Hash functions

- (Cryptographic) *hash function*: deterministic function mapping arbitrary length inputs to a short, fixed-length output (sometimes called a *digest*)
- Hash functions can be *keyed* or *unkeyed*
  - In practice, hash functions are unkeyed
  - We will assume unkeyed hash functions for simplicity

Collision-resistence

- Let $H : \{0,1\}^* \to \{0,1\}^\ell$ be a *hash function*
- A *collision* is a pair of distinct inputs x, x' such that $H(x) = H(x')$.
- **Theorem 7.1** H is *collision-resistant* if it is infeasible to find a collision in H.
- What is the best "generic" collision attack on a hash function $H; \{0,1\}^* \to \{0,1\}^\ell$?
  - Note that collisions are guaranteed to exist!
  - If we compute $H(x_1), \ldots, H(x_{2^\ell+1})$, we are guaranteed to find a collision.
  - Can we do better?

Birthday attacks

- Compute $H(x_1), \ldots, H(x_k)$
  - What is the probability of a collision?
- Related to the so-called *birthday paradox*
  - How many people are needed to have a 50% chance that some two people share a birthday?
- When $k \approx H^{1/2}$, probability of a collision is $\approx 50\%$
  - Birthdays: 23 people suffice!
  - Hash functions: $O(2^{\ell/2})$ hash-function evaluations
- Need $\ell = 2n$-bit output length to get security against attackers running in time 2^n
- Event A: at least two people in the room have the same birthday
  Event B: no two people in the room have the same birthday

  $\Pr[A] = 1 - \Pr[B]$

$$\begin{aligned}\Pr[B] &= \left(1 - \frac{1}{365}\right) \cdot \left(1 - \frac{2}{365}\right) \cdot \cdots \cdot \left(1 - \frac{n-1}{365}\right) \\ &= \prod_{i=1}^{n-1} \left(1 - \frac{i}{365}\right).\end{aligned}$$

  $\Pr[A] = 1 - \prod_{i=1}^{n-1} \left(1 - \frac{i}{365}\right)$

  $p(n; H) := 1 - \prod_{i=1}^{n-1}(1 - \frac{i}{H})$
- Since $e^x = 1 + x + \frac{x^2}{2!} + \cdots$, for $|x| \ll 1$, $e^x \approx 1 + x$

  Thus, we have $e^{-i/H} \approx 1 - \frac{i}{H}$.

  Recall that $p(n; H) := 1 - \prod_{i=1}^{n-1}(1 - \frac{i}{H})$

  This probability can be approximated as

$$p(n; H) \approx 1 - e^{-n(n-1)/2H} \approx 1 - e^{-n^2/2H}.$$

  Let $n(p; H)$ be the smallest number of values we have to choose, such that the probability for finding a collision is at least p. By inverting the expression above, we have

$$n(p; H) \approx \sqrt{2H \ln \frac{1}{1-p}}.$$

确定性的

强 collision resistant : no collision in H

① ↓

second-preimage resistant : given a uniform x, it is infeasible to find an x' with $x \neq x'$ s.t. $H(x) = H(x')$

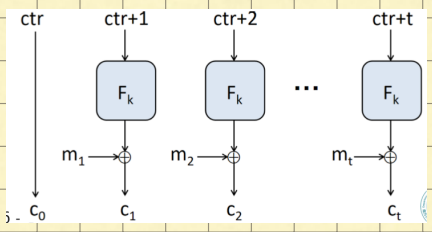
② ↓

弱 preimage resistant : given a uniform y, it is infeasible to find an x s.t. $H(x) = y$.

① given the condition of second-preimage resistant, then (x, x') is a pair of collision.

② given the condition of preimage resistant, then it is a high probability to find $x' \neq x$ s.t. $H(x') = y = H(x)$.

Birthday Bound



- The *birthday bound* comes up in many other cryptographic contexts
- **Example**: *IV* reuse in CTR-mode encryption
  - If k messages are encrypted, what is the chance that some *IV* is used twice?
  - Note: this is much higher than the probability that a *specific* *IV* is used again

Collision resistant hash functions

- CRH vs. PRG
  - CRHs are dual to PRGs, in the sense that the goal is to *shrink* the input as much as possible. Similar to PRGs, if one can shrink the input by even by one bit, one can get a CRH collection that shrinks the bits by an amount of polynoimial.
  - In practice, people usually talk about a *single* hash func. rather than a collection. One can think of this that someone chose the key k once and then fixed the function h_k for everyone to use. In fact, most practical constructions involve some hardwired standardized constants, often known as *IV* that can be thought of as a choice of the key.

Construction of CRH

- Practical constructions of cryptographic hash functions start with a basic block, a.k.a. compression function $h : \{0,1\}^{2n} \to \{0,1\}^n$. The function $H : \{0,1\}^* \to \{0,1\}^n$ is defined as
  $$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$
  when the message is composed of t blocks (and we can pad it otherwise). This is known as the *Merkle-Damgård construction*.

- **Theorem 8.1** (Merkle-Damgård preserves collision resistance) Let H be constructed from h above. Then given two messages $m \neq m' \in \{0,1\}^{tn}$ such that $H(m) = H(m')$ we can efficiently find two messages $x \neq x' \in \{0,1\}^{2n}$ such that $h(x) = h(x')$.

Proof. The intuition behind the proof is that if h is invertible then we could invert H by simply going backwards.

In principle, if a collision for H exists then so does a collision for h.

We look at the computation of $H(m)$ and $H(m')$ and at the first block in which the inputs differ but the output is the same (there must be such a block). This block will yield a collision for h.

$$H(m_1, \ldots, m_t) = h(h(h(m_1, IV), m_2), \ldots, m_t),$$

即h是collision resistant ⇒ H是collision resistant

Suppose H is not CR, we can efficiently find $m' \neq m$, s.t., $H(m') = H(m)$, i.e.,

$h(h(h(m_1, IV), m_2), \ldots, m_t)$
$= h(h(h(m'_1, IV), m'_2), \ldots, m'_t)$

① 若 $h(h(m_1, IV), m_2), \ldots, m_t) \neq h(h(m'_1, IV), m'_2), \ldots, m'_t)$ 此时最外层h发生碰撞.

② else. 继续递归下去, 必然找到碰撞, 因为 $m \neq m'$

The random-oracle (RO) model

- Treat H as a public, *random* function
- Then $H(x)$ is *uniform* for any x
  - Unless the attacker computes $H(x)$ explicitly
- Intuitively
  - Assume the hash function "*is random*"
  - Models attacks that are *agnostic* to the specific hash function being used
  - Security in the real world as long as "*no weaknesses found*" in the hash function

- Formally
  - Choose a *uniform* hash function as part of the security experiment
  - Attacker can *only* evaluate H via *explicit* queries to an oracle
  - Simulate H for the attacker as part of the security proof/reduction
- In practice
  - Prove security in the RO model
  - Instantiate the RO with a "*good*" hash function
  - Hope for the best

Pros & cons of the RO model

- Cons
  - There is no such a thing as a public hash function that "*is random*"
    - Not even clear what this means formally
  - Known counterexamples
    - There are (contrived) schemes secure in the RO model, but insecure when using any real-world hash function
  - Sometimes *over-abused* (arguably)
- Pros
  - No known example of "*natural*" scheme secure in the RO model being attacked in the real world
  - If an attack is found, just replace the hash
  - Proof in the RO model better than no proof at all
    - Evidence that the basic design principles are sound

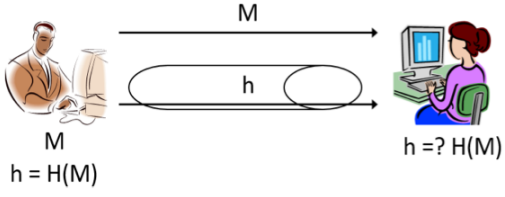
没有哈希函数是真正的随机

Hash functions in practice

- MD5
  - Developed in 1991 by Ron Rivest
  - 128-bit output length
  - Collisions found in 2004, and more recently in 2013, should no longer be used
- SHA-1
  - Introduced in 1995 by NSA
  - 160-bit output length
  - Theoretical analysis indicates some weaknesses
  - Current trend to migrate to SHA-2
  - Collision found by brute force in 2017!
- SHA-2
  - Supports 224, 256, 384, and 512-bit outputs
  - No serious known weaknesses
- SHA-3 / Keccak
  - Result of a public competition from 2008-2012
  - Very different design from SHA-1/SHA-2
  - Supports 224, 256, 384, and 512-bit outputs

Recall

- We showed how to construct a *secure MAC* for short, fixed-length messages based on any PRF/block cipher
- We want to extend this to a *secure MAC* for arbitrary-length messages
  - Before: using CBC-MAC
  - After: using hash functions



Security

- **Theorem 8.2** If the *MAC* is *secure* for fixed-length messages and H is *collision-resistent*, then the previous construction is a *secure* MAC for arbitrary-length messages
- Proof sketch
  Say the sender authenticates M_1, M_2, ...
  - Let $m_i = H(M_i)$

  Attacker outputs forgery (M, t), $M \neq M_i$ for all i

  Two cases:
  - $H(M) = H(M_i)$ for some i
    - Collision in H!
  - $H(M) \neq m_i$ for all i
    - Forgery in the underlying, fixed-length MAC!

攻击可以这两方面入手

secure MAC for fixed length + collision-resistent hash

= secure MAC for arbitrary length

Instantiation

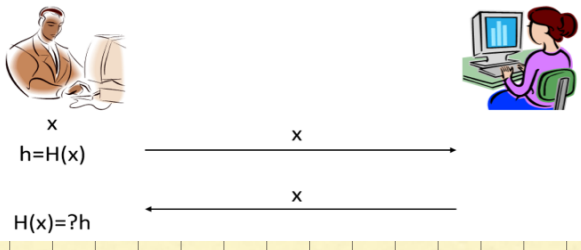
- Hash function + block-cipher-based MAC
  - Block-length mismatch
  - Need to implement two crypto primitives (block cipher and hash function)
- HMAC: constructed entirely from (certain type of) hash functions
  - MD5, SHA-1, SHA-2
  - Not SHA-3
- Can be viewed as following the hash-and-MAC paradigm
  - With (part of the) hash function being used as a PRF

Other application of hash functions

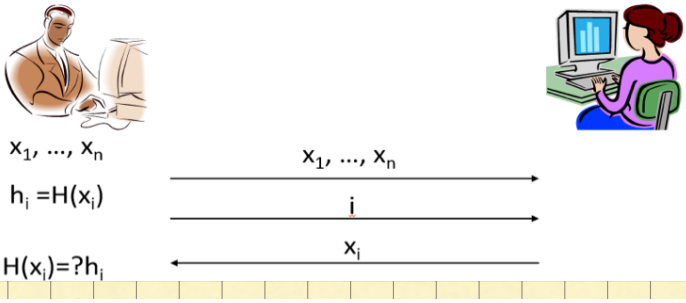
- Hash functions are ubiquitous
  - Collision-resistance ⇒ "fingerprinting"
  - Used as a *one-way function*
  - Used as a "random oracle"
    - Proofs of work
  - E.g., virus scanning
  - E.g., deduplication

Fingerprinting

- E.g., file integrity
  - Assuming it is possible to get a reliable copy of $H(x)$ for file x
  - Note: different from integrity in the context of message-authentication codes
- How to outsource files to an untrusted server?



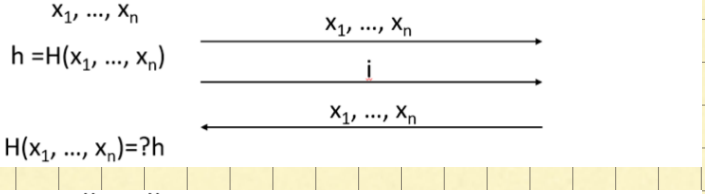
Outsourced storage

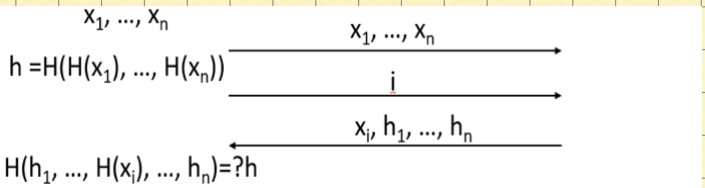


n files outsourcing:

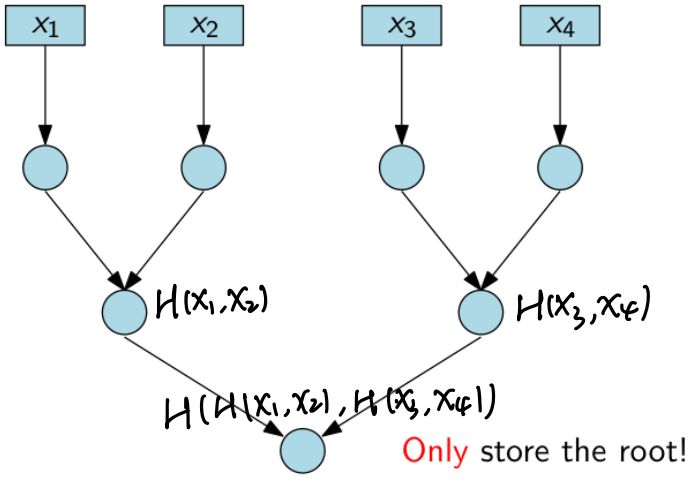
远程存3client storage个hash 信道通信

	client storage	communication
	$O(n)$	$O(\|x\|)$
	$O(1)$	$O(n\cdot\|x\|)$
	$O(1)$	$\|x_i\|+O(n)$





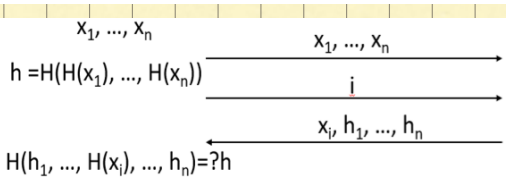
Merkle tree



Only store the root!

$O(\log n)$ communication/computation!

所有文件都在叶子节点.
中间节点都是左右子节点的hash
client只需要存根节点的值.



$|x_i| + O(n)$ communication!

- Using a Merkle tree, we can solve the outsourcing problem with $O(1)$ client storage and $|x| + O(\log n)$ communication.

Server发x_i包时，需同时发其兄弟节点x_j.
再发多个hash

Key derivation

- Consider deriving a (shared) key from (shared) high-entropy information
  - E.g., biometric data
  - E.g., generating randomness
- Cryptographic keys must be *uniform*, but shared data is only high-entropy

Min-entropy

- Let X be a distribution
- The *min-entropy* of X (measured in bits) is
  $$H_\infty(X) = -\log \max_x \{\Pr[X = x]\}$$
  - I.e., if $H_\infty(X) = n$, then the probability of guessing x sampled from X is (at most) 2^{-n}
- Min-entropy is more suitable for crypto than entropy
- Given shared information x (sampled from distribution X), derive shared key $k = H(x)$
  - In what sense can we claim that k is a "good" (i.e., uniformly distributed) cryptographic key?

Private-key scheme

- We have seen how to construct schemes based on various lower-level primitives
  - Stream ciphers / PRGs
  - Block ciphers / PRFs
  - Hash functions
- How do we construct these primitives?
- Construct from even lower-level assumptions
  - Can prove secure (given lower-level assumption)
  - Typically inefficient
- Build directly
  - Much more efficient!
  - Need to assume security, but
    - We have formal definitions to aim for
    - We can concentrate our analysis on these primitives
    - We can develop/analyze various design principles

Teminology

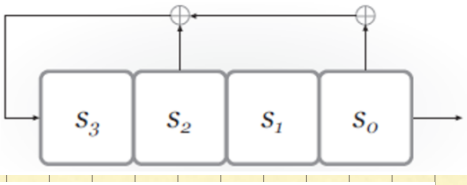
- *Init* algorithm
  - Takes as input a key + initialization vector (IV)
  - Outputs initial state
- *GetBits* algorithm
  - Takes as input the current state
  - Outputs next bit/byte/chunk and updated state
  - Allows generation of as many bits as needed

Security requirements

- If there is no IV, then (for a uniform key) the output of *GetBits* should be indistinguishable from a uniform, independent stream of bits
- If there is an IV, then (for a uniform key) the output of *GetBits* on multiple, uniform IVs should be indistinguishable from multiple uniform, independent streams of bits
  - Even if the attacker is given the IVs

LFSRs

- Degree $n \Rightarrow n$ registers
- State: bits $s_{n-1}, \ldots, s_0$ (contents of the registers)
- Feedback coefficients $c_{n-1}, \ldots, c_0$ (view as part of state; do not change)
- State updated and output generated in each "clock tick"



- Assume initial content of registers is 0100
- First 4 state transitions:
  0100 → 1010 → 0101 → 0010 → ...
- First 3 output bits:
  0 0 1 ...

LFSRs as stream ciphers

- Key + *IV* used to initialize the state of the LFSR (possibly including feedback coefficients)
- One bit of output per clock tick
  - State updated
- State (and output) "cycles" if state ever repeated
- *Maximal-length LFSR* cycles through all $2^n - 1$ nonzero states
  - Known how to set feedback coefficients so as to achieve maximal length
- *Maximal-length LFSR*s have good statistical properties, but they are not cryptographically secure!

-1是减去全0的状态，因为全0输出会一直为0.

因为是线性的，若知道反馈系数或前2n个输出，就能还原其他信息.

Security of LFSRs

- If feedback coefficients known ,the first n output bits directly reveal the initial state!
- Even if feedback coefficients are unknown, can use linear algebra to learn everything from $2n$ consecutive output bits (*Berlekamp-Massey algorithm*)

$$y_i = s_{i-1}^{(0)}, \qquad i = 1, \ldots, n$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j y_{i-n+j-1}, \qquad i > n$$

$$\begin{aligned} y_{n+1} &= c_{n-1} y_n \oplus \cdots \oplus c_0 y_1 \\ &\vdots \\ y_{2n} &= c_{n-1} y_{2n-1} \oplus \cdots \oplus c_0 y_n \end{aligned}$$

- Linearity is *bad* for cryptography (because linear algebra is so powerful)

连续2n个输出.

Nonlinear FSRs

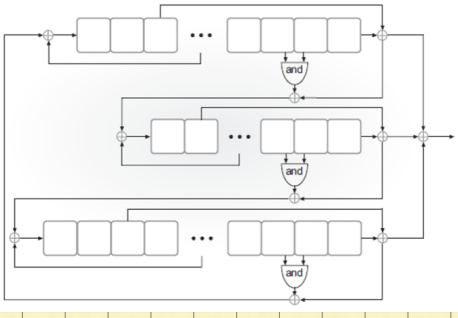
- Add *nonlinearity* to prevent attacks
  - Nonlinear feedback
  - Output is a nonlinear function of the state
  - Multiple (coupled) LFSRs
  - or any combination of the above
- Still want to preserve statistical properties of the output, and long cycle length

满足$f(x+y)=f(x)+f(y)$是线性函数.

对非线性FSR，很难说其是否安全.

Example : Trivium

- Designed by De Canniere and Preneel in 2006 as part of eSTREAM competition
- Intended to be simple and efficient (especially in hardware)
- Essentially no attacks better than brute-force search are known



这个degree是经验的

- Three FSRs of degree 93, 84, and 111
- Initialization:
  - 80-bit key in left-most registers of first FSR
  - 80-bit *IV* in left-most registers of second FSR
  - Remaining registers set to 0, except for three right-most registers of third FSR
  - Run for 4×288 clock ticks

$288 = 93+84+111$

Block ciphers

DES → 3DES → AES

- Want keyed permutation
  $$F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$$
  - n = key length, ℓ = block length

- Want F_k (for *uniform*, unknown key k) to be *indistinguishable* from a *uniform* permutation over $\{0,1\}^\ell$

The security provided by an algorithm is the most important factor. . . . Algorithms will be judged on the following factors . . .

- *The extent to which the algorithm output is indistinguishable from a random permutation . . .*

Attack models

分组密码不是加密模式

- A *block cipher* is *not* an encryption scheme !!

- Nevertheless, some of the terminology used in the same (for historical reasons)
  - "*known-plaintext attack*": attacker given $\{x, F_k(x)\}$ for random x (outside control of the attacker)
  - "*chosen-plaintext attack*": attacker can query $F_k(\cdot)$ (this is the default model we have been using)
  - "*chosen-ciphertext attack*": attacker can query $F_k(\cdot)$ and $F_k^{-1}(\cdot)$

Concrete security

- As in the case of stream ciphers, we are interested in *concrete security* for a given key length n
  - Best attack should take time $\approx 2^n$
  - If there is an attack taking time $2^{n/2}$ then the cipher is considered *insecure*

- Designing block ciphers: Want F_k (for uniform, unknown key k) to be *indistinguishable* from a uniform permutation over $\{0,1\}^\ell$. If x and x' differ in one bit, what should be the relation between $F_k(x)$ and $F_k(x')$?
  - How many bits should change (on average)?
  - Which bits should change?
  - How to achieve this?

输出的差距与输入的差距无关.

Confusion / diffusion

- "*Confusion*"
  - Small change in input should result in local, "random" change in output

- "*Diffusion*"
  - Local change in input should be propagated to entire output

混淆

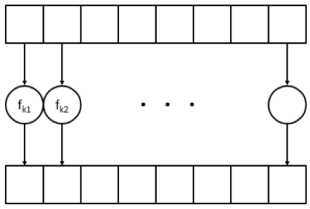
在输入上的小改变会影响局部输出

扩散

在输入上的改变会扩展到全部的输出

Design paradigms

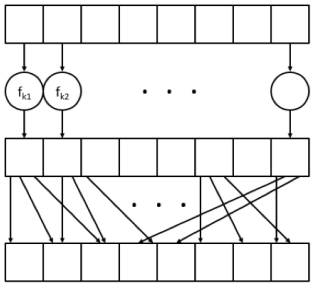
- Two design paradigms
  - *Substitution-permutation networks* (SPNs)
  - *Feistel networks*
- SPNs: build "random-looking" permutation on large input from random permutations on small inputs
  - E.g., assume 8-byte block length
    $F_k(x) = f_{k_1}(x_1) f_{k_2}(x_2) \cdots f_{k_8}(x_8)$,
    where each f is a random permutation
  - How long is k?



Is this a pseudorandom function?

SPN

- This has *confusion* but no diffusion
  - Add a *mixing permutation*



Note that the structure is *invertible* (given the key) since the f's are permutations

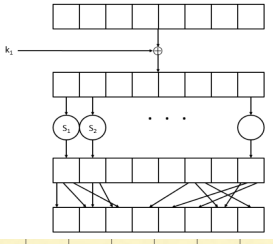
多做几轮SPN才能 diffusion

- Mixing permutation is public
  - Chosen to ensure good diffusion
- Does this give a pseudorandom function?
- What if we repeat for another round (with independent, random functions)?
  - What is the *minimal* # of rounds we need?
  - *Avalanche effect*

雪崩效应：输入改变一点，输出变化很大。

- Using random f's is not practical
  - Key would be too large
- Instead, use f's of a particular form
  - $f_{k_i}(x) = S_i(k_i \oplus x)$, where S_i is a public permutation
  - S_i are called "*S-boxes*" (*substitution boxes*)
  - XORing the key is called "*key mixing*"
  - Note that this is still invertible (given the key)

S盒是分组密码中唯一提供非线性的。



Avalance effect

- Design S-boxes and mixing permutation to ensure *avalanche effect*
  - Small differences should eventually propagate to entire output
- S-boxes: 1-bit change in input causes $\geq$ 2-bit change in output
  - Not so easy to ensure!
- Mixing permutation
  - Each bit output from a given S-box should feed into a *different* S-box in the next round

SPN

- One round of an SPN involves
  - Key mixing
    - Ideally, round keys are independent
    - In practice, derived from a master key via *key schedule*
  - Substitution (S-boxes)
  - Permutation (mixing permutation)
- r-round SPN has r rounds as above, plus a final key-mixing step
  - Why?
- *Invertible* regardless of how many rounds

Key-recovery attacks

- *Key-recovery attacks* are even more damaging than distinguishing attacks
  - As before, a cipher is *secure* only if the best key-recovery attack takes time $\approx 2^n$
  - A fast key-recovery attack represents a "complete attack" of the cipher

Key-recovery attack, 1-round SPN

- Consider first the case where there is no final key-mixing step
  - Possible to get the key immediately!
- What about a full 1-round SPN?
  - Attack 1: for each possible 1^{st}-round key, get corresponding 2^{nd}-round key
  - Continue process of elimination
  - Complexity $\approx 2^\ell$ for key of length 2ℓ
- Better attack: work S-box-by-S-box
  - Assume 8-bit S-box
  - For each 8 bits of 1^{st}-round key, get corresponding 8 bits of 2^{nd}-round key
  - Continue process of elimination
  - Complexity?

Lindell & Katz p.209-211

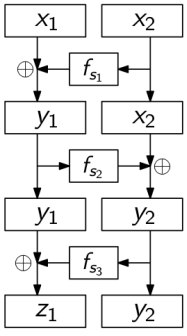
Feistel networks

通过不可逆的组件构造出可逆排列

- Build (invertible) permutation from non-invertible components
- One round:
  - Keyed round function f: $\{0,1\}^n \times \{0,1\}^{\ell/2} \to \{0,1\}^{\ell/2}$
  - $F_{k_1}(L0, R0) \to (L1, R1)$
    where $L1 = R0$; $R1 = L0 \oplus f_{k_1}(R0)$
- Always invertible!

Luby-Rackoff construction

- This is so-called *Luby-Rackoff construction*, using several rounds of *Feistel Transformation*.



We build a PRP p on $2n$ bits from three PRFs $f_{s_1}, f_{s_2}, f_{s_3}$ on n bits by letting

$p_{s_1,s_2,s_3}(x_1, x_2) = (z_1, y_2)$

where $y_1 = x_1 \oplus f_{s_1}(x_2)$,
$y_2 = x_2 \oplus f_{s_2}(y_1)$, and
$z_1 = f_{s_3}(y_2) \oplus y_1$.

Security

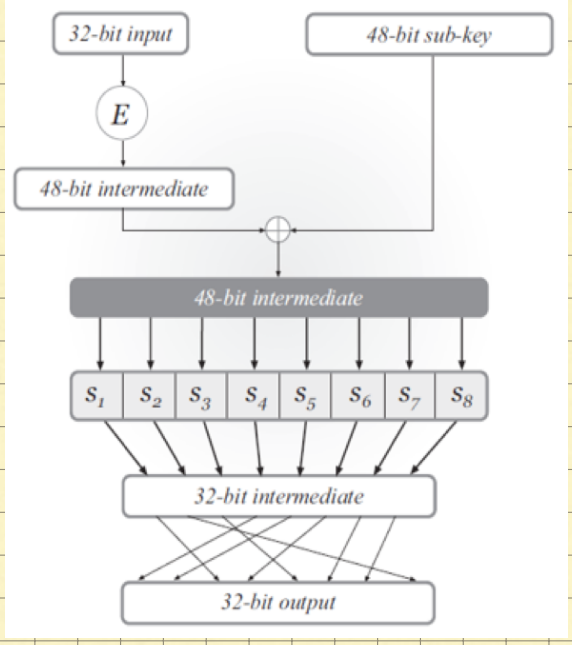
- Security of 1-round Feistel?
- Security of 2-round Feistel (with independent keys)?
- Security of 3/4-round Feistel?

Lindell & Katz p.216-218

Data Encryption Standard (DES)

- Standardized in 1977
- 56-bit keys, 64-bit block length
- 16-round Feistel network
  - Same round function in all rounds (but different sub-keys)
  - Basically an SPN design!

DES mangler function



- S-boxes
  - Each S-box is 4-to-1
  - Changing 1 bit of input changes at least 2-bits of output
- Mixing permutation
  - The 4 bits of output from any S-box affect the input to 6 S-boxes in the next round

Key schedule + Avalanche effect

- 56-bit master key, 48-bit subkey in each round
  - Each subkey takes 24 bits from the left half of the master key, and 24 bits from the right half of the master key
- Consider 1-bit difference in left half of input
  - After 1 round, 1-bit difference in right half
  - S-boxes cause a 2-bit difference, implying a 3-bit difference overall after 2 rounds
  - Mixing permutation spreads differences into different S-boxes

Security of DES

- DES is extremely well-designed
  - Except for some attacks that require large amounts of plaintext, no attacks better than brute-force are known
- But, parameters are too small!
  - I.e., brute-force search is feasible

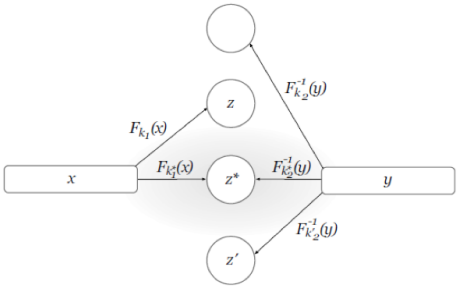
DES的破解方法中暴力法是最优的

Double encryption

- Let $F : \{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$
  - i.e., $n = 56$, $\ell = 64$ for DES
- Define $F^2 : \{0,1\}^{2n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as follows:
  $$F^2_{k_1,k_2}(x) = F_{k_1}(F_{k_2}(x))$$
  (still invertible)
- If best attack on F takes time 2^n, is it reasonable to assume that the best attack on F^2 takes time 2^{2n}?

Meet-in-the-middle attack

- No! There is an attack taking 2^n time
  - And 2^n memory



- The attack applies any time a block cipher can be "factored" into 2 independent components

两边并行计算

Triple encryption

- Define $F^3 : \{0,1\}^{3n} \times \{0,1\}^\ell \to \{0,1\}^\ell$ as follows:
  $$F^3_{k_1,k_2,k_3}(x) = F_{k_1}(F_{k_2}(F_{k_3}(x)))$$
- What is the best attack now?
- Best attacks take time 2^{2n} – optimal given the key length!
- This approach is taken by triple-DES

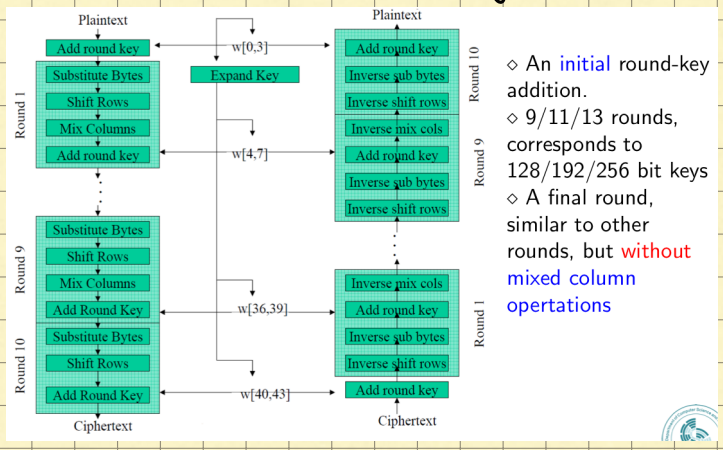
Advanced Encryption Standard (AES)

- 128-bit block length
- 128-, 192-, and 256-bit key lengths
- Basically an SPN structure!
  - 1-byte S-box (same for all bytes)
  - Mixing permutation replaced by invertible linear transformation
- No attacks better than brute-force known

Rijndael: Key and Block Size

Key Size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

AES Encryption & Decryption



- An initial round-key addition.
- 9/11/13 rounds, corresponds to 128/192/256 bit keys
- A final round, similar to other rounds, but without mixed column opertations

AES Round Function