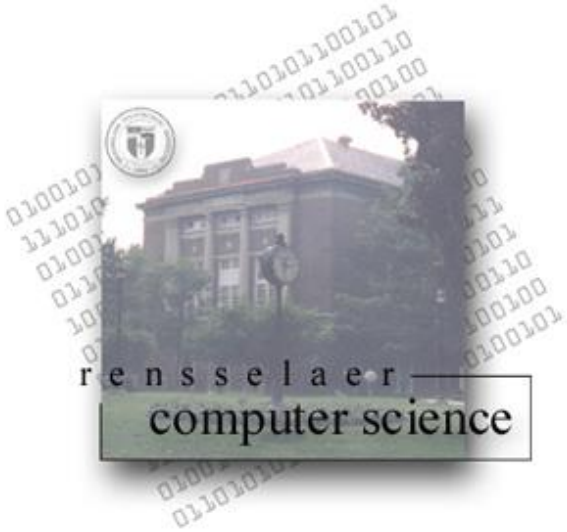


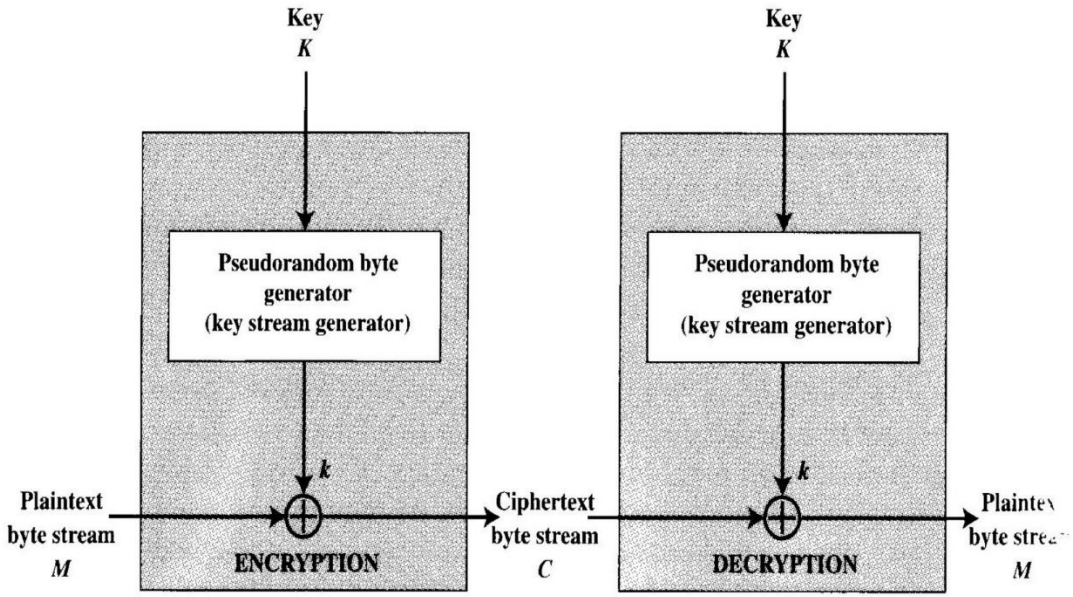
Cryptography &Network Security I

Bülent Yener
yener@cs.rpi.edu

Lecture 3.2

Last time: RC4 Stream Cipher

```
  11001100  plaintext
⊕ 01101100  key stream
  10100000  ciphertext
```



Today: Randomness, PRG and PRF

Randomness

Randomness is a property of a distribution not a string

The distribution of X is uniform (aka random) over V (e.g., $V=\{0,1\}^n$):
For each $v \in V$: $Pr[X=v]=1/|V|$ (e.g., 2^{-n})

Similarly, being “almost random” is a property of a distribution not a string

Random Numbers- In practice

- In practice applications use pseudo-random numbers, must ensure cannot predict future elements of sequence on basis of earlier elements.
- Consider key generators. We want to have generators that produces evenly ***balanced*** outputs (0's and 1's) that has ***no memory***.
- Balanced Binary Bernoulli sources
- Sources
  - Random: Output is not deterministic
  - Pseudorandom: Output is deterministic. Appears to be random to observers who do not know underlying determinism.

Random Numbers

– Consider following sequence of numbers. Is this a random sequence? Can you tell what the x is?

926535897932384626433832795028841971693993751**X**

Random Numbers

– Now?

3.1415926535897932384626433832795028841971693993751**X**

– Yes, it is π, and x = 0.
– We fail to initially perceive the **underlying determinism**.

Random Numbers

– What is x?

  OTTFFSSEN**x**

Random Numbers

- What is x?

  One**T**wo**T**hree**F**our**F**ive**S**ix**S**even**E**ight**N**ine**X**

- X=T, right?

Random Numbers

- Consider a deck of card. 52 distinct cards can be arranged in 52! ≈ 10^{67} different ways.
- Now, consider riffle shuffle of a deck of cards.
  - Cut the deck into two from randomly selected point.
  - Meld the cards.
- Lets see how many possible arrangement in this case?
  - Assume that deck is broken into two sets, Right and Left sets, where each set has **A** and **52-A** cards.
  - Melding process simply selects $\mathbf{l_i}$ cards from Left set and $\mathbf{r_i}$ cards from Right set to form following sequence of 52 cards:
  - $L^{l1}R^{r1}L^{l2}R^{r2}L^{l3}R^{r3}$….. Where $l_i \geq 0$ and $r_i \geq 0$

Random Numbers

- $L^{l1}R^{r1}L^{l2}R^{r2}L^{l3}R^{r3}$….. Where li ≥0, ri ≥ 0 and

$$\sum_i l_i = A \qquad \sum_i r_i = 52 - A$$

- The number of such sequences is:

$$\binom{52}{A}$$

- And finally, the number of such shuffles are

$$\sum_{A=0}^{52} \binom{52}{A} = 2^{52} \approx 10^{15.6}$$

Random Numbers

- $10^{15.6}$ compared to 10^{67}.
- No hope to perform random rearrangement.
- Need to shuffle at least n=5 times, $(10^{15.6})^n \geq 10^{52}$

Randomness is hard to get and bad implementations become vulnerability !

An Example
[*Ian Goldberg and David Wagner 1996]*

- Many common computer applications (games, for instance) use any readily available source of randomness to provide an initial value, called a "seed" to a pseudorandom number generator (PRNG). PRNGs operate by repeatedly scrambling the seed. Typically, the seed is a short, random number that the PRNG expands into a longer, random-looking bitstream. A typical game might seed a PRNG with the time of day;
- *A typical C program that uses a PRNG. A typical C program that uses a PRNG.*

```
srand(time(0));
...
printf("You rolled the die, and got a %d.\n", 1 + (rand()%6));
```

- In cryptographic applications, however, the seed's unpredictability is essential-if the attacker can narrow down the set of possible seeds, his job is made significantly easier.
  - Since the function used by the PRNG to turn a seed into a pseudorandom number sequence is assumed to be known, a smaller set of possible seeds yields a correspondingly small set of sequences produced by the PRNG.
- A good method to select seed values for the PRNG is an essential part of a cryptographic system such as SSL. If the seed values for the PRNG can easily be guessed, the level of security offered by the program is diminished significantly, since it requires less work for an attacker to decrypt an intercepted message.

Weaknesses

Unfortunately for Netscape, U.S. regulations prohibit the export of products incorporating strong cryptography. In order to distribute an international version of its browser overseas, Netscape had to weaken the encryption scheme to use keys of just 40 bits, leaving only a million million possible key values.

That may sound like a lot of numbers to try, but several people (David Byers, Eric Young, Damien Doligez, Piete Brooks, Andrew Roos, Adam Back, Andy Brown and many others) have been able to try every possible key and recover SSL- encrypted data in as few as 30 hours using spare CPU cycles from many machines.

Since nearly all Netscape browsers in use are the free international version, the success of this attack demonstrates a fundamental vulnerability in Netscape that cannot be repaired under current export regulations.

Example: Netscape

- Reverse engineering Netscape's SSL algorithm by manually decompiling their executable program.

- Serious flaws in Netscape's implementation of SSL that make it relatively easy for an eavesdropper to decode the encrypted communications. (Netscape 2.0 beta1 and Netscape Navigator 1.22 on is fixed)

The Netscape 1.1 seeding process:

- global variable seed;
- RNG_CreateContext()
- (seconds, microseconds) = time of day; /* Time elapsed since 1970 */
- pid = process ID; ppid = parent process ID;
- a = mklcpr(microseconds);
- b = mklcpr(pid + seconds + (ppid << 12));
- seed = MD5(a, b);
- mklcpr(x) /* not cryptographically significant; shown for completeness

*/

- return ((0xDEECE66D * x + 0x2BBB62DC) >> 1);

MD5() /* a very good standard mixing function, source omitted */

The Netscape v1.1 key-generation process

Note that each possibility for the time of day, process ID, and parent-process ID produces a unique seed, which in turn produces a unique encryption key.

1.x = MD5(seed);

2.seed = seed + 1;

3.return x;

global variable challenge, secret_key;

1.create_key()

2.challenge = RNG_GenerateRandomBytes();

3.secret_key = RNG_GenerateRandomBytes();

Attack I

- When a connection is first established, a challenge value is calculated and sent *unencrypted from the Netscape client* to the secure server.
  - This allows an attacker to learn that value, which will be useful later.

- Assume: An attacker who has an account on the UNIX machine running the Netscape browser
  - can easily discover the pid and ppid values used in using the ps command

- All that remains is to guess the time of day.
  - Most popular Ethernet sniffing tools (including tcpdump) record the precise time they see each packet.
  - Using the output from such a program, the attacker can guess the time of day on the system running the Netscape browser to within a **second.**

Attack I cont.

- We know the values of seconds, pid, and ppid; only the value of the microseconds variable remains unknown. However, there are only one million possible values for it, resulting in only one million possible choices for the seed.

- We can use the algorithm to generate the challenge and secret_key variables for each possible seed. Comparing the computed challenge values to the one we intercepted will reveal the correct value of the secret key.

- Testing all one million possibilities takes about 25 seconds on an HP 712/80.

Attack II

- Assume: the attacker does not have an account on the attacked UNIX machine
  - the pid and ppid quantities are no longer known.
  - Nonetheless, these quantities are rather predictable, and several tricks can be used to recover them.

- “The unknown quantities are mixed in a way which can cancel out some of the randomness.”

Attack II cont.

- Even though the pid and ppid are 15bit quantities on most UNIX machines, the sum pid + (ppid << 12) has only 27 bits (not 30bits)

- If the value of seconds is known, a has only 20 unknown bits, and b has only 27 unknown bits.
  - (reduced entropy)

- This leaves, at most, 47 bits of randomness in the secret key-a far cry from the 128-bit security claimed by the domestic U.S. version.

Some more tricks

- Some programs will leak information about the parameters of interest.
- For example, the popular mail-transport agent sendmail generates Message-IDs for outgoing mail using its process ID;
  - as a result, sending e-mail to an invalid user on the attacked machine will cause the message to bounce back to the sender;
  - the Message-ID contained in the reply message will tell the sender the last process ID used on that machine.
- Assuming that the user started Netscape relatively recently, and that the machine is not heavily loaded, this will closely approximate Netscape's pid.

Randomness, PRG and PRF
theoretical foundations

Randomness is a property of a distribution not a string

The distribution of X is uniform (aka random) over V (e.g., $V=\{0,1\}^n$):
For each $v \in V$: $Pr[X=v]=1/|V|$ (e.g., 2^{-n})

Similarly, being “almost random” is a property of a distribution not a string

What does the distribution of Y is almost uniform mean ???

Basics

(Discrete) Probability: A function *Pr* : Ω="all outcomes of an experiment" → [0,1].
Ω is called the *sample space.*
Example: In the experiment of throwing a dice: $\Omega = \{1,\ldots,6\}$

Event: A particular occurrence in some experiment $(E \subseteq \Omega)$
Example: $E = \{1,3,5\}$ or *"the outcome is odd"*

Basic properties:

$$\sum_{\omega \in \Omega} \Pr[\omega] = 1$$

$$\text{for all events } E: \ 0 \leq \Pr[E] \leq 1$$

Basics

Conditional Probability: The probability that an event occurs given that another event has occurred.

Example: In throwing a dice:

Pr[“The outcome is 2” I “the outcome is even”]

Baye’s Theorem:

$$\Pr[E_1|\ E_2] = \frac{\Pr[E_1 \cap E_2]}{\Pr[E_2]}$$

Example (cont’d). E_1=“The outcome is 2” and E_2=“the outcome is even”

$$\Pr[E_1|\ E_2] = \frac{\Pr[E_1 \cap E_2]}{\Pr[E_2]} = \frac{\frac{1}{6}}{\frac{1}{2}} = \frac{1}{3}$$

Basics

Total Probability Theorem: Let $B_1,\ldots,B_n$ be mutually exclusive (aka pairwise disjoint) events such that $\Pr[B_1] + \ldots + \Pr[B_n] = 1$.

Then $\Pr[A] = \Pr[A|B_1]\Pr[B_1] + \ldots + \Pr[A|B_n]\Pr[B_n]$

Often used as: $\Pr[A] = \Pr[A|B]\Pr[B] + \Pr[A|\bar{B}]\Pr[\bar{B}]$

$$\Pr[A \cup B] = \Pr[A] + \Pr[B] - \Pr[A \cap B]$$

The union bound:

$$\Pr[A_1 \cup \ldots \cup A_n] \leq \sum_{i=1,\ldots,n} \Pr[A_1] + \ldots \Pr[A_n]$$

Basics

Random variable (informal): A variable X that takes on (usually discrete) values with some probability.

A "standard" event is that a random variable takes on a given value,
Example. E: $\mathrm{X} = \mathrm{x}$

Probability Distribution: Assigns a probability to each outcome of the random variable.

Independent r.v.:

$$\Pr[X = x|Y = y] = \Pr[X = x]$$

Revisiting Security of CDH

- Computational Diffie-Hellman (CDH) by itself is not sufficient to prove that the DH protocol is useful for practical cryptographic purposes.
- Even though Eve may be unable to recover the entire secret, she may still be able to recover valuable information about it.
- For instance, even if CDH is true, Eve may still be able to predict %80 of the bits of g^{ab} with some confidence.
- Current state of art: we are unable to prove that no such attack exists.

Decision Diffie-Hellman Assumption (DDH)

The DDH assumption states that no efficient algorithm can distinguish between the two distributions

$<g^a ; g^b ; g^{ab}>$

and

$<g^a ; g^b ; g^c>$

where a; b; c are chosen at random in [1; G]

Polynomial and Negligible

Usually in cryptography:

- Efficient=Polynomial (usually in the key size n)

There exists a polynomial p(.) such that the adversary does at most p(n) “computations”

- Why?

Because polynomial has nice composability properties:

- polynomial * polynomial = polynomial
- polynomial + polynomial = polynomial

Polynomial and Negligible

• Tiny=Negligible (in n)

A function $f: \mathbb{N} \rightarrow \mathbb{R}+$ is **negligible** if for every integer $c>0$ there exists an integer $n_c > 0$ such that for every $n>n_c$: $f(n)<1/n^c$

• Why?

Because (1) it is tiny (2) it has composability properties and (3) it is convenient (it works nice with polynomial ;))

• negligible * negligible = negligible
• negligible + negligible = negligible
• negligible * polynomial = negligible
• negligible + polynomial = polynomial

A ‘’classic” negligible function: 2^{-n}

Statistical Distance

X, Y: random variables that take values from V
The **statistical distance** of X and Y is defined as follows:

$$\Delta[X,Y] = \frac{1}{2} \sum_{v \in V} | \Pr[X = v] - \Pr[Y = v] |$$

- We say that X and Y are **ε-close** if $\Delta[X,Y] \leq \varepsilon$

- We say that X and Y with values from $\{0,1\}^n$ are **statistically indistinguishable** if they are $f(n)$-close for a negligible function $f(n)$

Almost uniformly random: The distribution of K is statistically close the uniform distribution

Properties of Statistical Distance

$$\Delta[X,Y] = \frac{1}{2}\sum_{v \in V} |\Pr[X=v] - \Pr[Y=v]|$$

$\Delta(X,Y) \geq 0$

$\Delta(X,X) = 0$ (but $\Delta(X,Y) = 0 \nRightarrow X=Y$)

$\Delta(X,Y) = \Delta(Y,X)$

$\Delta(X,Y) \leq \Delta(X,Z) + \Delta(Z,Y)$

Properties of $\Delta(X,Y)$

$$V_{P_X \geq P_Y} = \{v \in V \ : \ \Pr(X = v) \geq \Pr(Y = v)\}$$
$$V_{P_X < P_Y} = \{v \in V \ : \ \Pr(X = v) < \Pr(Y = v)\}$$

$$\Delta(X,Y) = \Pr[x \in V_{P_X \geq P_Y}] - \Pr[y \in V_{P_X \geq P_Y}]$$
$$\Delta(X,Y) = \Pr[y \in V_{P_X < P_Y}] - \Pr[x \in V_{P_X < P_Y}]$$

$\Delta(X,Y)$ as Advantage of Best Distinguisher

An equivalent formulation:

B: An algorithm with inputs from V and output a single bit

X: A r.v. with values from V

Then

- $B(X)$ is a random variable with values from {0,1}

$\Delta(X,Y)$ as Advantage of Best Distinguisher

For two random variables X and Y taking values from V:

$$\Delta[X,Y] = \max_B \{|\Pr[B(X)=1] - \Pr[B(Y)=1]|\}$$

$$\Delta[X, Y] = \text{negl.} \iff \forall\, B : \Delta^B[X, Y] = \text{negl.}$$

Perfect indistinguishability

Consider the following experiment (game) for an encryption scheme $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ involving an adversary and a challenger:

- Adversary choses plaintexts m_0 and m_1
- Challenger chooses random key $k \leftarrow \mathrm{KeyGen}$ and bit b
- Challenger computes $\mathrm{Enc}_k(m_b)$ and gives it to the adversary
- Adversary tries to guess b, i.e., outputs b' that is his guess for b.
- Adversary wins if he guesses correctly, i.e., $b'=b$

An encryption scheme is **perfectly indistinguishable** if the adversary cannot guess b

… with probability better than 1/2

$$\Pr[\mathrm{PerfExp}_{\mathrm{Adv}}=1]=1/2$$

Example- OTP

Claim. The one-time pad is perfectly indistinguishable

- Single use keys
- Fixed length
- Key as long as the message
- perfectly random keys

Inherent Limitations of Perfect Secrecy also in perfect indistinguishability

ε-indistinguishable

An encryption scheme is **ε-indistinguishable**
If For all Adv:

$\Pr[\mathrm{PerfExp}_{\mathrm{Adv}}=1] \leq 1/2 + \varepsilon$

Every perfectly indistinguishable encryption scheme is ε-indistinguishable for every $\varepsilon>0$.

Relaxations

1. Allow the adversary to win with a tiny probability
2. Restrict the adversary's assumed computing power

Concrete Security (informal): An encryption scheme is ε-secure for $0<\varepsilon<1$ if and only if any attacker (*adversary*) cannot break the encryption with probability better than ε.

Concrete vs Asymptotic Security

Concrete Security (informal): An encryption scheme is (ε,t)-secure for $0<\varepsilon<1$ if and only if any attacker (*adversary*) **who runs in time t** cannot break the encryption with probability better than ε.

Asymptotic Security (informal): An encryption scheme is secure if and only if for an appropriate choice of parameters (e.g., key length) any **efficient** attacker (*adversary*), cannot break the encryption except with **tiny** probability.

What is a good ε?

Say security fails with probability 2^{-60}
–Should we be concerned about this?

–With probability > 2^{-60}, the sender and receiver will both be struck by lightning in the next year… ☺

Something that occurs with probability 2^{-60}/sec is expected to occur once every 100 billion years

Thus if we use a key of size $n>60$ bits then $\varepsilon=2^{-n}$ is small enough

What is a good t?

- Consider brute-force search of key space; assume one key can be tested per clock cycle
- Desktop computer 2^{57} keys/year
- Supercomputer 2^{80} keys/year
- Supercomputer since Big Bang 2^{112} keys

–Restricting attention to attackers who can try 2^{112} keys is reasonable…

- Modern key space: 2^{128} keys or more…

So how do we obtain pseudo randomness?

Pseudo-randomness

- Fix some distribution P_X on *n*-bit strings
  - Notation: $x \leftarrow P_X$ means "sample x according to P_X"

- Historically, D was considered pseudorandom if it "passed a bunch of statistical tests"
  - $\Pr_{x \leftarrow P_x}[1^{st} \text{ bit of } x \text{ is } 1] \approx ½$
  - $\Pr_{x \leftarrow P_x}[\text{parity of } x \text{ is } 1] \approx ½$
  - $\Pr_{x \leftarrow P_x}[A_i(x)=1] \approx \Pr_{x \leftarrow U_n}[A_i(x)=1], \text{ for } i = 1, \ldots, n$

- This is not sufficient in an adversarial setting!
  - if adversary knows what statistical test T is in use?

- Cryptographic definition of pseudo-randomness:
  - P_X is pseudorandom if it passes all *efficient* statistical tests

Pseudo-randomness

Concrete

- Let P_X be a distribution on *n*-bit strings.
- $D=P_X$ is (t, ε)-pseudorandom if for all A running in time ≤ t,

$$| \Pr_{x \leftarrow D}[A(x)=1] - \Pr_{x \leftarrow U_n}[A(x)=1] | \leq \varepsilon$$

A(x)=1 : "A believes x←D
A(x)=0 : "A believes x←U_n

Asymptotic

- Security parameter *n*, polynomial *p*
- Let D_n be a distribution over p(n)-bit strings
- Pseudorandomness is a property of a *sequence* of distributions

$\{D_n\}n\in\mathbb{N} = \{D_1, D_2, \dots\}$

- $\{D_n\}n\in\mathbb{N}$ is pseudorandom if for every probabilistic, polynomialtime A, there is a negligible function μ such that

$$| \Pr_{x \leftarrow D_n}[A(x)=1] - \Pr_{x \leftarrow U_{p(n)}}[A(x)=1] | \leq \mu(n)$$

From statistical to computational indistinguishability

Recall: Statistical indistinguishability requires that every algorithm (distinguisher) has an at most negligible probability to distinguish between the two.

$$\Delta[X,Y] = \max_B\{|Pr[B(X)=1 - Pr[B(Y)=1]|\} \leq \mu$$

for some negligible function μ

Computational indistinguishability: Every **efficient** algorithm (distinguisher) has an at most negligible probability to distinguish between the two.

For two random variables X and Y taking values from V:

$$D[X,Y] = \max_{B \in PPT}\{|Pr[B(X,1^n)=1 - Pr[B(Y,1^n)=1]|\} \leq \mu$$

Pseudorandom Number Generators (PRNGs)

- A PRG is an efficient, deterministic algorithm that expands a *short, uniform seed* into a *longer, pseudorandom* output
  - Useful whenever you have a “small” number of true random bits, and want lots of “random-looking” bits
- i.e., it is an algorithmic technique to create “random numbers”
  - although not truly random
  - can pass many tests of “randomness”

Pseudo-random (number) Generator (PRG)

Let G be a deterministic, poly-time algorithm

G is *expanding*: $|G(x)| = p(|x|) > |x|$

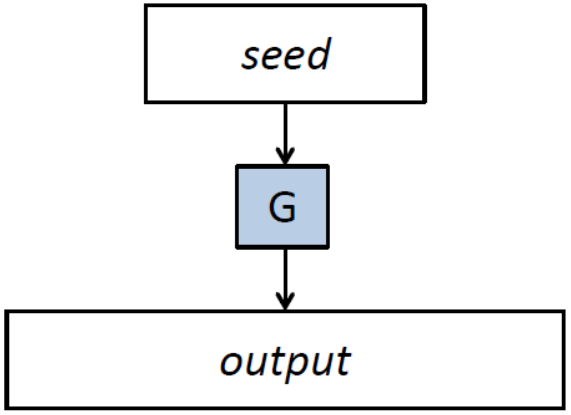
G defines a sequence of distributions!

- D_n the distribution on p(n)-bit strings defined by choosing $x \leftarrow U_n$ and outputting $G(x)$
- $D_n = G(U_n) = G(X)$, where $X \leftarrow P_X = U_n$

$$P_{D_n}(y) = P_{x \leftarrow U_n}(G(x) = y) = \Sigma_{x\,:\,G(x)=y}\, P_{U_n}(x)$$

$$= \Sigma_{x\,:\,G(x)=y}\, 2^{-n}$$

$$= |\{x : G(x)=y\}|/2^n$$



Pseudo-random (number) Generator (PRG)

G is a *pseudorandom generator* if $\{D_n\}$ is pseudorandom

- I.e., for all PPT attackers (distinguishers) A, there is a negligible function μ such that

$$| \Pr_{x \leftarrow U_n}[A(G(x))=1] - \Pr_{y \leftarrow U_{p(n)}}[A(y)=1] | \leq \mu(n)$$

- No efficient A can distinguish whether it is given $G(x)$ (for uniform x) or a uniform string y!

Pseudo-random Functions (PRF)

Informally, a pseudorandom function "looks like" a random function (to a PPT distinguisher)

Random Function

$Func_n$ = all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$
Can represent a function in $Func_n$ using $n \cdot 2^n$ bits
$\Rightarrow |Func_n| = 2^{n \cdot 2^n}$

000	010
001	100
010	100
011	111
100	001
101	010
110	010
111	000

Size $2^3 = 8$

(Uniform function)

- Choose uniform $f \in Func_n$
- Equivalently, for each $x \in \{0,1\}^n$, choose f(x) uniformly in $\{0,1\}^n$
  - I.e., fill up the function table with uniform values
  - Can also think of this being done "on-the-fly"

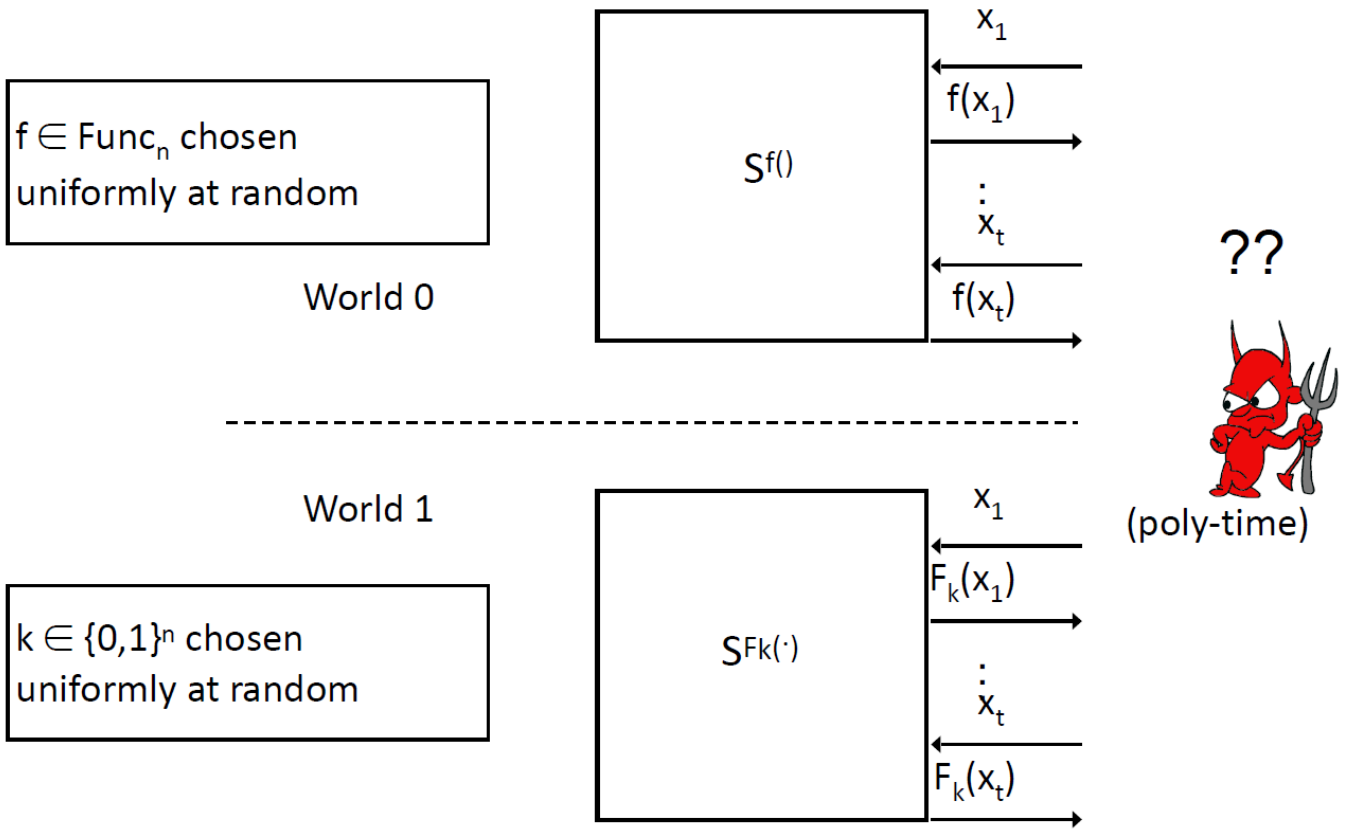
We look at *keyed* functions

PRF

F is a *pseudorandom function* if F_k, for uniform key $k \in \{0,1\}^n$, is indistinguishable from a uniform function $f \in \text{Func}_n$

Formally, for all poly-time D:

$$\Delta^D(S^{F_k(\cdot)}, S^{f(\cdot)}) = | \Pr_{k \leftarrow \{0,1\}^n}[D(S^{F_k(\cdot)}) = 1] - \Pr_{f \leftarrow \text{Func}_n}[D(S^{f(\cdot)}) = 1] | \leq \mu(n)$$

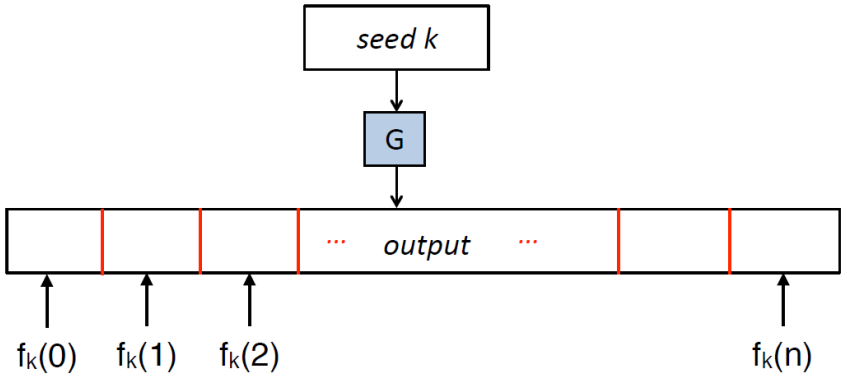


PRFs vs PRGs

PRF F immediately implies a PRG G: $G(k) = F_k(0) \mid F_k(1) \mid F_k(2) \mid \ldots$

PRF from PRGs

A simplistic approach: PRF can be viewed as a PRG with random access to exponentially long output



The Goldreich, Goldwasser, Micali (GGM) construction:

- Let G be a PRG: $\{0,1\}^k \rightarrow \{0,1\}^{2k}$
- Define G_0 the function $\{0,1\}^k \rightarrow \{0,1\}^k$ that on input $x \in \{0,1\}^k$ outputs the first half of G(x)
- Define G_1 the function $\{0,1\}^k \rightarrow \{0,1\}^k$ that on input $x \in \{0,1\}^k$ outputs the second half of G(x)

Do PRGs exists?

• We don't know…
• Can *assume* certain G's are PRGs
–This is what is often done in practice: *Fortuna*, *Yarrow* (/dev/random), …
•Based on assumptions:
• *Blum Blum Shub:* a PRG if the Quadratic Residuosity (QR) problem is hard
• "Blum Micali": PRG if DL is secure
•*Dual-EC-DRBG*: PRG if DDH is hard on elliptic curve groups.
• AES-CTR

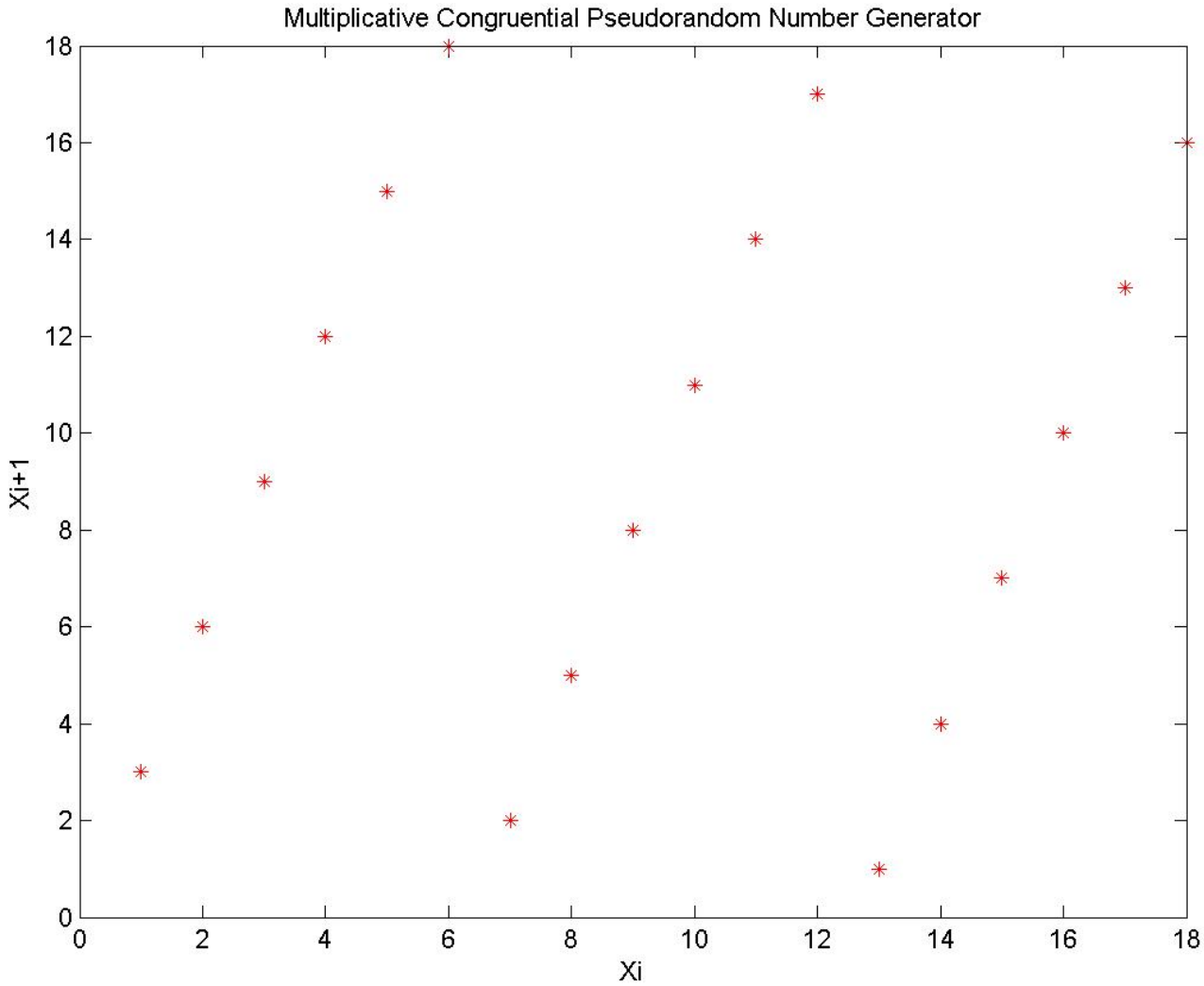
Pseudorandom Numbers in Practice

- We want balanced sources which are memoryless.
- Consider ***Multiplicative Congruential Pseudorandom Generator (MCPG)***.

$$\mathbf{x_{i+1} = 3 \,.\, x_i \bmod 19}$$

- Let $x_0 = 1$, then the generated sequence is:
- 1 3 9 8 5 15 7 2 6 18 16 10 11 14 4 12 17 13 1
- Consider the following graph, x_i versus x_{i+1}. Although sequence seems random, there is an order!

Pseudorandom Numbers (MCPG)



Pseudorandom Numbers (MCPG)

- Consider following output of a binary source

  01 11 01 11 00 00 10 10 00 01 11 10

- There are 12 one and 12 zeros. Source seems to be balanced.
- Consider di-bit patterns:

3	00
3	01
3	10
3	11

Pseudorandom Numbers (MCPG)

011 101 110 000 101 000 011 110

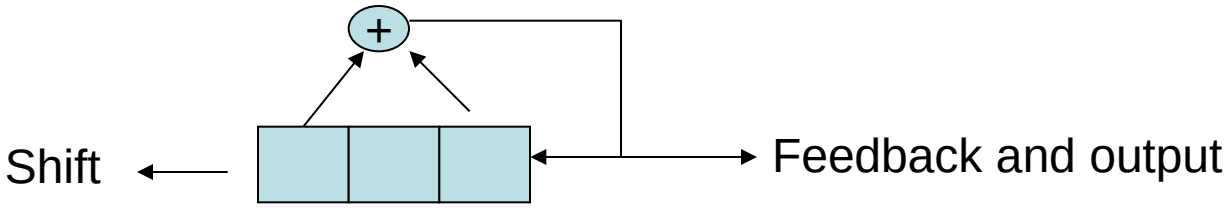
- What about tri-bit patterns:

2	000
0	001
0	010
2	011
0	100
2	101
2	110
0	111

- As we go higher dimensional analysis, we got striking results!
- **How much analysis do we need in order to safely conclude that generator is random?**
  - **Memory and processing time limit?**

M-Sequences

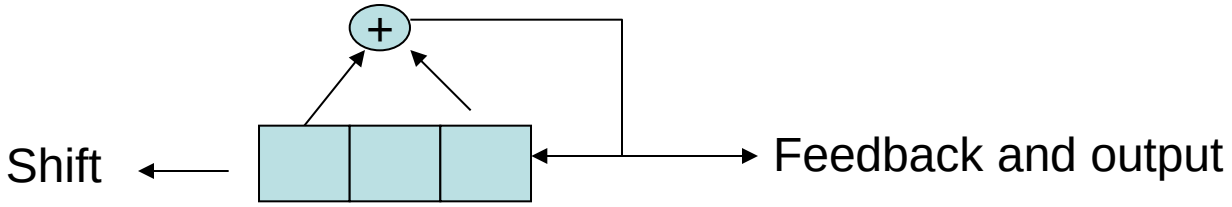
- Finite state sequential machines
- Can produce long cycles
- Remarkable statistical properties (distribution of **k-tuple** patterns we have seen before)
- Easy and fast to implement



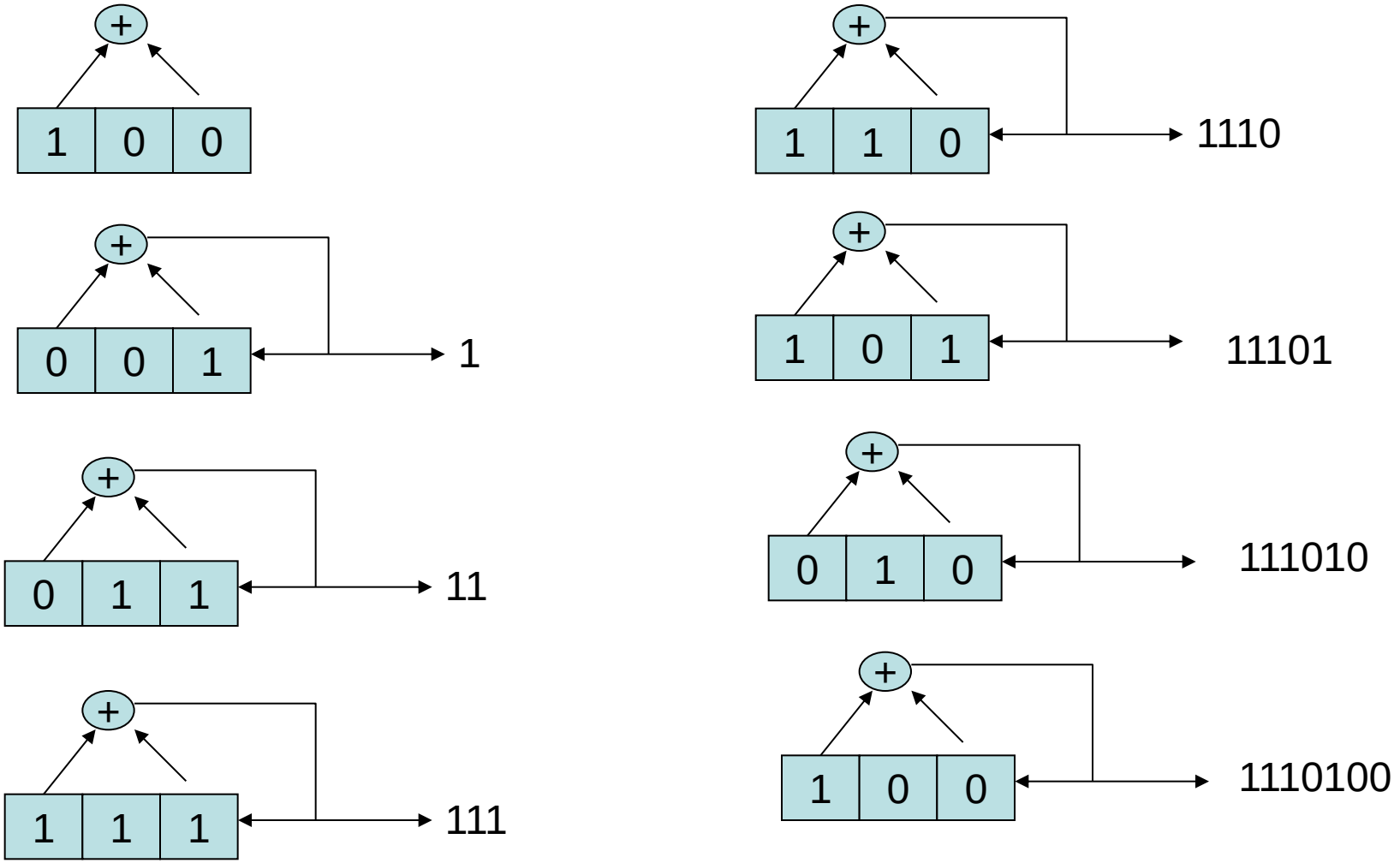
Primitive Polynomial : x^3+x+1

M-Sequences

- Primitive polynomial x^3+x+1
- Means has 3 bits,
- Bit positions 1 and 3 will be XORed and will be fed back
- Cycle length is $2^n-1 = 2^3-1= 7$. Meaning after 7 step, it will return initial state and start generating same sequence.



M-Sequences



Primitive Polynomial : x^3+x+1

M-Sequences

- Some primitive polynomial

  x^2+x+1

  x^3+x+1

  x^4+x+1

  x^5+x^2+1

  …..

  $x^{18}+x^7+1$

  $x^{21}+x^2+1$ → cycle length $2^{21}-1$

Linear Congruential Generator

- common iterative technique using:

  $X_{n+1} = (aX_n + c) \bmod m$

- given suitable values of parameters (a, c, m, x_0) can produce a long random-like sequence. Basically cycle length is m/d where:
  - GCD (a, m) = 1
  - d = GCD(m, x_0 (a-1) + c)
- ANSI C rand() function ($m=2^{31}$, a=1103515245, c=12345, x_0=12345)
- suitable criteria to have are:
  - function generates a full-period
  - generated sequence should appear random
  - efficient implementation with 32-bit arithmetic
- note that an attacker can reconstruct sequence given a small number of values

Using Block Ciphers as Stream Ciphers

- can use block cipher to generate numbers
- use Counter Mode

  $X_i = \mathrm{E}_{Km}[\mathrm{i}]$

- use Output Feedback Mode

  $X_i = \mathrm{E}_{Km}[X_{i-1}]$

- ANSI X9.17 PRNG
  - uses date-time + seed inputs and 3 triple-DES encryptions to generate new seed & random

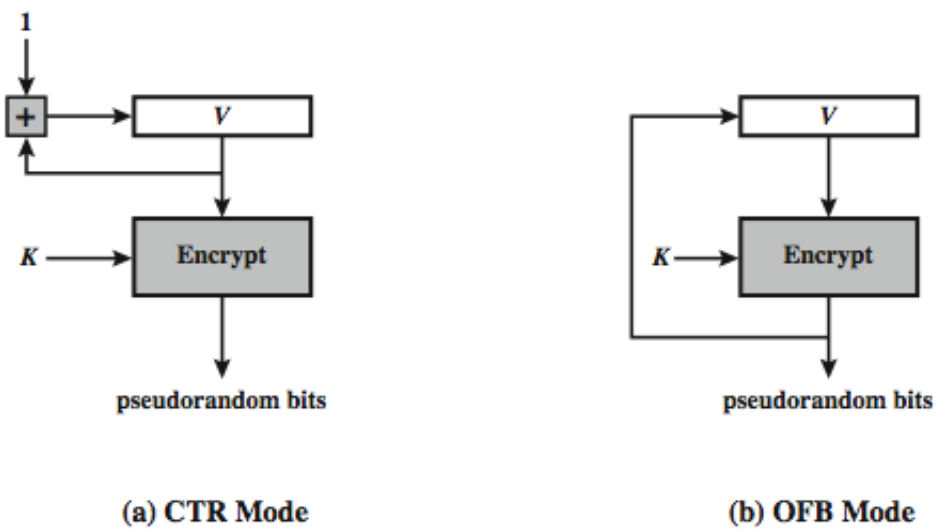
Using Block Ciphers as PRNGs

- For any block of plaintext, a symmetric block cipher produces an output block that is apparently random → no patterns or regularities in the ciphertext that provide information that can be used to deduce the plaintext.
- Application: session keys from master key
- CTR (ANSI x9.82 and RFC4036)

  $X_i = E_K[V_i]$

  V is incremented by 1 after each encryption
- OFB (X9.82 and RFC 4086)

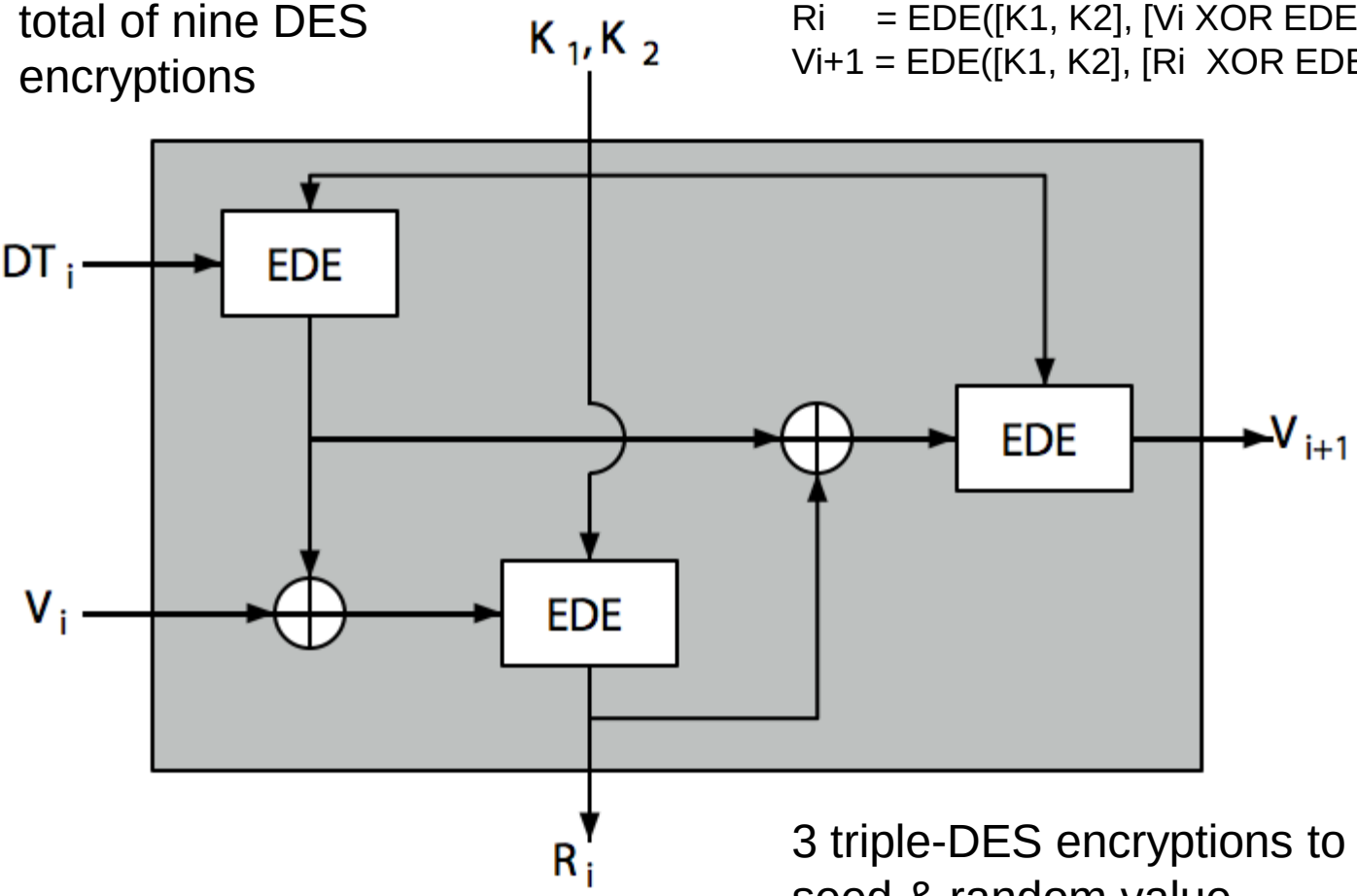
  $X_i = E_K[X_{i-1}]$



ANSI X9.17 PRG

DTi - Date/time value at the beginning of ith generation stage
Vi - Seed value at the beginning of ith generation stage
Ri - Pseudorandom number produced by the ith generation stage
K1, K2 - DES keys used for each stage
Then compute successive values as:
Ri = EDE([K1, K2], [Vi XOR EDE([K1, K2], DTi)])
Vi+1 = EDE([K1, K2], [Ri XOR EDE([K1, K2], DTi)])

112-bit key and three EDE encryptions for a total of nine DES encryptions



3 triple-DES encryptions to generate a new seed & random value

Blum Blum Shub Generator

- based on public key algorithms
- use least significant bit from iterative equation:
  - $n=p.q$, and large primes $p,q=3 \bmod 4$
  - $x_0 = s^2 \bmod n \quad \text{where } (s,p) = (s,q) = 1$
  - $x_{i+1} = x_i^2 \bmod n$
- unpredictable, passes **next-bit** test
- security rests on difficulty of factoring N
- is unpredictable given any run of bits
- slow, since very large numbers must be used
- too slow for cipher use, good for key generation

PERFECT SECRECY
(Information Theoretic Cryptography)

- Computational Security
- Unconditional Security
  - based on information theory

Our aim is to answer to questions like:

- What can adversary learn from ciphertexts
  - How much information is leaked by ciphertext about the plaintext or the key?
- What is *information*?
- Can two different keys decrypt the same ciphertext to two different but valid plaintexts?
- What should be the length of

BASICS

- X, Y random variables (r.v.)
  - A random variable is a mapping from a probability space to real numbers
- P(x) = prob. that X takes value x
- P(y) = prob. that Y takes value y
- P(x|y) = conditional prob. X takes on the value x given that Y takes on value y.
- P(x,y) = prob. that X=x and Y=y
- X and Y are independent if

$$P(x,y) = P(x) \cdot P(y) \qquad \forall x,y$$

Bayes' Theorem

- Relationship between joint probability and conditional probability.

Bayes' Theorem:

$$\text{If } P(y) > 0 \text{ then } P(x \mid y) = \frac{P(x) \cdot P(y \mid x)}{P(y)}$$

Corollary:

X, Y are independent r.v. iff $P(x|y) = P(x) \quad \forall x,y$

→ How do we make use of these?...

Perfect Secrecy

- Assumptions:
  - A key is used only for one encryption.
  - P is the plaintext space with a probability distribution.
  - Key k is chosen by Alice and Bob using some fixed prob. distribution.
  - k and plaintext x are independent events.

Perfect Secrecy

- Observation
  - Two prob. distributions on P and *K* induce a prob. dist. on ciphertext space C.
- Can we compute probability $P_C(y)$ $\therefore$ y is the ciphertext that is transmitted?

For $k \in K$ define

$C(k) = \{e_k(x) : x \in P\}$

i.e. C(k) is the set of possible ciphertexts if k is the key.

We must consider every y in C

$$\Longrightarrow P_C(y) = \sum_{\{k:\, y \in C(k)\}} P_K(k) \cdot P_P(d_k(y))$$

Perfect Secrecy

Prob. that y is the ciphertext
given x is the plaintext

$$P_C(y \mid x) = \sum_{\{k:\, x = d_k(y)\}} P_K(k)$$

Prob. that x is the plaintext
given y is the ciphertext

$$P_P(x \mid y) = \frac{P_P(x) \sum_{\{k:\, x = d_k(y)\}} P_K(k)}{\sum_{\{k:\, y \in C(k)\}} P_K(k)\ P_P(d_k(y))}$$

Example 1

- P={a,b} with $P_P(a)=1/4$
  $P_P(b)=3/4$
- $K = (k_1,k_2,k_3)$ with $P_K(k_1)=1/2$
  $P_K(k_2)=1/4$
  $P_K(k_3)=1/4$
- C = {1,2,3,4}

$$e_{k_1}(a)=1 \qquad e_{k_1}(b)=2$$
$$e_{k_2}(a)=2 \qquad e_{k_2}(b)=3$$
$$e_{k_3}(a)=3 \qquad e_{k_3}(b)=4$$

Compute prob. dist. P_C

$$P_C(1)=\frac{1}{8}$$
$$P_C(2)=\frac{3}{8}+\frac{1}{16}=\frac{7}{16}$$
$$P_C(3)=\frac{3}{16}+\frac{1}{16}=\frac{1}{4}$$
$$P_C(4)=\frac{3}{16}$$

Example 1

- Now we can compute conditional prob. dist. on the plaintext given that a certain ciphertext has been observed.

$P_P(a|1) = 1$ $P_P(b|1) = 0$

$P_P(a|2) = 1/7$ $P_P(b|2) = 6/7$

$P_P(a|3) = 1/4$ $P_P(b|3) = 3/4$

$P_P(a|4) = 0$ $P_P(b|4) = 1$

•Perfect Secrecy: No information can be obtained about plaintext by observing the ciphertext.

•Formally: A cryptosystem has perfect secrecy if $P_P(x|y) = P_P(x)$ for all $x \in P$ and $y \in C$.

•In the example, is there a ciphertext that satisfies perfect secrecy?

Generalization – *Theorem (Shannon)*:

Suppose $(P,C,K,\mathcal{E},D)$ is a cryptosystem where $|K|=|C|=|P|$. Then the cryptosystem provides perfect secrecy iff every key is used with equal prob. $1/|K|$, and $\forall x \in P$ and $\forall y \in C$, $\exists$ unique key k s.t. $e_k(x)=y$.

Example (theorem)

- Suppose the 26 keys in the SHIFT CIPHER (SC) are used with equal probaility 1/26. Then for any plaintext probability distribution, the SC has perfect secrecy.

- **Proof**: P=C=*K*=Z_{26} and for $0 \leq K \leq 25$

  $C_k(x) = x + k \bmod 26 \quad (x \in Z_{26})$

- Let $y \in Z_{26}$ then $P_C(y) = \sum_{k \in Z_{26}} P_K(k) P_P(d_k(y))$

$$= \sum_{k \in Z_{26}} \frac{1}{26} P_P(y-k)$$

$$= \frac{1}{26} \sum_{k \in Z_{26}} P_P(y-k)$$

Example (theorem)

For fixed y, the values y-k mod 26 comprise a permutation of Z_{26}.

$$\sum_{k \in Z_{26}} P_P(y-k) = \sum_{y \in Z_{26}} P_P(y) = 1$$

$$\Rightarrow P_C(y) = \frac{1}{26} \quad \text{for any } y \in Z_{26}$$

$$\text{Next we have that } P_C(y \mid x) = P_K(y - x \bmod 26) = \frac{1}{26}$$

for every x,y since for every x,y the unique key k such that $e_k(x)=y$ is $k=y-x \bmod 26$.

Now apply Bayes' theorem $$P_p(x \mid y) = \frac{P_p(x)\,\frac{1}{26}}{\frac{1}{26}} = P_p(x)$$

One-time Pad

(Gilbert Vernam's 1917)

- Let $n \geq 1$ be an integer and take $P=C=K=(Z_2)^n$.
- For $k \in (Z_2)^n$ define $e_k(x)$ for

  $x = (x_1 \ldots x_n)$

  $k = (k_1 \ldots k_n)$

  $e_k(x) = (x_1+k_1, \ldots , x_n+k_n) \bmod 2$, and

  $d_k(y) = (y_1+k_1, \ldots , y_n+k_n) \bmod 2$

Cannot reuse a key! $\Rightarrow$ not practical.

Information and Uncertainty

Example: ROP_ what do you expect as the next letter?
E? Or B? or something else?

→ the number of possible outcomes is important

Guess a prime X between 10 and 20
X={11,13,17,19} uncertainty is 4

- *amount of information obtained after observing the outcome*

=

- *amount of uncertainty before seeing the outcome*

Uncertainty about an outcome is inversely related to its probability

$$P(X=11)=1/4$$

Mathematically quantifying information

- The concept of information is too broad to be captured completely by a single definition.
- However, for any probability distribution, we define a quantity called the entropy, which has many properties with intuitive interpretations.
- Entropy is a measure of the uncertainty of a random variable.

Mathematically quantifying information

- Consider an experiment with M outcomes

$$\mathcal{X} = \{x_1, x_2, \ldots, x_M\}$$

Notations:

- **information source:** the experiment
- $\mathcal{X}$: alphabet
- x_m: outcome m
- p_m: the probability of outcome m

Examples:

- outcome of a soccer match
- text, image, voice, video
- training data for machine learning
- observational data for inference

information/uncertainty about outcome $x_m = \log \frac{1}{p_m}$

Courtesy of Prof. Ali Tajer: Notes on Information Theory.

ENTROPY

- Mathematical measure of information or uncertainty.
- Computed as a fraction of a probability distribution.

DEFINITION: Suppose X is a (discrete) r.v. which takes on a finite set of values according to a pmf. p(X). Then the entropy of this prob. dist. is defined as

$$H(X) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

If the possible values of X are x_i, $1 \leq i \leq n$ then we have

$$H(X) = -\sum_{i=1}^{n} p(X = x_i) \log_2 p(X = x_i)$$

If $p_i = 1/n$ for $1 \leq i \leq n$ then $H(X) = \log_2 n$.

Entropy of a source as the expected value of the information it contains

Entropy

For an information source with alphabet $\mathcal{X} = \{x_1, \ldots, x_M\}$ and probability mass function (PMF)

$$p_X(x) = \mathbb{P}(X = x)$$

we define the entropy as

$$H(X) = \mathbb{E}\left[\log \frac{1}{p_X(x)}\right] = \sum_{m=1}^{M} p_X(x_m) \log \frac{1}{p_X(x_m)}$$

Entropy-Formal

Definition 1.1 (Entropy)**.** Let X be a discrete RV with pmf P_X. The entropy of X is

$$\begin{aligned} H(X) &\triangleq \mathbb{E}_X\left[\log \frac{1}{P_X(X)}\right] \\ &= \sum_{x\in X} P_X(x) \log \frac{1}{P_X(x)} \\ &= -\sum_{x\in X} P_X(x) \log P_X(x) \\ &= -\mathbb{E}_X\left[\log P_X(X)\right] \ . \end{aligned}$$

Remark 1.1. The entropy of X can be also interpreted as the expected value of the random variable $\frac{1}{P_X(X)}$. Note that $P_X(X)$ is a random variable, while $P_X(x)$ for any atom x is a deterministic value.

Courtesy of Prof. Ali Tajer: Notes on Information Theory.

Binary Entropy Function

$$h(p) \triangleq p \log \frac{1}{p} + \bar{p} \log \frac{1}{\bar{p}}$$

maximal entropy / uncertainty / information

- $h(p)$ is symmetric with respect to line $p = \frac{1}{2}$.
- $h(p)$ is maximized at $p = \frac{1}{2}$.
- $h(p)$ is continuous and concave over [0.1], i.e.,

$$\frac{\partial^2 h(p)}{\partial p^2} < 0$$

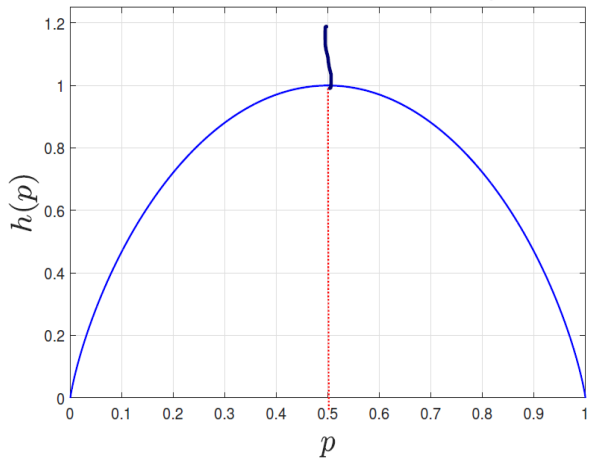


Figure. $h(p)$ versus p.

The 20 Question Game

- You have to guess the value of a random variable $\in \{a, b, c, d\}$ with probabilities

$$p_X(a) = \frac{1}{2} \qquad p_X(b) = \frac{1}{4} \qquad p_X(c) = \frac{1}{8} \qquad p_X(d) = \frac{1}{8}$$

- One strategy to play: start with the most likely answer
  - Is the outcome $X = a$?
  - If not, is the outcome $X = b$?
  - If not, is the outcome $X = c$?
  - If not, is the outcome $X = d$?
- The **average** number of question we ask

$$1 \times \frac{1}{2} + 2 \times \frac{1}{4} + 3 \times \frac{1}{8} + 3 \times \frac{1}{8} = 1.75$$

- Let us also compute the **entropy** of X

$$H(X) = \sum_{m=1}^{M} p_X(x_m) \log \frac{1}{p_X(x_m)} = \frac{1}{2} \log 2 + \frac{1}{4} \log 4 + \frac{1}{8} \log 8 + \frac{1}{8} \log 8 = 1.75$$

Operational Interpretation of Entropy

Entropy of X is the **average** number of bits required for describing X

Courtesy of Prof. Ali Tajer: Notes on Information Theory.

Joint Entropy

We can generalize the notion of entropy to more than one information source

- Example (correlated sources):

How much information do we get from an image (source 1) and its label (source 2)?

- Example (independent sources):

How much uncertainty do we have about the outcome of two coin flips?

Joint Entropy

Joint entropy of two random variables X and Y with joint PMF $p_{XY}(x, y)$ is

$$H(X, Y) = \mathbb{E}_{XY}\left[\log \frac{1}{p_{XY}(x, y)}\right]$$

Joint Entropy of Independent Sources

When X and Y are independent, we have

$$H(X, Y) = \mathbb{E}_{XY}\left[\log \frac{1}{p_{XY}(x, y)}\right] = \mathbb{E}_X\left[\log \frac{1}{p_X(x)}\right] + \mathbb{E}_Y\left[\log \frac{1}{p_Y(y)}\right] = H(X) + H(Y)$$

Conditional Entropy

Motivation: Assume two related sources of information X and Y :
X: A dataset of images and
Y is a dataset of associated labels

After observing an outcome of X how much uncertainty do we have about Y ?

How to formalize: Use the notion of conditional probability, $p_{Y|X}(y|x)$.

- Fix $X = x$ and find the entropy of $Y|X = x$ for this specific choice of $X = x$:

$$H(Y|X = x) = \sum_{y \in \mathcal{Y}} p_{Y|X}(y|x) \log \frac{1}{p_{Y|X}(y|x)}$$

- Find the average of $H(Y|X = x)$ over all possible choices of X

$$H(Y|X) = \sum_{x \in \mathcal{X}} p_X(x) H(Y|X = x) = \sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p_X(x) p_{Y|X}(y|x) \log \frac{1}{p_{Y|X}(y|x)} = \mathbb{E}_{XY}\left[\log \frac{1}{p_{Y|X}(y|x)}\right]$$

In general, $H(X|Y) \neq H(Y|X)$.

Properties of Entropy

1. **Positivity:** $H(X) \geq 0$ with equality iff X is a constant (no randomness).

2. **Maximum Entropy**: For finite $\mathcal{X}$, $H(X) \leq \log |\mathcal{X}|$ with equality iff X is uniform on $\mathcal{X}$.

3. **Relabeling:** For any bijective function f:

$$H(f(X)) = H(X) \ .$$

4. **Conditioning**: Conditioning reduces entropy: $H(X|Y) \leq H(X)$ with equality iff X and Y are independent.

5. **Small Chain Rule:**

$$\begin{aligned} H(X,Y) &= H(X) + H(Y|X) \\ &\leq H(X) + H(Y) \ . \end{aligned}$$

Properties of Entropy (cont)

6. **Full Chain Rule:**

$$H(X_1, ..., X_n) = \sum_{i=1}^{n} H(X_i|X_1, ..., X_{i-1}) \leq \sum_{i=1}^{n} H(X_i) \ .$$

with equality iff X_i's are mutually independent.

7. **Conditional Chain Rule:** For given RVs X, Y, Z we have

$$\begin{aligned} H(X, Y|Z) &= H(X|Z) + H(Y|X, Z) \\ &= H(Y|Z) + H(X|Y, Z) \ . \end{aligned}$$

8. **Entropy Under Function:** For any function f we have

$$H(f(X)) \leq H(X)$$

with equality iff f is one-to-one.

ENTROPY of a Cryptosystem

We are interested in the entropy of the various components of a cryptosystem.

- How to first compute H(*K*), H(P), H(C).

- Then compute H(*K*|C): key equivocation
  - How much information about the key is revealed by the ciphertext…

Example

- P={a,b} with $P_P(a)=1/4$; $P_P(b)=3/4$
- $K = (k_1,k_2,k_3)$ with $P_K(k_1)=1/2$; $P_K(k_2)=1/4$; $P_K(k_3)=1/4$
- C = {1,2,3,4}

Compute H(P), H(*K*), H(C) .

$$\mathrm{H(P)} = -\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right)$$

$$= -\frac{1}{4}(-2)-\frac{3}{4}(\log_2 3-2)\approx 0.81$$

$$\mathrm{H}(K)=1.5 \qquad \mathrm{H(C)}\approx 1.85 \qquad \text{(verify!)}$$

How do we apply all these?

Theorem: Let $(P,C,K,\mathcal{E},D)$ be a cryptosystem. Then

$$H(K \mid C) = H(K) + H(P) - H(C)$$

Proof: Observe that

$$H(K,P,C) = H(C|K,P) + H(K,P) \qquad \text{(chain rule)}$$

Since $y = e_k(x)$, $H(C|K,P) = 0$ (i.e. K and P uniquely determine C)

Hence $H(K,P,C) = H(K,P)$. But K and P are independent

$$\Rightarrow H(K,P) = H(K) + H(P)$$

Similarly key and ciphertext determine the plaintext uniquely

i.e. $x=d_k(y)$

$\Rightarrow H(P \mid K,C) = 0 \quad \Rightarrow H(K,P,C)=H(K,C)$

$H(K|C) = H(K,C) - H(C)$

$= H(K,P,C) - H(C)$

$= H(K) + H(P) - H(C)$ ■

Example (1),(2) continue:

We have computed $H(P) \approx 0.81$, $H(K) = 1.5$, $H(C) \approx 1.85$

What is $H(K|C) \approx 1.5 + 0.81 - 1.85 \approx 0.46$

Entropy of a natural language

e.g. English…

- H_L: measure of the average information per letter in a "meaningful" string of plaintext.
- What is the entropy of a random string of alphabetical characters?
  - $\log_2 26 \approx 4.70$
- First order approximation:
  - Consider each letter

- Let P^n be the r.v. that has as its prob. distribution that of all n-grams of plaintext.
- The entropy of a natural language L is

$$H_L = \lim_{n\to\infty} \frac{H(P^n)}{n}$$

and the redundancy of L is

$$R_L = \underbrace{1 - \frac{H_L}{\log_2 |P|}}_{\text{Fraction of "excess characters"}}$$

for n=2 $H(P^2)/2 \approx$ 3.90

n=3 …

..

- Empirical result: $1.0 \leq H_L \leq 1.5$
- Avg information content in English: 1-1.5 bits per letter.

- Then $R_L \approx 0.75$. What does it mean English is 75% redundant?

  $\Rightarrow$ There is a Hoffman coding with ¼ of the length of the original text!

- How are we going to use this info?
  - Given prob. distr. on *K* and P^n, we can define induced prob. distr. on C^n (the set of n-grams of ciphertext)
  - Recall P^n was a r.v. for n-gram plaintext.
  - Define C^n for n-gram of ciphertext.

  $\Rightarrow$ We can relate the average number of spurious keys to $H(K \mid C^n)$ as

  $$H(K \mid C^n) \leq \log_2(\bar{s}_n + 1)$$

  since $H(K \mid C^n) \geq H(K) - nR_L \log_2(P)$

Theorem:

$$\bar{s}_n \geq \frac{|K|}{|P|^{nR_L}} - 1 \qquad \text{as } n \uparrow \;\; \bar{s}_n \to \infty$$

Definition: (UNICITY)

Unicity distance of a cryptosystem: value of $n=n_0$ at which $\bar{s}_{n_0} = 0$

Set $\bar{s}_n = 0$ and solve for n

$$n_0 \approx \frac{\log_2 |K|}{R_L \log_2 |P|}$$

Example: Substitution Cipher

- $|P| = 26$
- $|K| = 26!$
- Take $R_L = 0.75$

$$n_0 \approx 88.4/(0.75 \times 4.75) \approx 25$$

$\Rightarrow$ Given a ciphertext string of length at least 25, a unique decryption is possible.