T-110.5211 Cryptosystems

Symmetric and unkeyed primitives
Block cipher modes of operation

25.9.2008

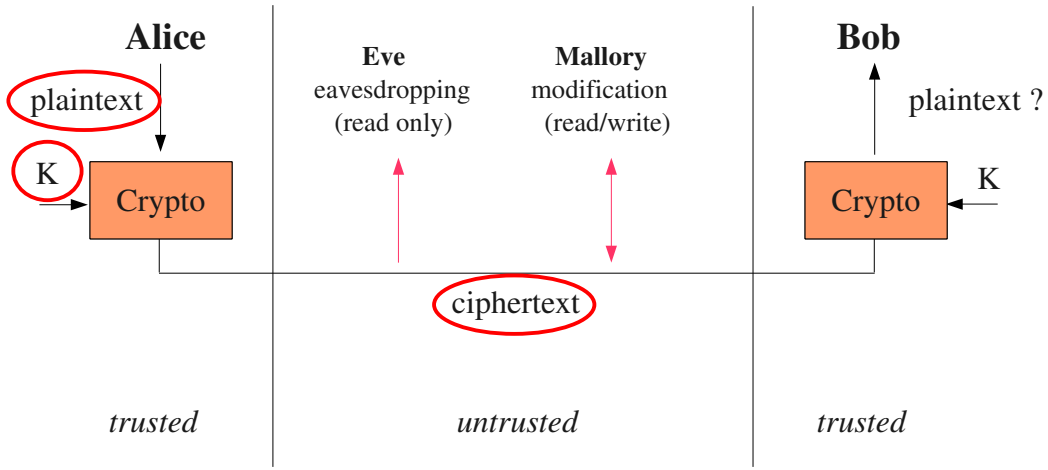
Kaufman et al: Chapters 2.6 / 3.1-3.2 / 4.1-4.2 / 5.1-5.2
Stallings: Chapters 3.1-3.7 / 11.3-11.5

Outline

- Basic symmetric (and unkeyed) primitives
  - Block and stream ciphers
  - Examples of trivial ciphers
  - Structure of modern block ciphers
  - Hash functions
- Block cipher modes of operation
- Assignments
  - Vigenere [crypto.implementations.Vigenere]
  - Generalized CBC [crypto.implementations.GenericCbc]

Basic symmetric (and unkeyed) primitives

Classic crypto (threat) model



Basic goals for “classic crypto”

- Confidentiality
  - An attacker cannot read any messages in flight
- Integrity
  - An attacker cannot modify any messages in flight
- Authentication
  - A receiver has reliable information about who actually sent the message
- (Availability)
  - An attacker cannot prevent a message from being sent and received
  - Typically not a cryptographic problem, and very difficult in practice too

Overview of symmetric primitives

- Symmetric and unkeyed primitives
  - Symmetric – secret key known only by legitimate parties
    - All parties have the same key
  - Unkeyed – algorithm does not use a key
- Symmetric encryption primitives
  - Block ciphers
  - Stream ciphers
- Hash functions
  - Unkeyed
- Message authentication codes (MACs)
  - Symmetric key

Security of symmetric crypto

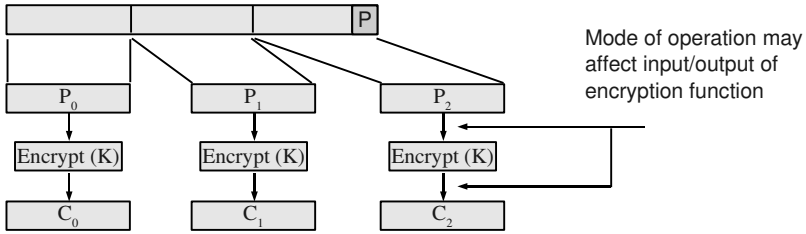
- General threats against symmetric encryption
  - Retrieve message from ciphertext without key
  - Retrieve key => break all past and future messages (for that key)
- Measuring security
  - One-time pad infeasible, security = computational difficulty
- => Time and space required to attack
  - Upper bound – **brute force** of key space $\mathbf{2^n}$
  - Some hash function collision attacks = ~$\mathbf{2^{n/2}}$ (square root)
- Effective key length
  - Any keyed algorithm with an **n**-bit key can be broken in $\mathbf{2^n}$ steps (brute force)
  - If attack against an algorithm with **n** bit key is feasible with less than $\mathbf{2^n}$ steps (say, $\mathbf{2^t}$), algorithm has been theoretically broken
  - In this case, effective key length is **t**
  - For a good algorithm, key length = effective key length
    - Brute force is the best attack against a good algorithm

Security of symmetric crypto

- Attack models
  - Ciphertext only attack
  - Known plaintext attack
  - Chosen plaintext attack
  - Chosen ciphertext attack
  - + other (adaptive chosen plaintext attack, related key attack, etc)
- Modern ciphers should be secure against all of these
  - Example: IPsec => easily guessable packet headers; attacker can often even feed data to be encrypted
- We won't cover cryptanalysis in depth
  - Cryptanalysis of the trivial ciphers is quite straightforward
  - Cryptanalysis of real-world algorithms is too complex for this course

Block cipher

- Block cipher encryption
  - Plaintext is divided into fixed size blocks, typically 64 or 128 bits
  - Padding is added (to last block) to make input full blocks
  - Each plaintext block is encrypted using **the same key**
  - A *mode of operation* may be employed, altering the encryption process; nevertheless, the same key is always used
- Decryption is similar
- Analogy – substitution cipher (classical ciphers)
  - Block size is large => very large number of permutations; key determines perm.



Block cipher

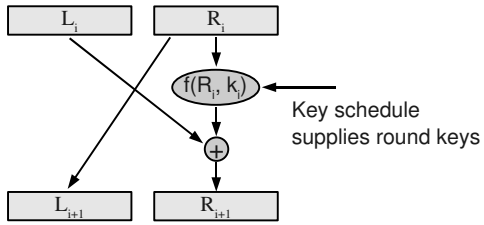
- More detailed definition
  - Message: $\mathbf{m=b_1b_2b_3...}$, where each $\mathbf{b_i}$ consists of **n** bits
  - Encryption function: $\mathbf{c = E_k(b)}$, encryption of block **b** with key **k**
  - Decryption function: $\mathbf{b = D_k(c) = E_k^{-1}(c)}$
  - Each block of the message is encrypted using the same key, **k**
  - => $\mathbf{E_k(m) = E_k(b_1)\ E_k(b_2)\ E_k(b_3)\ ...}$
- Block size (**n**) is usually 64 or 128 bits, key size varies
  - Block and key size do not have to match
- Examples (key size / block size)
  - DES (56 / 64), 3DES (168 / 64), AES (128,192,256 / 128), ...
- Mode of operation
  - None = Electronic Codebook mode (ECB), not recommended
  - We look at other modes of operation later this lecture

Structure of modern block ciphers

- Basic tools
  - **Substitution** provides "confusion"
  - **Permutation** provides "diffusion"
  - Product ciphers = iteration of **rounds** (primitive operations)
- Some design goals
  - Cipher is designed to be secure against *known attacks*
    - But new attacks may be possible
  - Important goal is efficiency, usually both hardware and software implementations
  - Estimated effective key length is long enough (with margin)
    - Estimate depends on best known attacks + guesswork
- Two common approaches
  - Feistel networks (e.g. DES)
  - Substitution-Permutation (SP) networks (e.g. AES)

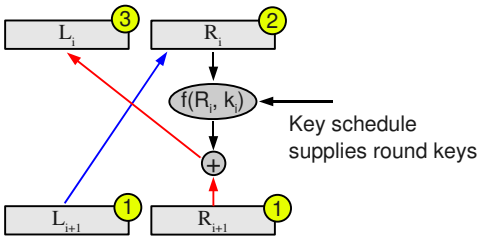
Feistel networks

- Feistel networks
  - Divide block into left and right components, $\mathbf{L_i}$ and $\mathbf{R_i}$
  - Apply round function:
    - $\mathbf{L_{i+1}=R_i}$
    - $\mathbf{R_{i+1}=L_i\ XOR\ f(R_i, k_i)}$ ($\mathbf{k_i}$ is a subkey or "round key")
  - **f** does not need to be invertible, round will be invertible anyway
  - E.g. DES and KASUMI are (essentially) Feistel networks



Feistel networks

- Decryption, round by round
  - (1) We know $\mathbf{L_{i+1}}$ and $\mathbf{R_{i+1}}$ (final values are ciphertext)
  - (2) Since $\mathbf{L_{i+1} = R_i}$, we also know $\mathbf{R_i}$
  - (3) Solve for $\mathbf{L_i}$ as follows:
    - Feistel encryption was $\mathbf{R_{i+1}=L_i}$ **XOR** $\mathbf{f(R_i, k_i)}$
    - Thus, $\mathbf{L_i = R_{i+1}}$ **XOR** $\mathbf{f(R_i, k_i)}$
    - Since $\mathbf{R_{i+1}}$ is known, round keys $\mathbf{k_i}$ are known, $\mathbf{f(R_i, k_i) = f(L_{i+1}, k_i)}$ can be computed => $\mathbf{L_i}$



Substitution-Permutation networks

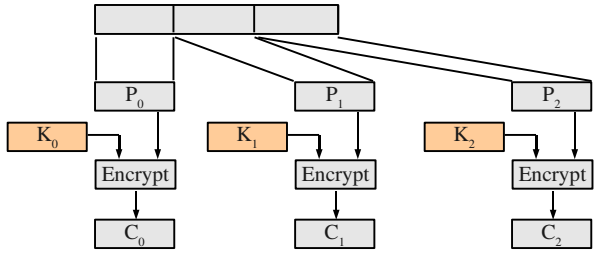
- Substitution-permutation networks (SP-networks)
  - Iterated rounds of substitution ("table lookup") and permutation
- S-boxes (substitution boxes)
  - Invertible functions, provide "confusion"
  - Changing one input bit changes (on the average) half of the output bits (i.e. sensitive to input)
- P-boxes (permutation boxes)
  - Invertible functions which rearrange bits, provide "diffusion"
  - P-boxes rearrange bits so that changes propagate as quickly as possible to as many as possible S-boxes
  - => Avalanche effect, magnified on each round
- Round keys are often combined with e.g. XOR
  - Round keys are derived from a "master key" by key setup alg.
  - XOR is, of course, invertible

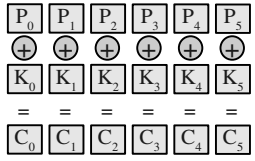
Block cipher security

- Security of the block encryption function
  - "Raw" block encryption function defines the permutation between plaintext and ciphertext spaces, and defines the security of a certain block cipher
  - Obtaining key even when plaintext and ciphertext known (or chosen) should be infeasible
  - Effective key length large enough (e.g. >80 bits)
- Mode of operation
  - Repeated ciphertext leaks information => avoid collisions
  - A good mode of operation should be used (later this lecture)
  - Large block size improves security (e.g. DES=64 => AES=128)
- Resistance against known (non-brute-force) attacks
  - Differential and linear cryptanalysis
  - Algebraic attacks
  - Weak keys
  - Etc...
  - => Block cipher should resist all known attacks, and hopefully be secure against unknown future attacks (margin of security – usually some extra rounds)

Stream cipher

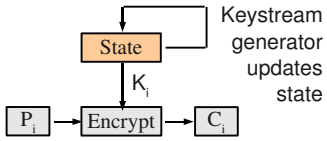
- Stream cipher encryption
  - Plaintext is processed in (small) blocks, typically bits or bytes
  - No padding is required
  - Each bit or byte is encrypted using **a different key** => keystream
  - The encryption function is typically trivial (e.g. XOR), while the keystream generation is the cryptographic part
  - Keystream generator state is initialized using Initial Value
- Decryption is straightforward





Stream cipher

- Stream ciphers approximate one-time pad
  - Keystream approximates a truly random sequence; it should be computationally infeasible to predict the sequence
- Stream ciphers are sometimes called *state ciphers*
  - Each ciphertext element is produced by *combining* current state (keystream element) with a plaintext element
- Keystream can be generated in two ways
  - Independently of plaintext
    - => Synchronous stream cipher
  - Based on **N** previous elements of ciphertext
    - => Self-synchronizing stream cipher



- State must be initialized to start encryption or decryption
  - Initial Value (IV), keystream generator state, public or secret
  - Initial value **must not repeat**, otherwise security disaster

Stream cipher

- More detailed definition
  - Message: $\mathbf{m=b_1b_2b_3...}$, where each $\mathbf{b_i}$ consists of n bits
  - Encryption function: $\mathbf{c = E_k(b)}$, encryption of block **b** with key **k**
  - Decryption function: $\mathbf{b = D_k(c) = E_k^{-1}(c)}$
  - Keystream: $\mathbf{k = k_1k_2k_3...}$
  - Each block of the message is encrypted using a different key, $\mathbf{k_i}$
  - => $\mathbf{E_k(m) = E_{k1}(b_1)\ E_{k2}(b_2)\ E_{k3}(b_3)\ ...}$
- Block cipher is theoretically a stream cipher
  - Keystream is simply $\mathbf{k = k_1k_1k_1...}$, where $\mathbf{k_1}$ is the block cipher key
  - This point of view is theoretical and impractical
- Examples
  - RC4, GSM A5, 3gpp f8 (uses KASUMI block cipher)

Stream cipher advantages & disadvantages

- Stream cipher advantages
  - Good performance in software, simplicity in hardware
  - Can be applied one bit or byte at a time, no need to pad
  - Synchronous stream cipher – no error propagation
  - Encryption function is often trivial (XOR) while keystream generation is cryptographically interesting => modular design
- Stream cipher disadvantages
  - Care must be taken to avoid reuse of IV, reuse is a fatal problem
  - Synchronous stream cipher allows precise flipping of plaintext bits (if used without message integrity protection)
  - Synchronous stream cipher can lose synchronization if even one bit of ciphertext lost in transmission

Stream cipher security

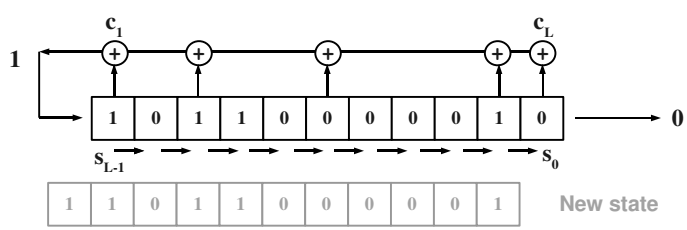
- Security goals – revolve around high quality keystream
  - Long period before keystream repeats (ideally: never)
  - For a fixed key, initial value must not repeat
  - For a fixed initial value, each key yields a different keystream
- Some attacks
  - Known / Chosen plaintext + Ciphertext => Keystream
    - May be used further to analyze cipher
    - Good cipher should withstand
  - Chosen ciphertext
    - May be useful in some cases to analyze the underlying algorithm
    - Especially useful against self-synchronizing ciphers
  - Forcing the keystream generator to a specific state
    - E.g. a state which makes behavior easier to analyze
  - (continues on next slide)

Stream cipher security

- Attacks (continued)
  - Predicting key stream (probabilistically)
    - If key stream can be predicted with some accuracy (i.e. distinguished from random), (probabilistic) knowledge of plaintext possible
    - Extreme example: bad keystream with 10% “1” and 90% “0”, encounter ciphertext bit “1” => with 90% probability plaintext was “1”
    - Any deviation from non-predictable true random leaks probabilistic information about plaintext

Stream cipher design

- Common tool – Linear Feedback Shift Register (LFSR)
  - Use fixed state of **L** bits, generate (weak) pseudorandom bits
- Definitions
  - State bits (stages), $\mathbf{s_0}$ to $\mathbf{s_{L-1}}$ / Taps $\mathbf{c_1}$ to $\mathbf{c_L}$ – tap active if $\mathbf{c_i=1}$
- Operation
  - Generate next bit (s_L) by XORing state bits at tap positions
    - $s_L = c_1 s_{L-1} + c_2 s_{L-2} + \ldots + c_L s_0$
  - Output $\mathbf{s_0}$, and shift right, replacing $\mathbf{s_{L-1}}$ with generated bit ($\mathbf{s_L}$)

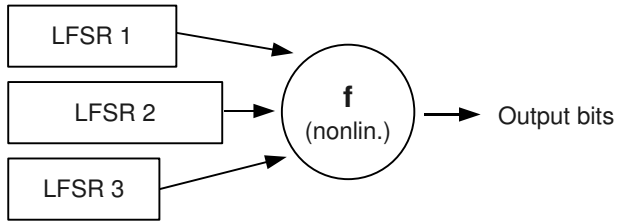


Stream cipher design

- LFSR connection polynomial
  - Taps can also be stated using a connection polynomial
    - $f(x) = 1 + c_1 x + c_2 x^2 + \ldots + c_L x^L$
  - Binary coefficients of the polynomial give tap positions
  - The polynomial can be analyzed, it describes the LFSR as well
- Some LFSR properties
  - Maximum cycle length $\mathbf{2^L-1}$ (**L** = number of state bits)
    - $\mathbf{2^{L-1}-1}$ zeroes and $\mathbf{2^{L-1}}$ ones (almost 50%-50%)
    - One string of **L** consecutive “**1**” bits
    - One string of **L-1** consecutive “**0**” bits (**L** “0” => all zero)
    - Thus, if you know **L** and find **L** consecutive “**1**” bits => next keystream bit is “**0**”
  - Maximum cycle length achieved using so called *primitive polynomials* as connection polynomials

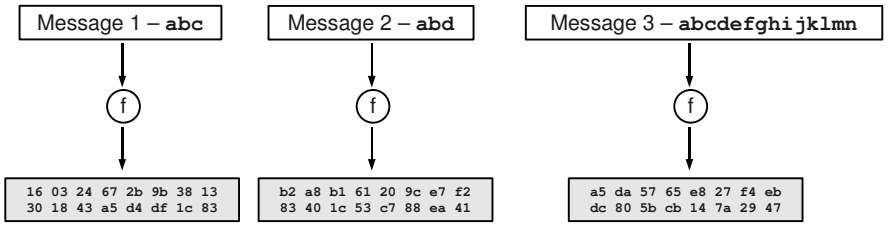
Stream cipher design...

- A single LFSR is not very strong by itself
  - Enough keystream discovered => can usually solve for “taps” (linear set of binary equations)
- Multiple LFSRs are usually combined to create keystream
  - Nonlinear combination of multiple LFSRs
  - Nonlinear filter – nonlinear function of stages of a single LFSR
  - Clock controlled generator – LFSRs are advanced (“clocked”) in an irregular manner and combined in some way
    - Example: GSM Encryption A5/1



Hash function

- Hash function "fingerprints" a message
  - "Fingerprint" (hash) has a fixed size, e.g. 128 or 160 bits
  - Very sensitive to input – one bit change in message causes roughly 50% of hash bits to change
- Uses
  - Hash a plaintext message + secret key => authenticate message
  - Identify a message uniquely with fixed storage size
  - Digital signatures (asymmetric crypto, later)



Hash function

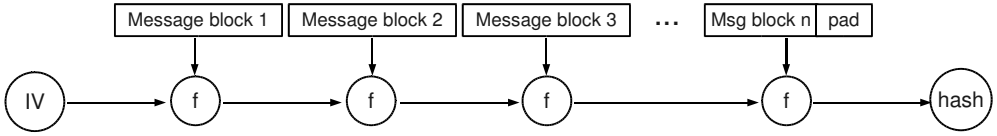
- More detailed definition
  - **X** = set of possible messages, usually arbitrary size
  - **Y** = set of message digests, usually fixed size
  - Hash function h: **X** → **Y**
  - E.g. MD5 (|**Y**| = 2^{128}), SHA1 (|**Y**| = 2^{160})
- Hash functions are usually defined to be unkeyed
  - Message Authentication Codes (MACs) are sometimes called "keyed hash functions"
  - Hash function family, keyspace **K**, for each key **k**, h_k: **X** → **Y**
- Hash functions usually require padding
  - Functions usually operate on blocks of data, e.g. 512 bits
  - Padding and other "termination bits" are required to make hash function secure

Hash function security

- Preimage resistance
  - Given hash **h**, impossible to find **P** such that **H(P)=h**
  - In other words – cannot find *any message* for a given hash
- Second preimage resistance
  - Given **P**, impossible to find **P!=P'** such that **H(P)=H(P')**
  - In other words – cannot find *another message* with same hash
- Collision resistance
  - Impossible to find *any* **P** and **P'**, **P!=P'** such that **H(P)=H(P')**
  - In other words – cannot find *any two messages* with same hash
  - Birthday attack (for collisions)
    - How many random hashes are required to get a 50% probability of finding a collision ? => Answer – roughly **1.18** $n^{1/2}$
    - Thus, for 50% protection against 2^{64} bit brute force, hash size must be more than 128 bits (rough estimate)

Hash function design

- Merkle-Damgård structure commonly used
  - Message processed (compressed) in fixed size blocks
    - Requires padding of message
  - Compression function **f** combines state + block => new state
  - Initialization vector (fixed) is initial state – hash is last state
- Security
  - If compression function **f** is collision resistant, overall construction is collision resistant (Merkle & Damgård)
- Commonly used
  - MD5 and SHA1; block size 512 bits

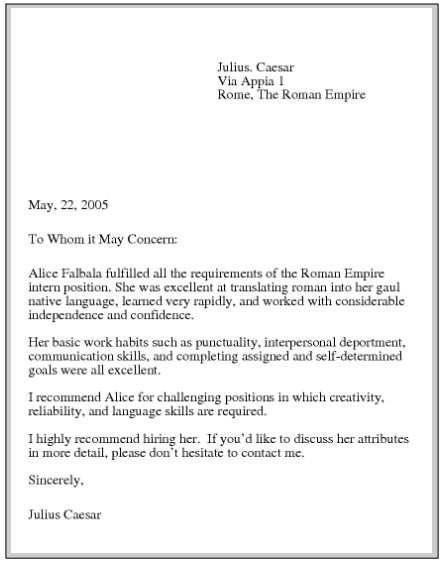


Hash function security crisis

- Hash functions are currently in a bit of a crisis
  - Popular hash functions are MD5 (128 bits) and SHA1 (160 bits)
  - Both have collision attacks faster than birthday paradox
  - => We'll also look at alternative hash functions in the course
- Given one collision, use **length extension** to find more
  - **h(p) = h(p')** => **h(p | pad | q) = h(p' | pad | q)**
  - This is a "feature" of many hash constructions
- This can be turned into practical attacks – example
  - Two PostScript (PS) files with same hash (malicious construct)
  - File 1 – **p** | pad | <if prefix is '**p**', output document 1, else document 2>
  - File 2 – **p'** | pad | <if prefix is '**p**', output document 1, else document 2>
  - Suffix (actual PS code) is identical, outputs different documents
  - Similar approach for EXE files, etc.
  - Important point – random collisions can lead to practical attacks

Example MD5 collision attack

```
sva@sarkofagi:~/svn/T110.5210/2005/lectures/md5coll$ md5sum *
a25f7f0b29ee0b3968c860738533a4b9  letter_of_rec.ps
a25f7f0b29ee0b3968c860738533a4b9  order.ps
```



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

To Whom it May Concern:

Alice Falbala fulfilled all the requirements of the Roman Empire intern position. She was excellent at translating roman into her gaul native language, learned very rapidly, and worked with considerable independence and confidence.

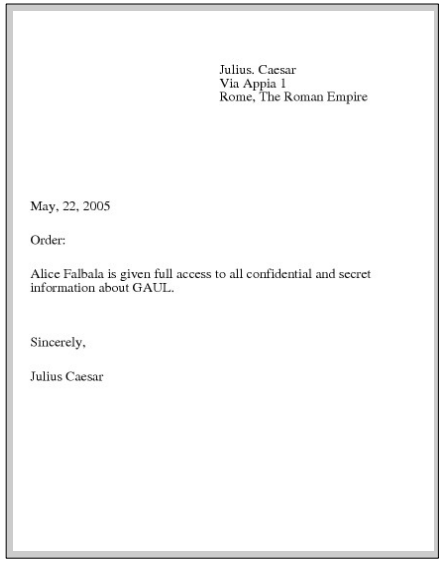
Her basic work habits such as punctuality, interpersonal deportment, communication skills, and completing assigned and self-determined goals were all excellent.

I recommend Alice for challenging positions in which creativity, reliability, and language skills are required.

I highly recommend hiring her. If you'd like to discuss her attributes in more detail, please don't hesitate to contact me.

Sincerely,

Julius Caesar



Julius. Caesar
Via Appia 1
Rome, The Roman Empire

May, 22, 2005

Order:

Alice Falbala is given full access to all confidential and secret information about GAUL.

Sincerely,

Julius Caesar

Source – http://www.cits.rub.de/MD5Collisions/

Message Authentication Code (MAC)

- MAC function authenticates a message
  - Sender and receiver share a secret (symmetric) key, **k**
  - Sender sends message **M** as: **(M, MAC$_k$(M))**
  - Receiver computes **MAC$_k$(M)** based on received message, and compares against sent MAC value
  - If either message **M** or key **k** other than expected, comparison fails with high probability
- MAC functions are used to protect integrity of data
  - Because MACs are keyed, works against active attackers
  - Attacker cannot change message without affecting MAC value
  - Without the key, correcting the MAC value after modification is computationally infeasible

MAC Security

- Resistance against **existential forgery**
  - Given a large amount of correct (message, MAC) pairs, attacker should not be able to construct a fresh valid pair **(M, MAC$_k$(M))**, where **M** is any message (meaningful or not)
  - Weakest attack => Strongest criterion
- Finding the key
  - Breaking the key **k** should be infeasible even with large amount of correct (message, MAC) pairs
  - Breaking an "expired key" is not a catastrophe

MAC Design

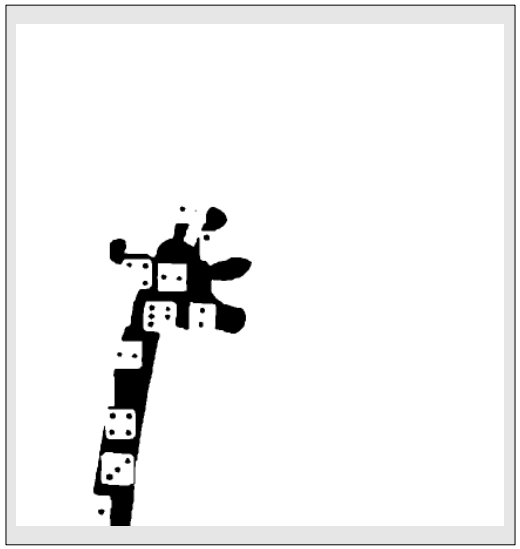
- MAC design is similar to hash function design
  - HMAC is a MAC which uses a hash function as a component
  - But there are also MACs which don't use an underlying hash algorithm (e.g. MACs based on block or stream ciphers)
- Typically iterative design
  - Initial state with fixed values, possibly depending on key
  - Message is padded and processed in fixed size chunks
  - Current state and message chunk *compressed* into new state
  - Final state is (optionally) postprocessed and then output
- Examples
  - HMAC (next lecture)
  - CBC-MAC (later this lecture)

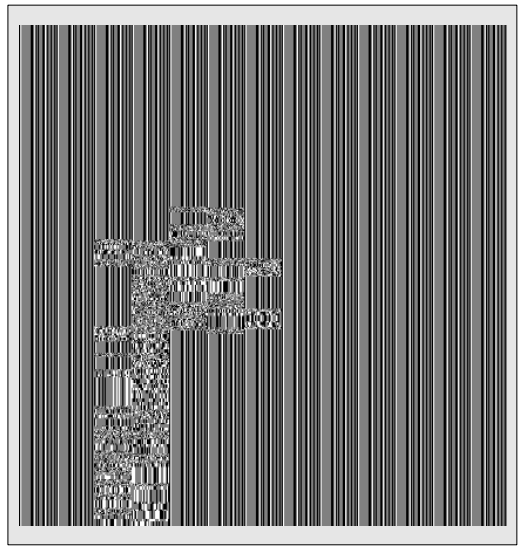
Block cipher modes of operation

Overview

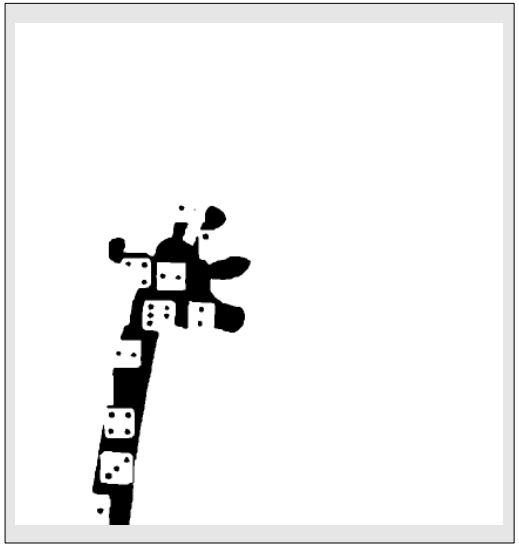
- The “dirty secret” of block ciphers – they leak information
  - If two ciphertexts equal, inputs to block cipher function equal
    - If no mode of operation (input = plaintext), plaintext blocks equal
    - Even with a mode of operation, ciphertext collisions are unwanted
  - Block cipher modes try to avoid this and other information leaks
- Some block cipher **modes of operation** (others exist)
  - Electronic codebook (ECB) (= direct use)
  - Cipher-block chaining (CBC)
  - Counter mode (CTR)
  - Cipher feedback (CFB)
  - Output feedback (OFB)
- Differences between methods
  - Security
  - Error propagation and (re)synchronization
  - Parallelism in encryption and decryption

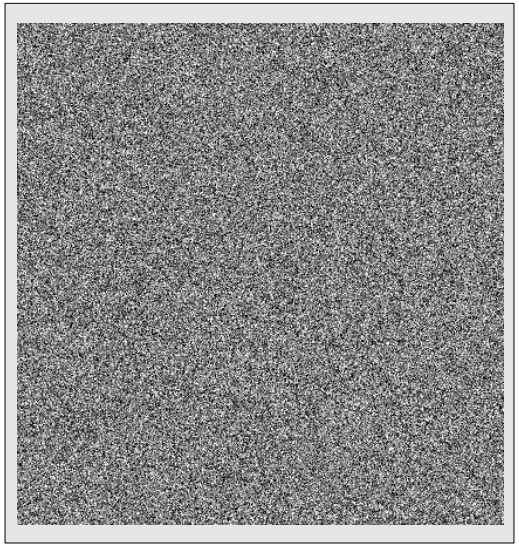
Giraffe bitmap – plain and DES-ECB





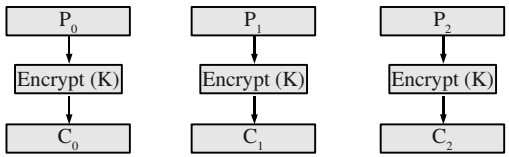
Giraffe bitmap – plain and DES-CBC





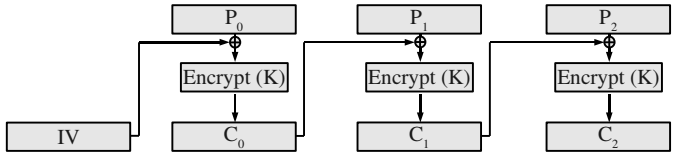
Electronic CodeBook (ECB)

- Encrypt each block separately using the same key
  - If plaintexts are equal, ciphertexts are equal
  - => Leaks lots of information about the plaintext (low security)
- Error propagation
  - Intra-block propagation, $\mathbf{C_a}$ corrupted => $\mathbf{P_a}$ corrupted
- Synchronization
  - Lost ciphertext blocks are not a problem
- Parallelism
  - Fully parallel in both encryption and decryption



Cipher-Block Chaining (CBC)

- Encrypt (plaintext XOR previous ciphertext)
  - For first block, use an Initialization Vector (IV)
- Error propagation
  - If ciphertext $\mathbf{C_a}$ is corrupted, both $\mathbf{P_a}$ and $\mathbf{P_{a+1}}$ corrupted on decr.
- Synchronization
  - To decrypt block $\mathbf{C_a}$, you need both $\mathbf{C_a}$ and $\mathbf{C_{a-1}}$
  - Otherwise can resynchronize
- Parallelism
  - Serialized encryption, almost random access decryption



Cipher-Block Chaining (CBC)

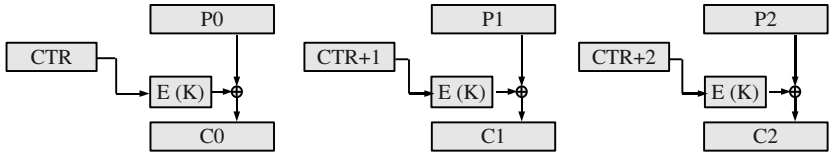
- If two ciphertexts, C_a and C_b equal, we have a problem
  - $\mathbf{C_a = E_k(C_{a-1} \text{ XOR } P_a) = E_k(C_{b-1} \text{ XOR } P_b) = C_b}$
  - <=> $\mathbf{C_{a-1} \text{ XOR } P_a = C_{b-1} \text{ XOR } P_b}$
  - <=> $\mathbf{C_{a-1} \text{ XOR } C_{b-1} = P_a \text{ XOR } P_b}$
  - We thus learn something about relationship between $\mathbf{P_a}$ and $\mathbf{P_b}$
- To avoid ciphertext collisions, limit encrypted blocks for key
  - Birthday paradox => roughly $\mathbf{sqrt(2^{blocksize}) = 2^{blocksize/2}}$
  - E.g. DES => $\mathbf{2^{32}}$ blocks (32 gigabytes of data)
  - Ciphertext collision only a problem if encryption key is the same => *rekeying* procedure
- Another perspective on CBC
  - Random XOR value for each input block would be nice
  - Previous ciphertext block is quite close to a random value

CBC MAC

- CBC mode of operation can be used to construct a MAC
  - Starting with IV = 0, process message in block sized chunks
    - Pre-processing, proper padding
  - Input for each encryption is as in CBC mode
    - = previous cipher block XOR current plaintext block
  - Final cipher block is CBC MAC output
    - Optional post-processing of last block (e.g. extra encryption)
- CBC-MAC requires careful design to be secure
  - Proper pre- and post-processing are required for security
  - Look up the literature for secure CBC-MAC designs

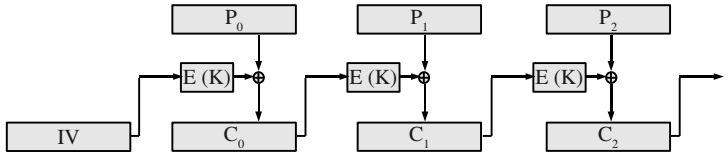
Counter Mode (CTR)

- Encrypt increasing counter, XOR keystream with plaintext
  - Block cipher converted into a stream cipher
  - Receiver must be able to regenerate start value of counter (CTR)
  - Counter may be split into e.g. packet ctr and intra-packet ctr
- Error propagation
  - No error propagation, corrupt $\mathbf{C_a}$ corrupts $\mathbf{P_a}$ (bit flipping attack)
- Synchronization
  - If bits lost, keystream will not match ciphertext
- Parallelism
  - Fully parallel, allows precomputation



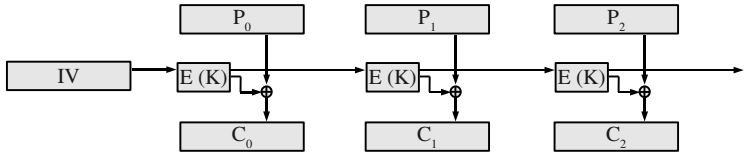
Cipher feedback (CFB)

- Encrypt previous ciphertext, XOR result with plaintext
  - Basically a stream cipher construction, but keystream dependent on ciphertext (and thus plaintext)
- Error propagation
  - Error in $\mathbf{C_a}$ => errors in both $\mathbf{P_a}$ and $\mathbf{P_{a+1}}$
- Synchronization
  - Like CBC, resynchronizes ($\mathbf{P_a}$ needs $\mathbf{C_a}$ and $\mathbf{C_{a-1}}$)
- Parallelism
  - Keystream generation cannot be parallelized, and cannot be precomputed



Output feedback (OFB)

- Use block cipher as an independent keystream generator
  - Start with a certain IV, and encrypt the result successively; XOR plaintext with keystream
  - Keystream is independent of the plaintext/ciphertext (cf. CTR)
- Error propagation
  - No propagation; error in $\mathbf{C_a}$ => error in $\mathbf{P_a}$ (bit flipping attack)
- Synchronization
  - If bits lost, cannot resynchronize
- Parallelism
  - Keystream gen. cannot be parallelized (can be precomputed)



Combined modes

- Combine encryption and authentication
  - Still under development
- Some combined modes
  - CCM (Counter with CBC-MAC) (Whiting, Ferguson, Housley)
  - Galois/Counter Mode (GCM) (McGrew, Viega)
  - Offset Codebook (OCB) (Rogaway, Bellare, Black, Krovitz)
  - Carter-Wegman auth + CTR (CWC) (Kohno, Viega, Whiting)
- Block cipher modes of operation research is still active

Complete symmetric cryptosystem

- For a useful symmetric cryptosystem, you need at least
  - An encryption algorithm + a mode of operation (block cipher)
  - An integrity protection algorithm (MAC)
  - A key management system (initial key, rekeying)
- For example, IPsec (in one configuration)
  - 3DES in CBC mode
  - HMAC-SHA1-96 (HMAC-SHA1 truncated to 96 bits)
  - Internet Key Exchange (IKE) or static keys
- Encryption and integrity protection can be combined
  - Combined mode – authenticated encryption
  - Combined modes not widely used at present
  - Instead, separate encryption and integrity protection using MAC

Summary

- Basic symmetric (and unkeyed) primitives
  - Block and stream ciphers
  - Examples of trivial ciphers
  - Structure of modern block ciphers
  - Hash functions
- Block cipher modes of operation
- Assignments
  - Vigenere [crypto.implementations.Vigenere]
  - Generalized CBC [crypto.implementations.GenericCbc]