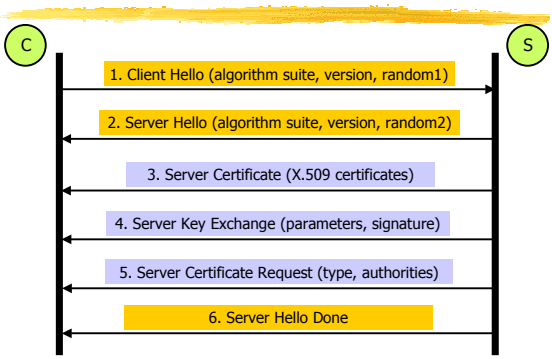
Split *Key_Block* to *ClientMACKey, serverMACKey, ClientEncryptKey,…*(using fixed order)



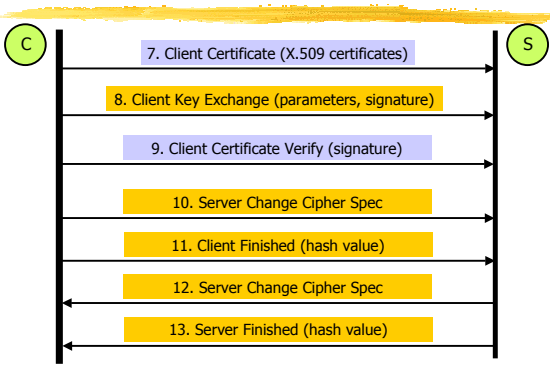
SSL Handshake Protocol

- Agree on *cipher suite*: algorithms and options:
  - Symmetric and Asymmetric Encryption
  - Signature and MAC
  - Compression
  - Options: client authentication, export (weak) versions,…
- Exchange random values
- Check for session resumption.
- Send certificate(s)
- Establish shared keys.
- Authenticate server
- Optionally authenticate client
- Confirm synchronization with peer

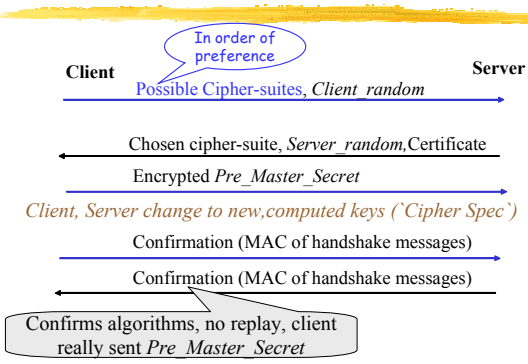
SSL Handshake Protocol (I)



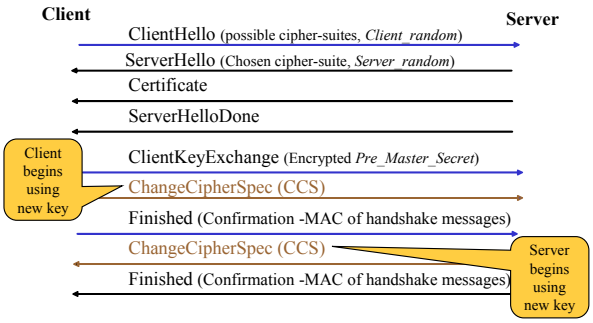
SSL Handshake Protocol (II)



SSL Handshake – Overview



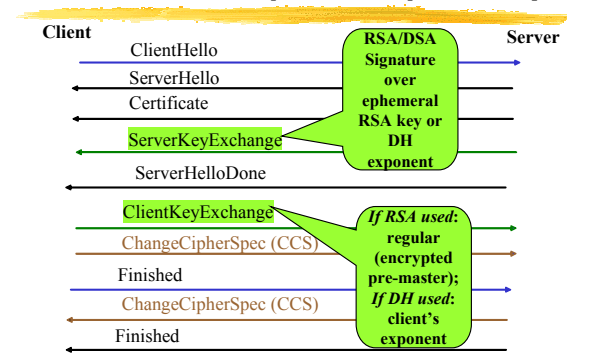
SSL Typical Handshake Messages



Advanced Handshake Features

- Session resumption
- Client authentication
- Ephemeral public keys
  - For forward security – (usually?) using Diffie-Hellman
  - Support for DH, with DSS signatures, is mandatory in TLS
  - Or, for using weak encryption public keys for export reasons (signed by strong public key) – Often with RSA
  - RSA key generation is expensive – often same ephemeral (and short, 512 bits) key used for multiple clients/sessions

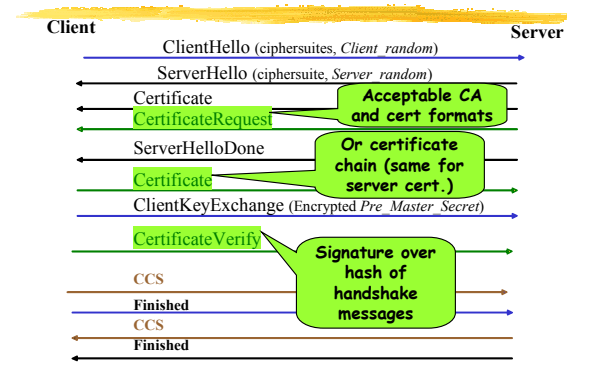
Handshake with Ephemeral public keys



SSL Client Authentication

- Usually, only the server has a certificate
  - Client can authenticate the server
  - Client sends some identification info (e.g. username, password) to server using the SSL tunnel – after it is established
- SSL also supports authentication with client certificates
  - Server requires certificate from client
  - Server signals acceptable Certificate Authorities (CAs) and certificate formats, options etc.
  - Client returns appropriate certificate (chain)
  - Client authenticates by signing using certified public key
- Client authentication using certificates is used mostly within organizations, communities – more on this later

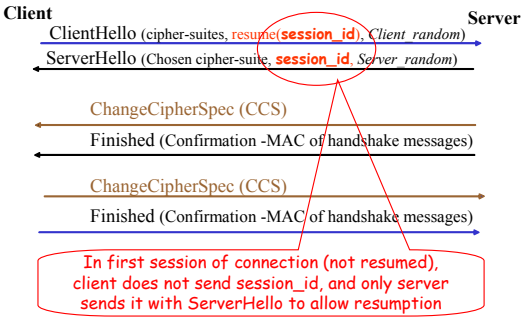
Client Authentication Handshake



SSL Session Resumption

- SSL session setup has substantial overhead
  - Randomness generation (both)
  - Transmission of certificates (both)
  - RSA encryption of Pre-Master-sercret (client)
  - RSA decryption of Pre-Master-secret (server)
  - Derivation of master secret and key block (both)
- Problems:
  - Significant performance penalty (mainly on server)
  - Server vulnerable to clogging (DOS) attacks
- Session resumption:
  - If client makes many connections to same server…
  - Server, client can re-use Pre-Master-secret from last connection
  - How? By identifying a session using *session ID*

Session Resumption Handshake



Session Resumption Issues

- Caching requires considerable server resources
  - Result: cache usually kept for only few minutest, not 24 hrs
- Resumption conflicts with replicated (cluster) servers
  - TCP connections routed to arbitrary server in cluster
  - Solution 1: server in cluster determined by client IP address → but requests from many clients may use same NAT IP addr
  - Solution 2: shared storage of session information → not easy!
  - Solution 3: SSL-session aware connection routing
  - Solution 4: Client side session caching – encrypted, authenticated cache; a non-standard SSL/TLS extension
- Session resumption helps only for repeating connections
  - SSL payments involve one (or few) connections → not much help
- Other possible optimizations (not standardized)
  - Client caching of certificates and other server info (`fast track`)
  - Encrypt using ephemeral, short server keys
  - Server encrypts Pre-Master-Secret using Client's public key

Secure Usage of SSL

- Designing Secure Applications using SSL API
- Validating Certificate (or certificates chain)
- Server Access Control (client authentication)
  - Using client certificates
  - Using username and password, etc.
- Client Access Control (server authentication)
- Site spoofing attacks on browsers

Designing Applications using SSL API

- Several SSL toolkits (e.g. OpenSSL); slightly different APIs
- Initialization tasks:
  - Load CA's certificates (at clients; servers: only if using client auth)
  - Load keys and certificates
  - Seed random number generator (use collected noise)
  - Load allowed cipher suites
    - Most toolkits allow adding new (more secure?) cipher suites
  - In server: generate/load ephemeral DH and/or RSA keys (if used)
- Connection API calls
  - Very similar to standard TCP (Sockets) API
  - But returns server (and optionally client) certificate
  - Need to validate certificate
  - Close (tear-down) connection - to identify truncation attacks

Validating Certificates

- Validation done by application, not SSL!!
- Verify root CA is trusted
  - Predefined list of `trusted CAs` in application
    - E.g. look in your browser…
  - Do we really trust all of them?
- Validate certificate (chain)
  - Validate signature(s)
  - Check validity/expiration dates
  - Check identities, constraints, key usage…
  - Check for revocations – SSL does not carry CRLs; application must collect by itself if CRL's are used.
- Reminder…