Cryptographic primitives

-- block ciphers
-- stream ciphers
-- public-key encryption schemes
-- hash functions
-- MAC functions
-- digital signature schemes

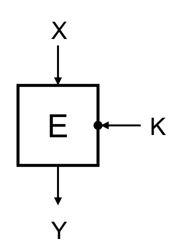
Block ciphers

- a block cipher is a function

$$E: \{0, 1\}^n \times \{0, 1\}^k \rightarrow \{0, 1\}^n,$$

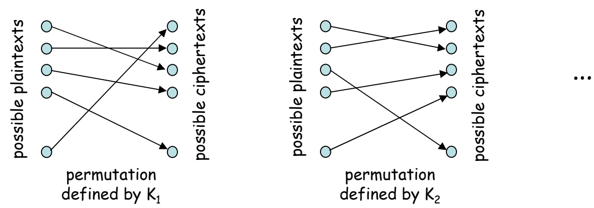
  such that for each $K \in \{0, 1\}^k$, $E(X, K) = E_K(X)$
  - is an invertible mapping from $\{0, 1\}^n$ to $\{0, 1\}^n$ ($E_K^{-1}(Y) = D_K(Y)$)
  - cannot be efficiently distinguished from a random permutation

- terminology
  - X – plaintext block
  - Y – ciphertext block
  - E – encryption/coding alg.
  - D – decryption/decoding alg.
  - K – key
  - $\mathcal{K} = \{0, 1\}^k$ – key space

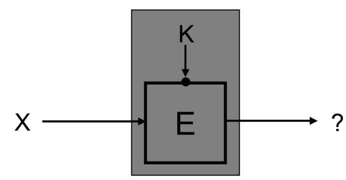


Another view on block ciphers

- a block cipher is a family of permutations (where each member is defined by a key)



- these permutations "look random"
  - unpredictability of the output (even parts of it, and even when some input-output pairs are known)
  - avalanche effect (when changing one bit in the input, each output bit changes with probability ~1/2)



Applications of block ciphers

- primarily:
  - encryption of data (of any size) → confidentiality services
- can also be used as a building block for
  - MAC functions → integrity and message authentication services
  - hash functions
  - PRNGs (Pseudo-Random Number Generator)
  - key-stream generators for stream ciphers

Some examples

- many of them have been proposed and are in use
  - AES (Rijndael), DES, RC5, Blowfish, Skipjack, IDEA, ...
- how to choose?
  - design assumptions vs. application requirements
    - e.g.: is it optimized for hardware or software implementations, or can be used in both
  - efficiency
    - speed
    - memory size
    - code size (or number of gates)
  - security
    - openness of specification (Kerckhoffs' principle)
    - key size
    - algebraic properties
    - complexity of best known attacks
  - patent issues

Kerckhoffs' principles

- Auguste Kerckhoffs, La cryptographie militaire, Journal des sciences militaires, Vol. IX, Janvier 1883.

  → see the principles on page 12 …

  - Principle 2 says that it must be assumed that the encryption algorithm is known to the "adversary". In other words, the security of the system cannot depend on the secrecy of the algorithm (security by obscurity).
  - advantages of adherence to Kerckhoffs 2^{nd} Principle:
    - published designs undergo public scrutiny
    - it is better if security flaws are revealed by "ethical hackers"
    - secret algorithms are vulnerable to the reverse engineering of the code
    - public designs allow for standards
  - Principle 1 is also interesting: it says that a cipher must be *practically*, if not mathematically, indecipherable.

Exhaustive key search attacks and the key size

- given a small number of plaintext-ciphertext pairs encrypted under a key K, K can be recovered by exhaustive key search with 2^{k-1} processing complexity (expected number of operations)
  - input: (X, Y), (X', Y'), …
  - progress through the entire key space
    - for each trial key K', decrypt Y
    - if the result is not X, then throw away K'
    - if the result is X, then check the other pairs (X', Y'), …
    - if K' does not work for at least one pair, then throw away K'
  - if K' worked for all pairs (X, Y), (X', Y'), …, then output K' as the target key
  - on average, the target key is found after searching half of the key space

- if the plaintexts are known to contain redundancy, then even ciphertext-only exhaustive key search is possible with a relatively small number of ciphertexts

→ 2^{k-1} must be sufficiently large

Large numbers

time until next ice age…………………………… 2^{39} seconds
time until the sun goes nova…………………… 2^{55} seconds
age of the planet………………………………… 2^{55} seconds
age of the Universe……………………………… 2^{59} seconds

number of atoms in the planet……………… 2^{170}
number of atoms in the sun………………….. 2^{190}
number of atoms in the galaxy……………… 2^{223}
number of atoms in the Universe ………….. 2^{265}
(dark matter excluded)

volume of the universe………………………….. 2^{280} cm^3

(source: Schneier, Applied Cryptography, 2nd ed., Wiley 1996)

Algebraic attacks

- weaknesses in the algebraic structure of a block cipher may lead to attacks that are *substantially* more efficient than the exhaustive key search attack
- attack models
  - ciphertext-only attack
  - known-plaintext attack
  - (adaptive) chosen-plaintext attack
- attack complexity measures
  - data complexity
    - expected number of input data units required for the attack
  - storage complexity
    - expected number of storage units required
  - processing complexity
    - expected number of "basic operations" required to process input data and/or fill storage with data
    - parallelization may reduce attack time but not processing complexity!

Examples for algebraic attacks

- linear cryptanalysis (LC) against DES
  - requires "only" ~2^{43} known plaintext-ciphertext pairs
  - could work in a ciphertext only model if plaintexts are redundant (e.g., contain parity bits)
- differential cryptanalysis (DC) against DES
  - requires "only" ~2^{47} chosen plaintext-ciphertext pairs
- anecdote:
  - DC and LC was discovered in the early 90's by academic researchers
  - one of the designers of DES announced that they knew about DC back in the 70's and optimized the DES S-boxes against it
  - it seems, however, that DES can be improved with respect to LC (apparently the designers of DES were not aware of this attack at that time)

Exercise

- complementation property of DES:

  $$Y = DES_K(X) \text{ implies } Y^* = DES_{K^*}(X^*)$$

  where X* denotes the bitwise complement of X

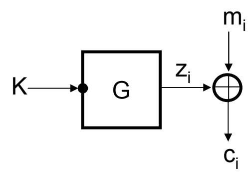
- How can this be used to reduce the complexity of exhaustive key search from 2^{55} to 2^{54}?

Solution

- assume an attacker can mount a chosen-plaintext attack
- the attacker chooses a plaintext X, and obtains $Y_1 = DES_K(X)$ and $Y_2 = DES_K(X^*)$
- by the complementation property, the attacker knows that $DES_{K^*}(X) = Y_2^*$
- the attacker then runs an exhaustive key search
  - for each trial key K', he computes $Y' = DES_{K'}(X)$
    - if $Y' = Y_1$, then K' is possibly the target key (should be further tested)
    - if $Y' = Y_2^*$, then K'* is possibly the target key (should be further tested)
    - otherwise throw away **both** K' and K'*
- expected number of keys required before success is reduced from 2^{55} to 2^{54}

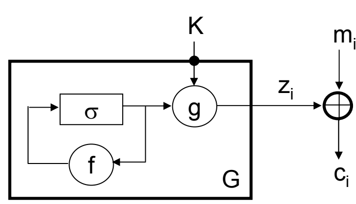
Stream ciphers

- general model:



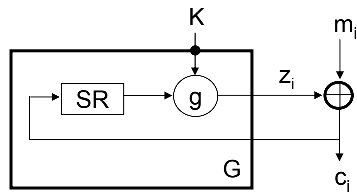
- terminology:
  - m_i – plaintext character
  - c_i – ciphertext character
  - z_i – key-stream character
  - K – key (seed)
  - G – key-stream generator
- application:
  - encryption of data → confidentiality services
  - PRNGs (Pseudo-Random Number Generator)
- examples:
  - LFSR based (typically hardware), RC4 (software)

Synchronous stream ciphers



- the key stream is generated independently of the plaintext and of the ciphertext
- needs synchronization between the sender and the receiver
  - if a character is inserted into or deleted from the ciphertext stream then synchronization is lost and the plaintext cannot be recovered
  - additional techniques must be used to recover from loss of synch
- no error propagation
  - a ciphertext character that is modified during transmission affects only the decryption of that character

Self-synchronizing stream ciphers



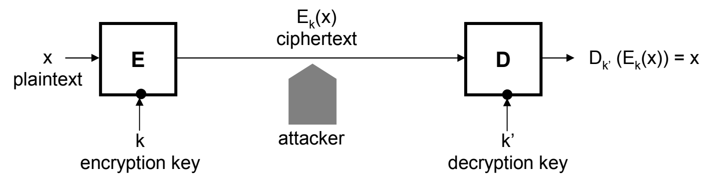
- the key stream is generated as a function of a fixed number of previous ciphertext characters
- self-synchronizing
  - since the size t of the shift register SR is fixed, a lost ciphertext character affects only the decryption of the next t ciphertext characters
- limited error propagation
  - if a ciphertext character is modified, then decryption of the next t ciphertext characters may be incorrect
- ciphertext characters depend on all previous plaintext characters
  - better diffusion of plaintext statistics

More properties

- stream ciphers are usually very efficient
  - fast (especially in hardware)
  - require small memory to store the internal state and the code of the generation and update functions
- the ciphertext always has the same length as the plaintext (in some block encryption modes, the ciphertext is longer)
- in case of synchronous stream ciphers, the large size of the effective state space is important
  - otherwise the key stream starts repeating
  - $c_{i..i+p} + c_{j..j+p} = (m_{i..i+p} + z_{1..p}) + (m_{j..j+p} + z_{1..p}) = m_{i..i+p} + m_{j..j+p}$
- synchronous stream ciphers do not provide any integrity protection !!!
  - an attacker can make changes to selected ciphertext characters and know exactly what effect these changes have on the plaintext
  - the receiver may not notice these changes

The idea of public-key encryption

- classical model of encryption



- symmetric-key encryption: k = k'
  - problem: how to setup the same key at the two ends?
- asymmetric-key encryption: k != k'
  - it is hard (computationally infeasible) to compute k' from k
  - k can be made public (public-key cryptography)
  - anyone can send messages encrypted with k, only the intended receiver can decrypt with k'
  - instead of the secrecy of k, "only" its authenticity and integrity must be ensured

Public-key encryption schemes

- functions (algorithms) and terminology:
  - key-pair generation function $G(\ ) = (K^+, K^-)$
    - K^+ – public key
    - K^- – private key
  - encryption function $E(K^+, X) = Y$
    - X – plaintext
    - Y – ciphertext
  - decryption function $D(K^-, Y) = X$
- typically, the plaintext (and the ciphertext) consists of a few hundred bits → operation is similar to symmetric-key block ciphers
- examples: RSA, ElGamal

Security of public-key encryption schemes

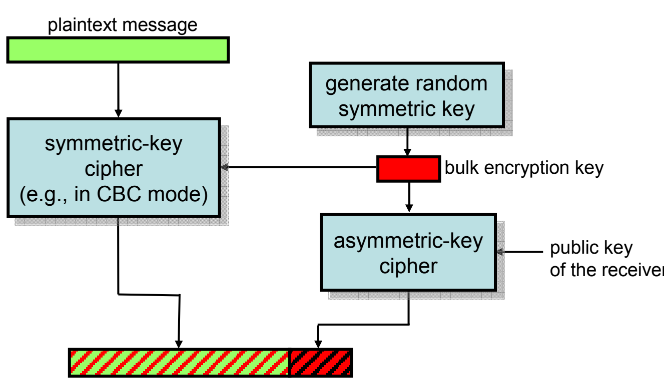
- security is usually related to the difficulty of some problems that are widely believed to be hard to solve (i.e., for which no polynomial time solution exists today), such as
  - factoring:
    given a positive integer N, find its prime factors
  - computing discrete logarithm:
    given a prime p, a generator g of Z_p^*, and an element y in Z_p^*, find the integer x, $0 \leq x \leq p-2$, such that $g^x \bmod p = y$
- sometimes it can even be rigorously proven that breaking the encryption scheme would mean that there exist an efficient solution to the related hard problem (reduction)
  - although widely used practical schemes have no complete proofs

Efficiency considerations

- hard problems are really hard only for large parameters

→ public-key encryption schemes use large number arithmetics, and hence, they are several orders of magnitude slower than the best known symmetric key ciphers (on the same platform)

- to overcome this problem, the following hybrid approach is used in practice:



Semantic security

- an adversary should not be able to choose two plaintexts x1 and x2 and later distinguish between the encryptions of these messages
  - note: symmetric-key block ciphers have this property
  - the problem with public-key encryption is that the adversary can compute the ciphertexts using the public key and trivially distinguish between the encryptions of x1 and x2
- the solution is ***probabilistic encryption***
  - computation of the ciphertext uses some random input → even when the same message is encrypted twice, the outputs will be different
  - some public-key encryption schemes are probabilistic by design (e.g., ElGamal, Goldwasser-Micali)
  - others need pre-formatting of messages which involves the addition of some randomness (e.g., RSA uses PKCS #1 formatting)

Beyond semantic security

- essentially, semantic security is only concerned with a passive attacker
  - it ensures that observed ciphertexts leak no information about the corresponding plaintexts
- a strong active attack model is the chosen-ciphertext attack
  - this means that the adversary has access to a decryption oracle, and he is allowed to send to it any ciphertext except the target ciphertext (that the adversary wants to decrypt)
  - semantically secure schemes (e.g., ElGamal) may not be secure in this model
- the property that ensures resistance against chosen-ciphertext attacks is ***non-malleability***
  - given a ciphertext, it is infeasible to generate another ciphertext such that the corresponding plaintexts are related in a known manner
- non-malleability can be achieved by ***plaintext-aware encryption***
  - e.g., RSA with version 2 of PKCS #1

Hash functions

- a hash function is a function H: {0, 1}* → $\{0, 1\}^n$ that maps arbitrary long messages into a fixed length output
- notation and terminology:
  - x – (input) message
  - y = H(x) – hash value, message digest, fingerprint
- typical application:
  - the hash value of a message can serve as a compact representative image of the message (similar to fingerprints)
    - H is a many-to-one mapping → collisions are unavoidable
    - however, finding collisions are very difficult (practically infeasable)
  - increase the efficiency of digital signatures by signing the hash instead of the message (expensive operation is performed on small data)
- examples:
  - MD5 and SHA-1

Desired properties of hash functions

- ease of computation
  - given an input x, the hash value H(x) of x is easy to compute
- weak collision resistance (2^{nd} preimage resistance)
  - given an input x, it is computationally infeasible to find a second input x' such that H(x') = H(x)
- strong collision resistance (collision resistance)
  - it is computationally infeasible to find any two distinct inputs x and x' such that H(x) = H(x')
- one-way hash function (preimage resistance)
  - given a hash value y (for which no preimage is known), it is computationally infeasible to find any input x such that H(x) = y
- collision resistant hash functions are similar to block ciphers in the sense that they can be modeled as a random function

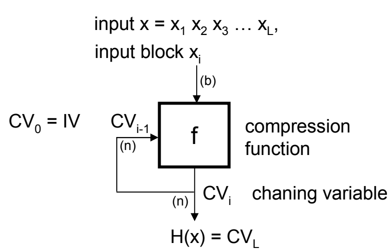
The Birthday Paradox

- fact: when drawing elements randomly (with replacement) from a set of N elements, with high probability a repeated element will be encountered after ~sqrt(N) selections

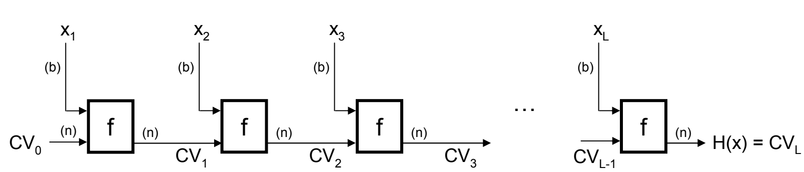
- this fact has a profound impact on the design of hash functions (and other cryptographic algorithms and protocols)!
  - among ~sqrt(2^n) = $2^{n/2}$ randomly chosen messages, with high probability there will be a collision pair
  - → in order to resist birthday attacks, n should be at least 128, but 160 is even better
  - the birthday attack against hash functions is the equivalent of the exhaustive key search attack against block ciphers
  - it is easier to find collisions than to find preimages or 2nd preimages for a given hash value

Iterative hash functions

- operation:
  - input is divided into fixed length blocks
  - last block is padded if necessary
  - each input block is processed according to the following scheme



alternative illustration:



Exercise

Assume that an iterated hash function H has a small output size such that h is not collision resistant (the birthday attack works). One may try to increase the output size by using the last two chaining variables as the output:

$$H'(x) = CV_{L-1}|CV_L$$

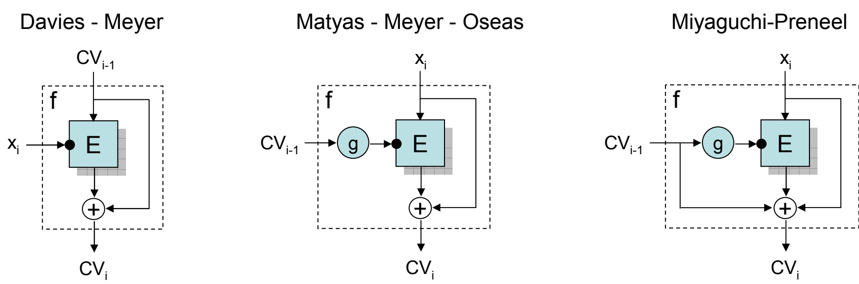
Prove that this is insecure by showing that H' is still not collision resistant.

Solution

- assume that (x, x') is a collision pair for H
  → $CV_L(x) = H(x) = H(x') = CV_L(x')$
- extend x and x' with one block B and observe that
  - $CV_{L-1}(x|B) = CV_L(x)$
  - $CV_{L-1}(x'|B) = CV_L(x')$

  → $CV_{L-1}(x|B) = CV_{L-1}(x'|B)$
  → $CV_L(x|B) = f(CV_{L-1}(x|B), B) = f(CV_{L-1}(x'|B), B) = CV_L(x'|B)$
  → $H'(x|B) = CV_{L-1}(x|B) \mid CV_L(x|B) = CV_{L-1}(x'|B) \mid CV_L(x'|B) = H'(x'|B)$
- we found a collision against H'

Compression functions based on block ciphers



MAC functions

- MAC = Message Authentication Code
- a MAC function is a function MAC: $\{0, 1\}^* \times \{0, 1\}^k \rightarrow \{0, 1\}^n$ that maps an arbitrary long message and a key into a fixed length output
  – can be viewed as a hash function with an additional input (the key)
- terminology and usage:
  – the sender computes the MAC value M = MAC(m, K), where m is the message, and K is the MAC key
  – the sender attaches M to m, and sends them to the receiver
  – the receiver receives (m', M')
  – the receiver computes M" = MAC(m', K) and compares it to M'; if they are the same, then the message is accepted, otherwise rejected
- services:
  – **message authentication and integrity protection:** after successful verification of the MAC value, the receiver is assured that the message has been generated by the sender and it has not been altered
- examples:
  – HMAC, CBC-MAC schemes

Desired properties of MAC functions

- ease of computation
- key non-recovery
  - it is computationally infeasible to recover the secret key K, given one or more message-MAC pairs (m_i, M_i) for that K
- computation resistance
  - given zero or more message-MAC pairs (m_i, M_i), it is computationally infeasible to find a valid message-MAC pair (m, M) for any new message $m \neq m_i$
  - computation resistance implies key non-recovery but the reverse is not true in general

A note on the key size and the MAC value size

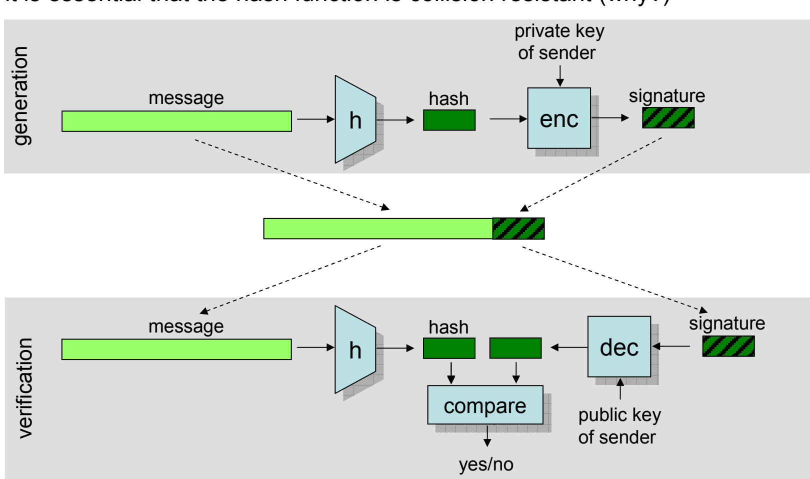
- guessing a correct MAC for a given message or a message for a given MAC have probability 2^{-n}
  - an important difference between MACs and hash functions that message-MAC guesses cannot be verified off-line, but need access to the key or to an oracle
- brute force attack on the key space has complexity 2^k
- thus, $\min(2^k, 2^n)$ should be sufficiently large
- how the birthday attack would work here?

Digital signature schemes

- functions (algorithms) and terminology:
  - key-pair generation function $G() = (K^+, K^-)$
    - K^+ -- public key
    - K^- -- private key
  - signature generation function $S(K^-, m) = s$
    - m – message
    - s – signature
  - signature verification function V: $V(K^+, m, s)$ = accept or reject
- services:
  - **message authentication and integrity protection:** after successful verification of the signature, the receiver is assured that the message has been generated by the sender and it has not been altered
  - **non-repudiation of origin:** the receiver can prove this to a third party (hence the sender cannot repudiate)
- examples: RSA, DSA, ECDSA (shorter key and signature length!)

"Hash-and-sign" paradigm

- public/private key operations are slow
- increase efficiency by signing the hash of the message instead of the message
- it is essential that the hash function is collision resistant (why?)



Security of digital signature schemes

- as in the case of public-key encryption, security is usually related to the difficulty of solving the underlying hard problems
- attack objectives:
  - existential forgery
    - attacker is able to compute a valid signature for at least one message
  - selective forgery
    - attacker is able to compute valid signatures for a particular class of messages
  - total break
    - the attacker is able to forge signatures for all messages or he can deduce the private key
- attack models:
  - key-only attack
  - known-message attack
  - (adaptive) chosen-message attack