CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. Qi Wang

Department of Computer Science and Engineering
Office: Room413, CoE South Tower
Email: wangqi@sustech.edu.cn

What does "pseudorandom" mean?

- Informal: cannot be distinguished from uniform ("random")

What does "pseudorandom" mean?

- Informal: cannot be distinguished from uniform ("random")
- Which of the following is pseudorandom?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000

What does "pseudorandom" mean?

- Informal: cannot be distinguished from uniform ("random")
- Which of the following is pseudorandom?
  - 0101010101010101
  - 0010111011100110
  - 0000000000000000
- *Pseudorandomness* is a property of a *distribution*, not a string.

Pseudorandomness

- Fix some distribution D on n-bit strings
  - $x \leftarrow D$ means "sample x according to D"

Pseudorandomness

- Fix some distribution D on n-bit strings
  - $x \leftarrow D$ means "sample x according to D"

- Historically, D was considered *pseudorandom* if it "*passed a bunch of statistical tests*"
  - $\Pr_{x \leftarrow D}[1^{st} \text{ bit of } x \text{ is } 1] \approx 1/2$
  - $\Pr_{x \leftarrow D}[\text{parity of } x \text{ is } 1] \approx 1/2$
  - $\Pr_{x \leftarrow D}[A_i(x) = 1] \approx \Pr_{x \leftarrow U_n}[A_i(x) = 1]$ for $i = 1, \ldots, 20$

Pseudorandomness

- Fix some distribution D on n-bit strings
  - $x \leftarrow D$ means "sample x according to D"

- Historically, D was considered *pseudorandom* if it "*passed a bunch of statistical tests*"
  - $\Pr_{x \leftarrow D}[1^{st}$ bit of x is $1] \approx 1/2$
  - $\Pr_{x \leftarrow D}[$parity of x is $1] \approx 1/2$
  - $\Pr_{x \leftarrow D}[A_i(x) = 1] \approx \Pr_{x \leftarrow U_n}[A_i(x) = 1]$ for $i = 1, \ldots, 20$

  This is not sufficient, since it is not possible to know what statistical test an attacker will use.

Pseudorandomness

- Cryptographic definition of *pseudorandomness*
  - D is *pseudorandom* if it passes all *efficient* statistical tests

Pseudorandomness

- Cryptographic definition of *pseudorandomness*
  - D is *pseudorandom* if it passes all *efficient* statistical tests

(Concrete) Let D be a distribution on p-bit strings. D is (t, ϵ)-*pseudorandom* if for all A running in time at most t,

$$\left|\Pr_{x \leftarrow D}[A(x) = 1] - \Pr_{x \leftarrow U_p}[A(x) = 1]\right| \leq \epsilon$$

Pseudorandomness

- Cryptographic definition of *pseudorandomness*
  - D is *pseudorandom* if it passes all *efficient* statistical tests

(Concrete) Let D be a distribution on p-bit strings. D is (t,ϵ)-*pseudorandom* if for all A running in time at most t,

$$|\Pr_{x\leftarrow D}[A(x)=1]-\Pr_{x\leftarrow U_p}[A(x)=1]|\le\epsilon$$

(Asymptotic) *Security parameter n*, polynomial p

Definiton 3.2 Let D_n be a distribution over $p(n)$-bit strings. $\{D_n\}$ is *pseudorandom* if for all probabilistic, polynomial-time (PPT) distinguishers A, there is a negligible function ϵ such that

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

Pseudorandom generators (PRGs)

- A *PRG* is an efficient, deterministic algorithm that expands a *short*, *uniform seed* into a *longer*, *pseudorandom* output

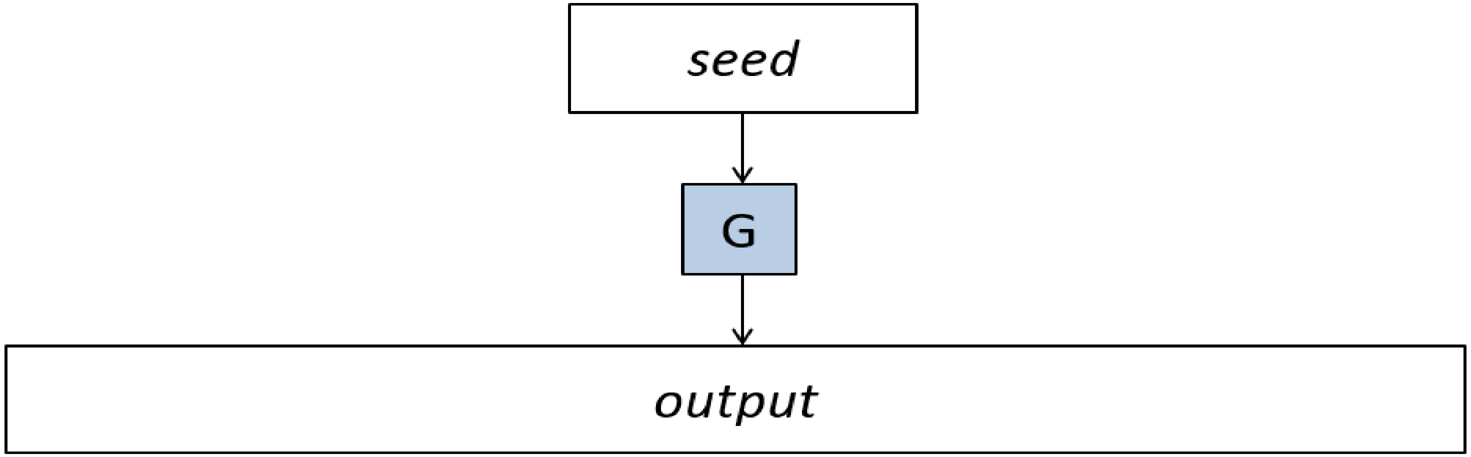
Pseudorandom generators (PRGs)

- 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

Pseudorandom generators (PRGs)

- 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

Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$.



Pseudorandom generators (PRGs)

- 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

Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$.

G defines a sequence of distributions.

- D_n = the distribution on $p(n)$-bit strings defined by choosing $x \leftarrow U_n$ and outputting $G(x)$
- $\Pr_{D_n}[y] = \Pr_{U_n}[G(x) = y] = \sum_{x:\ G(x)=y} \Pr_{U_n}[x]$

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

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

PRGs

- A *PRG* is an efficient, deterministic algorithm that expands a *short*, *uniform seed* into a *longer*, *pseudorandom* output

  Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$.

- For all efficient distinguishers A, there is a negligible function ϵ such that

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

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

PRGs

- For all efficient distinguishers A, there is a negligible function ϵ such that
$$\left|\Pr_{x \leftarrow U_n}[A(G(x)) = 1] - \Pr_{y \leftarrow U_{p(n)}}[A(y) = 1]\right| \leq \epsilon(n)$$

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

PRGs

- For all efficient distinguishers A, there is a negligible function ϵ such that

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

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

- PRGs are limited
  - They have fixed-length output
  - They produce the entire output in "one shot"
  - In practice, PRGs are based on *stream ciphers*
  - Can be viewed as producing an "unbounded" stream of pseudorandom bits, on demand
  - Will revisit later

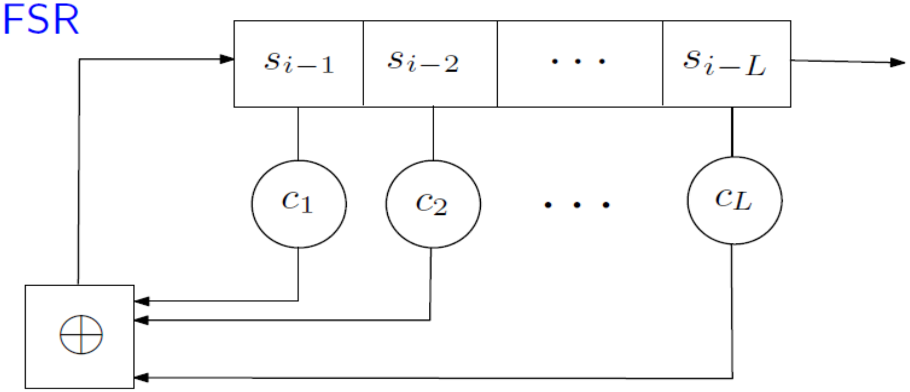
Do PRGs/stream ciphers exist?

- We don't know ...
  - We will assume certain algorithms are PRGs
  - Can construct PRGs from weaker assumptions (later)

Do PRGs/stream ciphers exist?

- We don't know ...
  - We will assume certain algorithms are PRGs
  - Can construct PRGs from weaker assumptions (later)

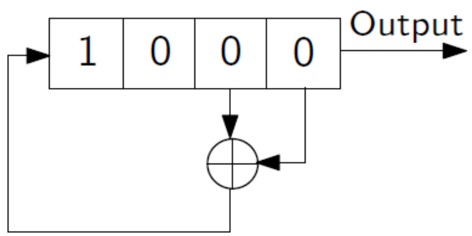
LFSR



Linear Feedback Shift Register (LFSR)

Example

$$\mathbf{s} = (000100110101111)^{15}$$



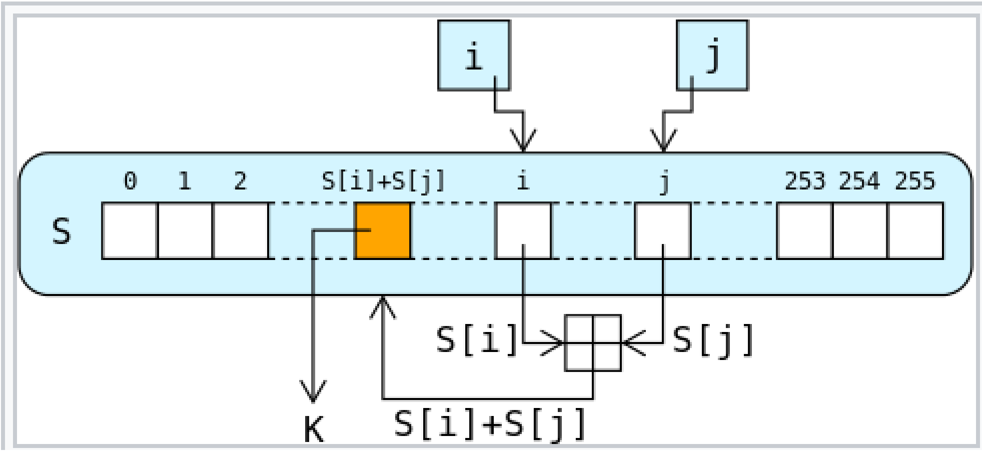
$\mathrm{LC}(\mathbf{s}) = 4$

$P_{\mathbf{s}}(x) = x^4 + x + 1$

Practical "PRGs"

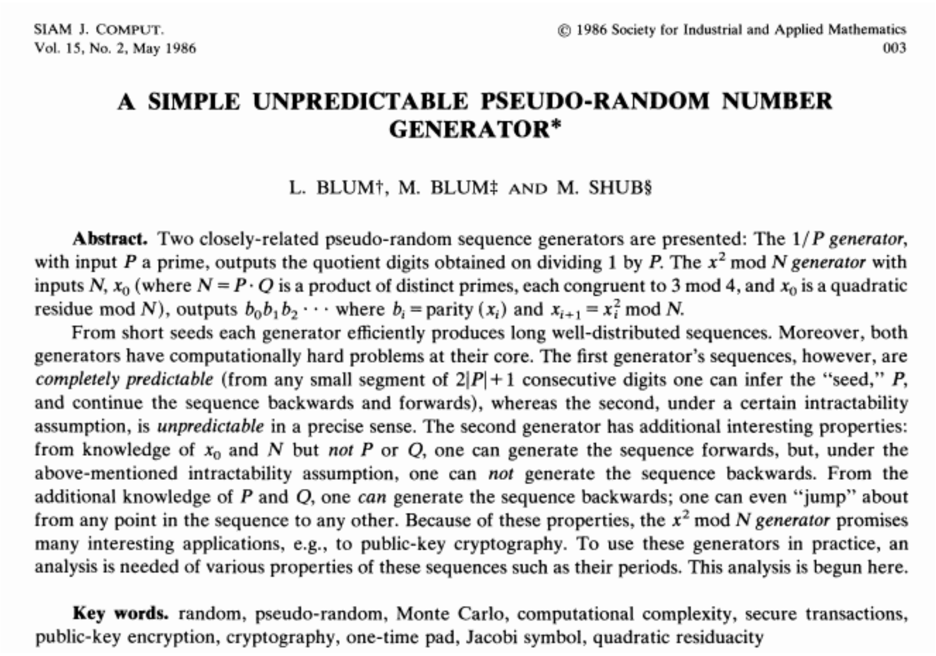
- RC4

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
endwhile
```



Blum-Blum-Shub

```
num_outputted = 0;
while num_outputted < m:
    X := X*X mod N
    num_outputted := num_outputted + 1
    output least-significant-bit(X)
```



SIAM J. COMPUT.
Vol. 15, No. 2, May 1986

© 1986 Society for Industrial and Applied Mathematics
003

A SIMPLE UNPREDICTABLE PSEUDO-RANDOM NUMBER GENERATOR*

L. BLUM†, M. BLUM‡ AND M. SHUB§

Abstract. Two closely-related pseudo-random sequence generators are presented: The $1/P$ *generator*, with input P a prime, outputs the quotient digits obtained on dividing 1 by P. The x^2 mod N *generator* with inputs N, x_0 (where $N = P \cdot Q$ is a product of distinct primes, each congruent to 3 mod 4, and x_0 is a quadratic residue mod N), outputs $b_0 b_1 b_2 \cdots$ where $b_i = \text{parity}(x_i)$ and $x_{i+1} = x_i^2$ mod N.

From short seeds each generator efficiently produces long well-distributed sequences. Moreover, both generators have computationally hard problems at their core. The first generator's sequences, however, are *completely predictable* (from any small segment of $2|P|+1$ consecutive digits one can infer the "seed," P, and continue the sequence backwards and forwards), whereas the second, under a certain intractability assumption, is *unpredictable* in a precise sense. The second generator has additional interesting properties: from knowledge of x_0 and N but *not* P or Q, one can generate the sequence forwards, but, under the above-mentioned intractability assumption, one can *not* generate the sequence backwards. From the additional knowledge of P and Q, one *can* generate the sequence backwards; one can even "jump" about from any point in the sequence to any other. Because of these properties, the x^2 mod N *generator* promises many interesting applications, e.g., to public-key cryptography. To use these generators in practice, an analysis is needed of various properties of these sequences such as their periods. This analysis is begun here.

Key words. random, pseudo-random, Monte Carlo, computational complexity, secure transactions, public-key encryption, cryptography, one-time pad, Jacobi symbol, quadratic residuacity

Where things stand

- We saw that there are some inherent limitations if we want *perfect security*
  - In particular, key must be as long as the message

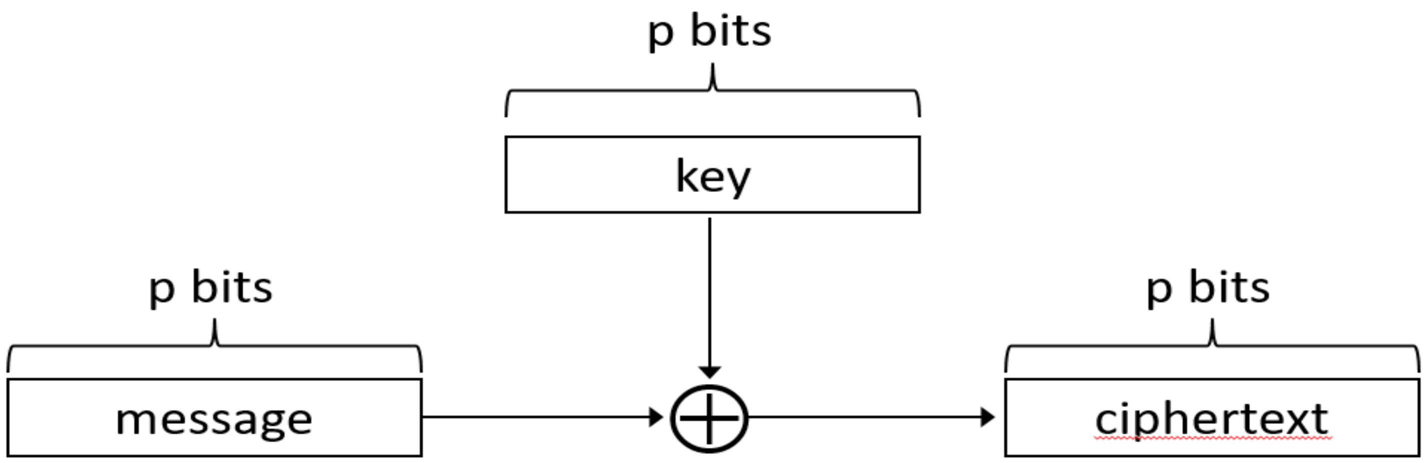
Where things stand

- We saw that there are some inherent limitations if we want *perfect security*
  - In particular, key must be as long as the message

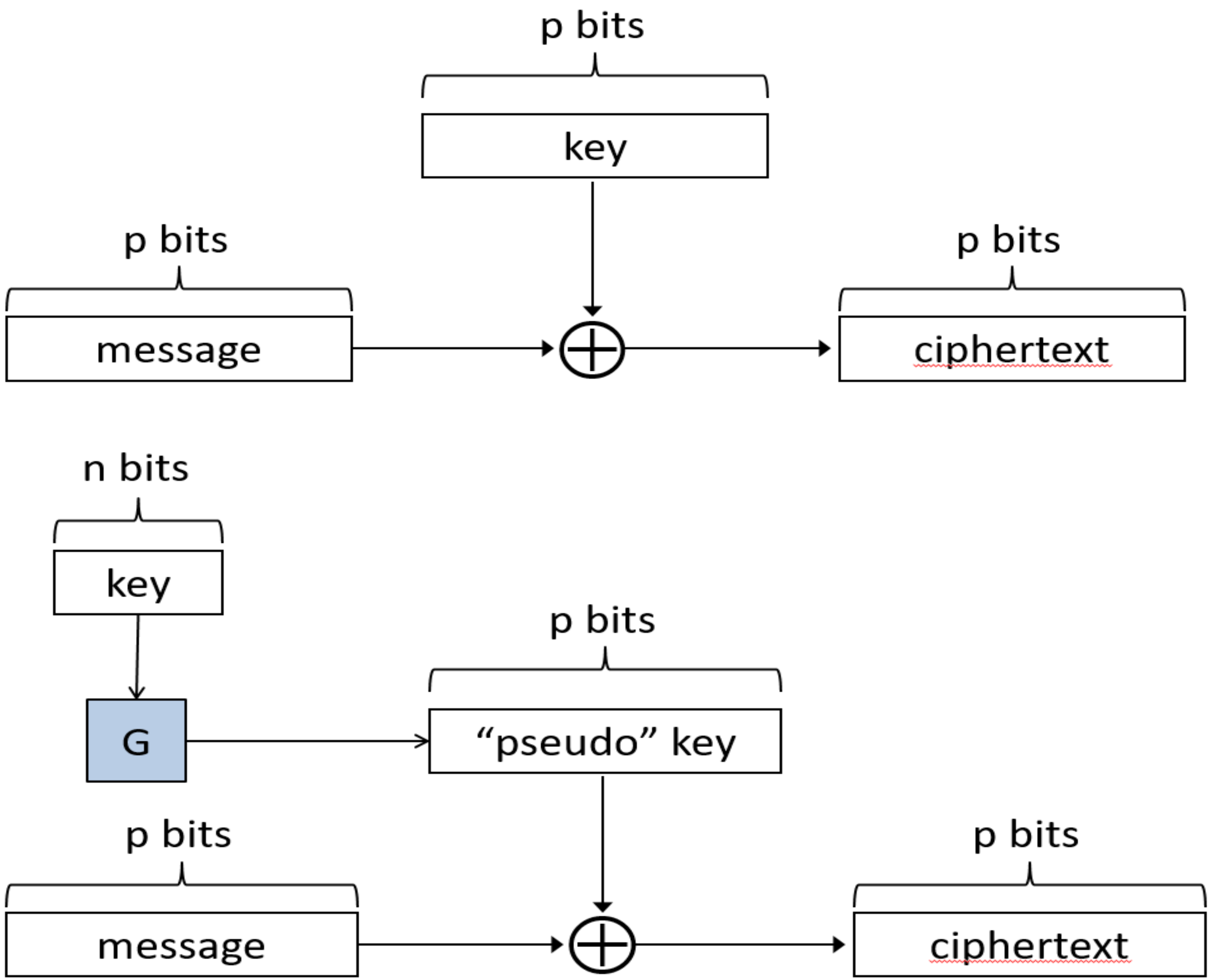
We defined *computational security*, a relaxed notion of security

$\mathcal{Q}$: Can we overcome prior limitations?

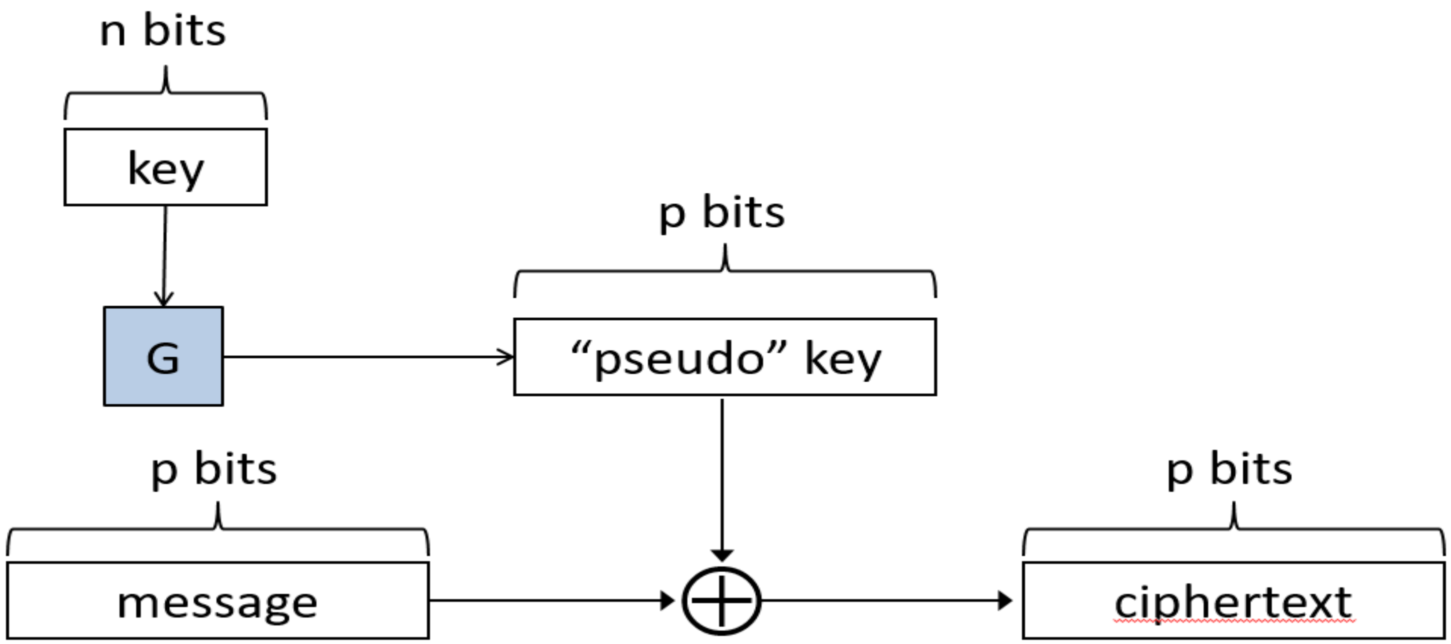
Recall: one-time pad



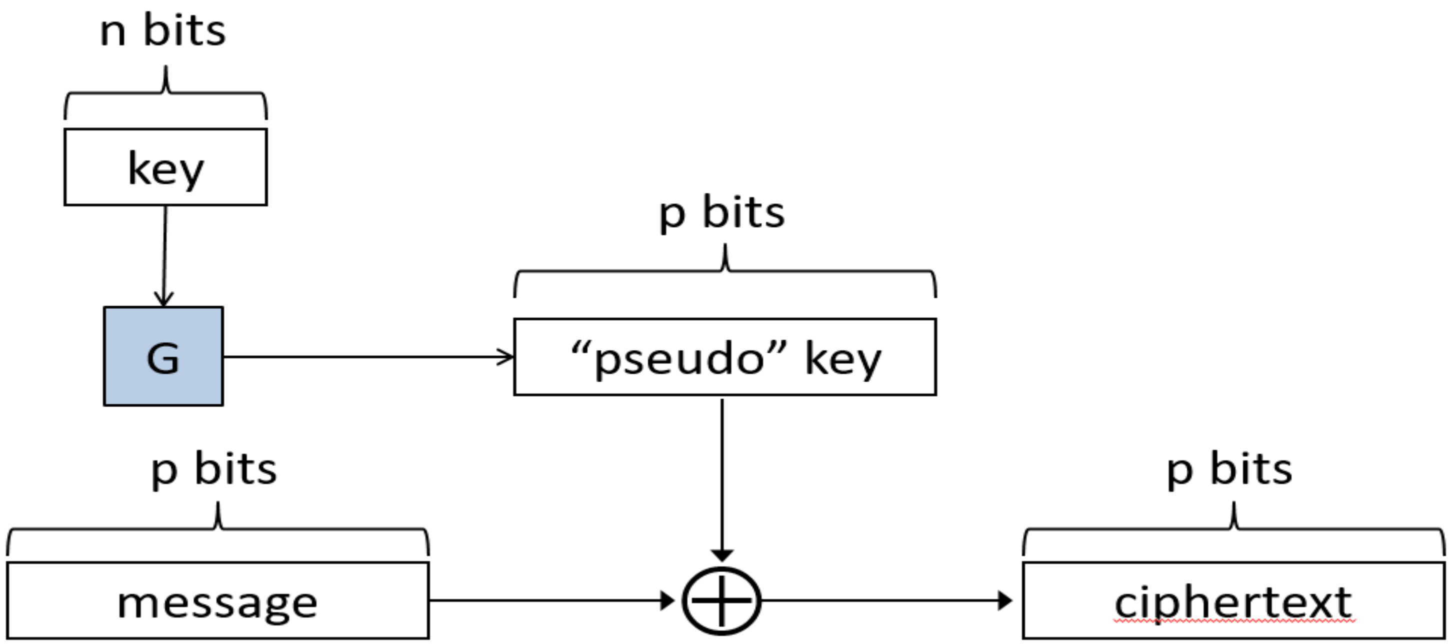
Recall: one-time pad



Pseudo one-time pad



Pseudo one-time pad



- Let G be a deterministic, with $|G(k)| = p(|k|)$

  $Gen(1^n)$: output uniform n-bit key k

  – Security parameter $n \Rightarrow$ message space $\{0,1\}^{p(n)}$

  $Enc_k(m)$: output $G(k) \oplus m$

  $Dec_k(m)$: output $G(k) \oplus c$

Proof by reduction

1. Assume that G is a *PRG*
2. Assume toward a contradiction that there is an efficient attacker A who "breaks" the pseudo-OTP scheme
3. Use A as a subroutine to build an efficient D that "breaks" *pseudorandomness* of G

– By assumption, no such D exists!

$\Rightarrow$ No such A can exist

Proof by reduction

- 1. Assume that G is a *PRG*

  2. Assume toward a contradiction that there is an efficient attacker A who "breaks" the pseudo-OTP scheme

  3. Use A as a subroutine to build an efficient D that "breaks" *pseudorandomness* of G

  – By assumption, no such D exists!

  $\Rightarrow$ No such A can exist

Theorem 3.3 If G is a pseudorandom generator (PRG), then the pseudo one-time pad (pseudo-OTP) Π is *EAV-secure* (i.e., *computationally secure*)

The reduction

- **Proof.**

The reduction

- **Proof.**
  Fix Π, A
  Define a randomized experiment $PrivK_{A,\Pi}(n)$:

  1. $A(1^n)$ outputs $m_0, m_1 \in \{0,1\}^*$ of equal length
  2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0,1\}$, $c \leftarrow Enc_k(m_b)$
  3. $b' \leftarrow A(c)$

  Adversary A *succeeds* if $b = b'$, and we say the experiment evaluates to 1 in this case.

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

The reduction

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

The reduction

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

The reduction

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

  This means, $\Pr[A(Enc_{U_n}(m)) = 1] - \Pr[A(U_{p(n)}) = 1] > 1/poly(n)$

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

  This means, $\Pr[A(Enc_{U_n}(m)) = 1] - \Pr[A(U_{p(n)}) = 1] > 1/poly(n)$

  $$|\Pr[A(G(U_n) \oplus m) = 1] - \Pr[A(U_{p(n)}) = 1]| > 1/poly(n)$$

The reduction

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

  This means, $\Pr[A(Enc_{U_n}(m)) = 1] - \Pr[A(U_{p(n)}) = 1] > 1/poly(n)$

  $$|\Pr[A(G(U_n) \oplus m) = 1] - \Pr[A(U_{p(n)}) = 1]| > 1/poly(n)$$

  $Enc_{U_n}()$ is not pseudorandom

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

  This means, $\Pr[A(Enc_{U_n}(m)) = 1] - \Pr[A(U_{p(n)}) = 1] > 1/poly(n)$

  $$|\Pr[A(G(U_n) \oplus m) = 1] - \Pr[A(U_{p(n)}) = 1]| > 1/poly(n)$$

  $Enc_{U_n}()$ is not pseudorandom

  Define $D : \{0,1\}^{p(n)} \to \{0,1\}$ as: $D(y) = A(y \oplus m)$, which means $A(z) = D(z \oplus m)$. Note that D is also efficient. But we have

  $$|\Pr[D(G(U_n)) = 1] - \Pr[D(U_{p(n)} \oplus m) = 1]| \geq 1/poly(n)$$

- **Proof.**

  Definition 3.1 Π is *computationally indistinguishable* (aka *EAV-secure*) if for all PPT attackers (algorithms) A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  Suppose to the contrary that there exists an efficient attacker A, s.t.

  $$\Pr[PrivK_{A,\Pi}(n) = 1] > 1/2 + 1/poly(n)$$

  This means, $\Pr[A(Enc_{U_n}(m)) = 1] - \Pr[A(U_{p(n)}) = 1] > 1/poly(n)$

  $$|\Pr[A(G(U_n) \oplus m) = 1] - \Pr[A(U_{p(n)}) = 1]| > 1/poly(n)$$

  $Enc_{U_n}()$ is not pseudorandom

  Define $D : \{0,1\}^{p(n)} \to \{0,1\}$ as: $D(y) = A(y \oplus m)$, which means $A(z) = D(z \oplus m)$. Note that D is also efficient. But we have

  $$|\Pr[D(G(U_n)) = 1] - \Pr[D(U_{p(n)} \oplus m) = 1]| \geq 1/poly(n)$$

  Since $U_{p(n)} \oplus m \equiv U_{p(n)}$, this contradicts that G is a PRG.

Proof by reduction (alternatively)

1. Assume that G is a *PRG*
2. Fix some arbitrary, efficient A attacking the pseudo-OTP scheme
3. Use A as a subroutine to build an efficient D attacking G
   – Relate the distinguishing probability of D to the success probability of A
4. By assumption, the distinguishing probability of D must be negligible

Proof by reduction (alternatively)

- 1. Assume that G is a *PRG*

  2. Fix some arbitrary, efficient A attacking the pseudo-OTP scheme

  3. Use A as a subroutine to build an efficient D attacking G
     – Relate the distinguishing probability of D to the success probability of A

  4. By assumption, the distinguishing probability of D must be negligible

     $\Rightarrow$ Bound the success probability of A

The reduction

- Let $\mu(n) = \Pr[PrivK_{A,\Pi}(n) = 1]$

The reduction

- Let $\mu(n) = \Pr[PrivK_{A,\Pi}(n) = 1]$

  If distribution of y is pseudorandom, then the view of A is *exactly* the same as in $PrivK_{A,\Pi}(n)$
  $\Rightarrow \Pr_{x \leftarrow U_n}[D(G(x)) = 1] = \mu(n)$

The reduction

- Let $\mu(n) = \Pr[PrivK_{A,\Pi}(n) = 1]$

  If distribution of y is pseudorandom, then the view of A is *exactly* the same as in $PrivK_{A,\Pi}(n)$
  $\Rightarrow \Pr_{x \leftarrow U_n}[D(G(x)) = 1] = \mu(n)$

  If distribution of y is uniform, then A succeeds with probability exactly $1/2$
  $\Rightarrow \Pr_{y \leftarrow U_{p(n)}}[D(y) = 1] = 1/2$

The reduction

- Let $\mu(n) = \Pr[PrivK_{A,\Pi}(n) = 1]$

  If distribution of y is pseudorandom, then the view of A is *exactly* the same as in $PrivK_{A,\Pi}(n)$

  $\Rightarrow \Pr_{x \leftarrow U_n}[D(G(x)) = 1] = \mu(n)$

  If distribution of y is uniform, then A succeeds with probability exactly $1/2$

  $\Rightarrow \Pr_{y \leftarrow U_{p(n)}}[D(y) = 1] = 1/2$

  Since G is pseudorandom,

  $$|\mu(n) - 1/2| \leq negl(n)$$

The reduction

- Let $\mu(n) = \Pr[PrivK_{A,\Pi}(n) = 1]$

  If distribution of y is pseudorandom, then the view of A is *exactly* the same as in $PrivK_{A,\Pi}(n)$
  $\Rightarrow \Pr_{x \leftarrow U_n}[D(G(x)) = 1] = \mu(n)$

  If distribution of y is uniform, then A succeeds with probability exactly $1/2$
  $\Rightarrow \Pr_{y \leftarrow U_{p(n)}}[D(y) = 1] = 1/2$

  Since G is pseudorandom,
  $$|\mu(n) - 1/2| \le negl(n)$$
  $\Rightarrow \Pr[PrivK_{A,\Pi}(n) = 1] \le 1/2 + negl(n)$

Have we gained anything?

- YES: the pseudo-OTP has a key shorter than the message
  – n bits vs. $p(n)$ bits

Have we gained anything?

- YES: the pseudo-OTP has a key shorter than the message
  - n bits vs. $p(n)$ bits

Recall: Perfect security has two limitations

- Key as long as the message
- Key can only be used once

Have we gained anything?

- YES: the pseudo-OTP has a key shorter than the message
  - n bits vs. $p(n)$ bits

Recall: Perfect security has two limitations

- Key as long as the message
- Key can only be used once

The pseudo OTP still has the second limitation (for the same reason as the OTP)

Have we gained anything?

- YES: the pseudo-OTP has a key shorter than the message
  - n bits vs. $p(n)$ bits

Recall: Perfect security has two limitations

- Key as long as the message
- Key can only be used once

The pseudo OTP still has the second limitation (for the same reason as the OTP)

How can we circumvent the second limitation?

But first...

- Develop an appropriate security *definition*
  - Security goal
  - Threat model

But first...

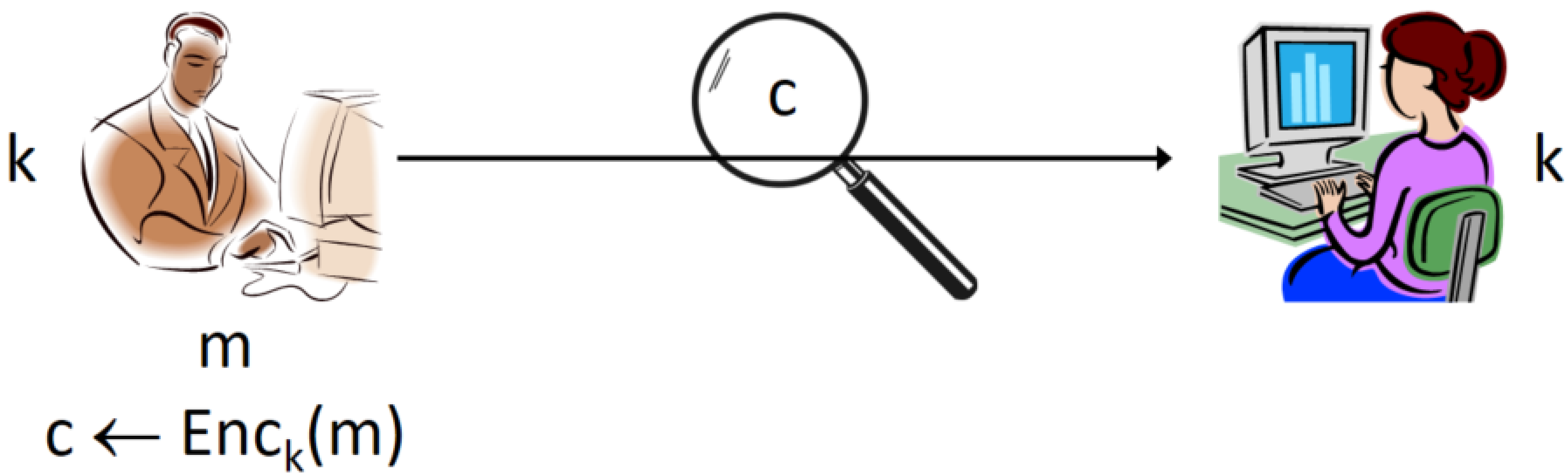
- Develop an appropriate security *definition*
  – Security goal
  – Threat model

  We will keep the security goal the same, but strengthen the threat model

But first…

- Develop an appropriate security *definition*
  - Security goal
  - Threat model

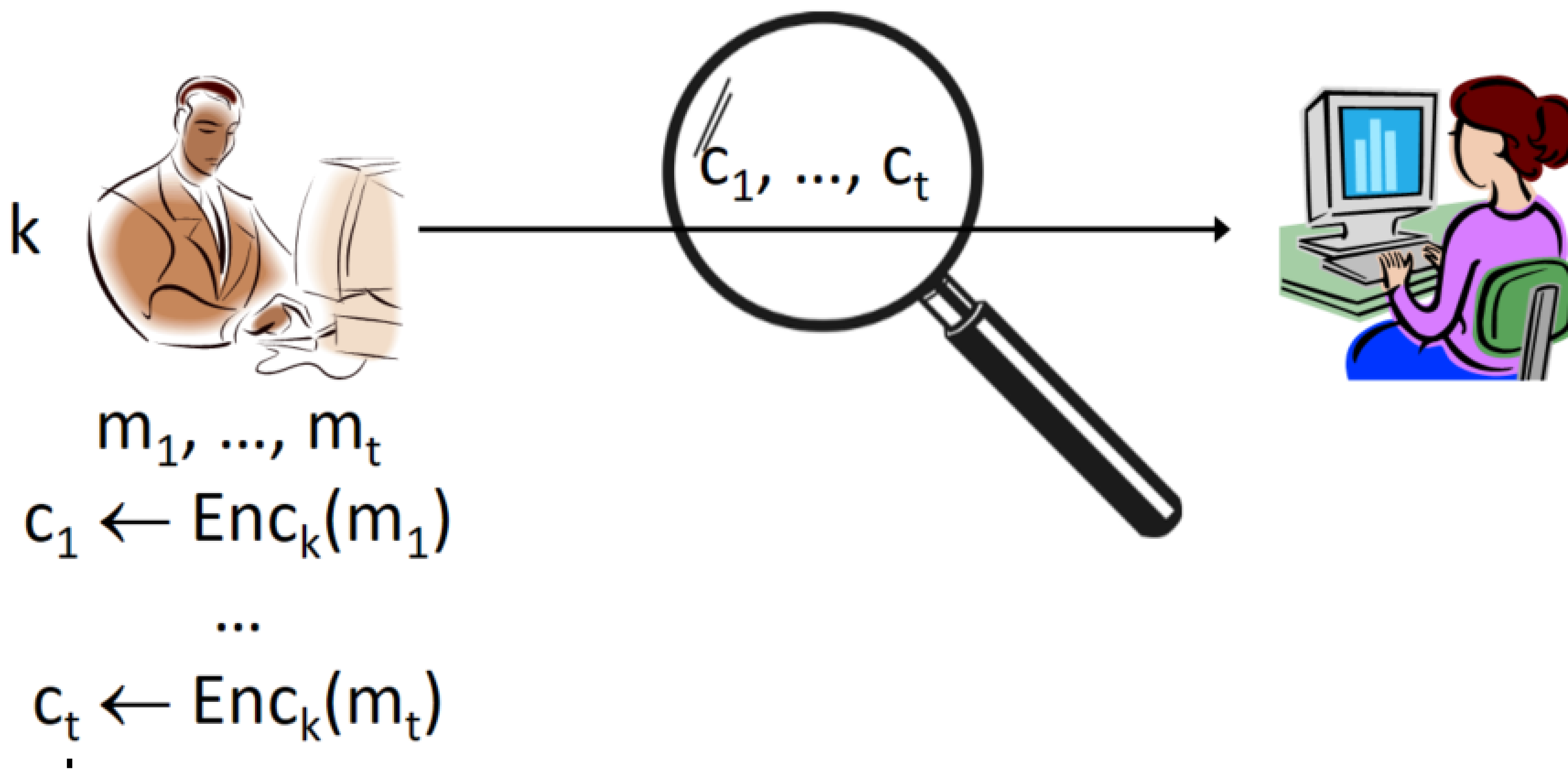
We will keep the security goal the same, but strengthen the threat model



But first…

- Develop an appropriate security *definition*
  - Security goal
  - Threat model

We will keep the security goal the same, but strengthen the threat model



A formal definition

- Fix Π, A

  Define a randomized experiment $PrivK^{mult}_{A,\Pi}(n)$:

  1. $A(1^n)$ outputs two vectors $(m_{0,1}, \ldots, m_{0,t})$ and $(m_{1,1}, \ldots, m_{1,t})$

     Required that $|m_{0,i}| = |m_{1,i}|$ for all i
  2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0,1\}$, for all i, $c_i \leftarrow Enc_k(m_{b,i})$
  3. $b' \leftarrow A(c_1, \ldots, c_t)$

  Adversary A *succeeds* if $b = b'$, and the experiment evaluates to 1 in this case.

A formal definition

- Fix Π, A

  Define a randomized experiment $PrivK^{mult}_{A,\Pi}(n)$:

  1. $A(1^n)$ outputs two vectors $(m_{0,1}, \ldots, m_{0,t})$ and $(m_{1,1}, \ldots, m_{1,t})$

     Required that $|m_{0,i}| = |m_{1,i}|$ for all i
  2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0, 1\}$, for all i, $c_i \leftarrow Enc_k(m_{b,i})$
  3. $b' \leftarrow A(c_1, \ldots, c_t)$

  Adversary A *succeeds* if $b = b'$, and the experiment evaluates to 1 in this case.

  Definition 3.4 Π is *multiple-message indistinguishable* if for all PPT attackers A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK^{mult}_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

A formal definition

- Fix Π, A

  Define a randomized experiment $PrivK^{mult}_{A,\Pi}(n)$:

  1. $A(1^n)$ outputs two vectors $(m_{0,1}, \ldots, m_{0,t})$ and $(m_{1,1}, \ldots, m_{1,t})$

     Required that $|m_{0,i}| = |m_{1,i}|$ for all i
  2. $k \leftarrow Gen(1^n)$, $b \leftarrow \{0,1\}$, for all i, $c_i \leftarrow Enc_k(m_{b,i})$
  3. $b' \leftarrow A(c_1, \ldots, c_t)$

  Adversary A *succeeds* if $b = b'$, and the experiment evaluates to 1 in this case.

  Definition 3.4 Π is *multiple-message indistinguishable* if for all PPT attackers A, there is a *negligible* function ϵ such that

  $$\Pr[PrivK^{mult}_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

  $\mathcal{Q}$: Show that the pseudo OTP is not multiple-message indistinguishable

CPA-security

- We are **not** going to work with *multiple-message secrecy*

CPA-security

- We are *not* going to work with *multiple-message secrecy*

  Instead, define someting *stronger*: security against chosen-plaintext attacks (*CPA-security*)

  – Nowadays, this is the minimal notion of security an encryption scheme should satisfy

CPA-security

- We are *not* going to work with *multiple-message secrecy*

  Instead, define someting *stronger*: security against chosen-plaintext attacks (*CPA-security*)

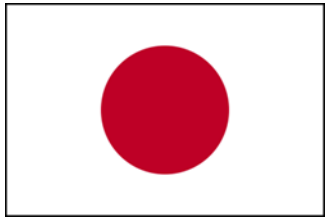
  – Nowadays, this is the minimal notion of security an encryption scheme should satisfy

  In practice, there are many ways an attacker can *influence* what gets encrypted

  – Not clear how best to model

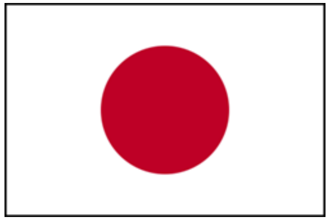
  – Chosen-plaintext attacks encompasses any such influence

"Midway" story



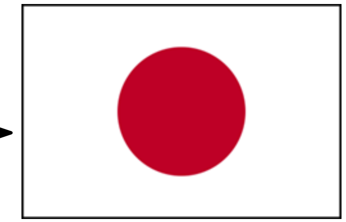
“Midway” story

Will attack AF ..



“Midway” story

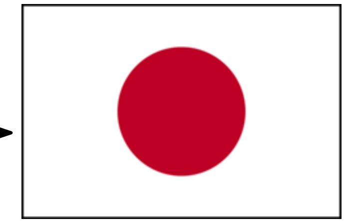
Will attack AF ..



Help! Fresh water needed

"Midway" story

AF is short of water



Help! Fresh water needed

CPA-security

- Fix Π, A

  Define a randomized experiment $PrivKCPA_{A,\Pi}(n)$:

  1. $k \leftarrow Gen(1^n)$
  2. $A(1^n)$ interacts with an *encryption oracle* $Enc_k(\cdot)$, and then outputs m_0, m_1 of the same length
  3. $b \leftarrow \{0,1\}$, $c \leftarrow Enc_k(m_b)$, give c to A
  4. A can continue to interact with $Enc_k(\cdot)$
  5. A outputs b'; A succeeds if $b = b'$, and experiment evaluates to 1 in this case

CPA-security

- Fix Π, A

Define a randomized experiment $PrivKCPA_{A,\Pi}(n)$:

1. $k \leftarrow Gen(1^n)$
2. $A(1^n)$ interacts with an *encryption oracle* $Enc_k(\cdot)$, and then outputs m_0, m_1 of the same length
3. $b \leftarrow \{0,1\}$, $c \leftarrow Enc_k(m_b)$, give c to A
4. A can continue to interact with $Enc_k(\cdot)$
5. A outputs b'; A succeeds if $b = b'$, and experiment evaluates to 1 in this case

Definition 4.1 Π is *secure against chosen-plaintext attacks (CPA-secure)* if for all PPT attackers A, there is a *negligible* function ϵ such that

$$\Pr[PrivKCPA_{A,\Pi}(n) = 1] \leq 1/2 + \epsilon(n)$$

Impossible?

- Consider the following attacker A;
  - Using a chosen-plaintext attack, get $c_0 = Enc_k(m_0)$ and $c_1 = Enc_k(m_1)$
  - Output m_0, m_1; get challenge ciphertext c
  - If $c = c_0$ output '0'; if $c = c_1$ output '1'
  - A succeeds with probability 1 (?)

Impossible?

- Consider the following attacker A;
  - Using a chosen-plaintext attack, get $c_0 = Enc_k(m_0)$ and $c_1 = Enc_k(m_1)$
  - Output m_0, m_1; get challenge ciphertext c
  - If $c = c_0$ output '0'; if $c = c_1$ output '1'
  - A succeeds with probability 1 (?)
- This attack only works if encryption is deterministic!
  - randomized encryption must be used!
  - It really is a problem if an attacker can tell when the same message is encrypted twice

Pseudorandom functions

- Informally, a *pseudorandom function* "looks like" a random (i.e., uniform) function

Pseudorandom functions

- Informally, a *pseudorandom function* "looks like" a random (i.e., uniform) function

  $Func_n$ = all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$

  How big is $Func_n$?

Pseudorandom functions

- Informally, a *pseudorandom function* "looks like" a random (i.e., uniform) function

  $Func_n$ = all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$

  How big is $Func_n$?

  – Can represent a function in $Func_n$ using $n \cdot 2^n$ bits

Pseudorandom functions

- Informally, a *pseudorandom function* "looks like" a random (i.e., uniform) function

  $Func_n$ = all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$

  How big is $Func_n$?
  – Can represent a function in $Func_n$ using $n \cdot 2^n$ bits

  $\Rightarrow |Func_n| = 2^{n \cdot 2^n}$

Pseudorandom functions

- Informally, a *pseudorandom function* "looks like" a random (i.e., uniform) function

  $Func_n$ = all functions mapping $\{0,1\}^n$ to $\{0,1\}^n$

  How big is $Func_n$?
  - Can represent a function in $Func_n$ using $n \cdot 2^n$ bits

  $\Rightarrow |Func_n| = 2^{n \cdot 2^n}$

  $\mathcal{Q}$: how many functions are there mapping from $\{0,1\}^n$ to $\{0,1\}^m$?

Random functions vs. pseudorandom functions

- Choose unifrom $f \in Func_n$

Random functions vs. pseudorandom functions

- Choose unifrom $f \in Func_n$
- **Equivalent**: 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

Random functions vs. pseudorandom functions

- Choose unifrom $f \in Func_n$
- **Equivalent**: 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

- Informally, a *pseudorandom function* "looks like" a random function
  – It does not make sense to talk about any fixed function being pseudorandom. We look instead at *keyed* functions

Keyed functions

- Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, deterministic algorithm
  - Define $F_k(x) = F(k, x)$
  - The first input is called the *key*

Keyed functions

- Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, deterministic algorithm
  - Define $F_k(x) = F(k,x)$
  - The first input is called the *key*
- Assume that F is *length preserving*: $F(k,x)$ only defined if $|k| = |x|$, in which case, $|F(k,x)| = |k| = |x|$

Keyed functions

- Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, deterministic algorithm
  - Define $F_k(x) = F(k,x)$
  - The first input is called the *key*
- Assume that F is *length preserving*: $F(k,x)$ only defined if $|k| = |x|$, in which case, $|F(k,x)| = |k| = |x|$
- Choosing a uniform $k \in \{0,1\}^n$ is equivalent to choosing the function $F_k : \{0,1\}^n \to \{0,1\}^n$
  - I.e., for fixed key length n, the algorithm F defines a *distribution* over functions in $Func_n$!

Keyed functions

- Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient, deterministic algorithm
  - Define $F_k(x) = F(k,x)$
  - The first input is called the *key*
- Assume that F is *length preserving*: $F(k,x)$ only defined if $|k| = |x|$, in which case, $|F(k,x)| = |k| = |x|$
- Choosing a uniform $k \in \{0,1\}^n$ is equivalent to choosing the function $F_k : \{0,1\}^n \to \{0,1\}^n$
  - I.e., for fixed key length n, the algorithm F defines a *distribution* over functions in $Func_n$!
  - E.g., $F(k,x) = k$, $F(k,x) = k \oplus x$

PRFs

- The number of functions in $Func_n$ is $2^{n \cdot 2^n}$

PRFs

- The number of functions in $Func_n$ is $2^{n \cdot 2^n}$
- $\{F_k\}_{k \in \{0,1\}^n}$ is a subset of $Func_n$
  - The number of functions in $\{F_k\}_{k \in \{0,1\}^n}$ is at most 2^n

PRFs

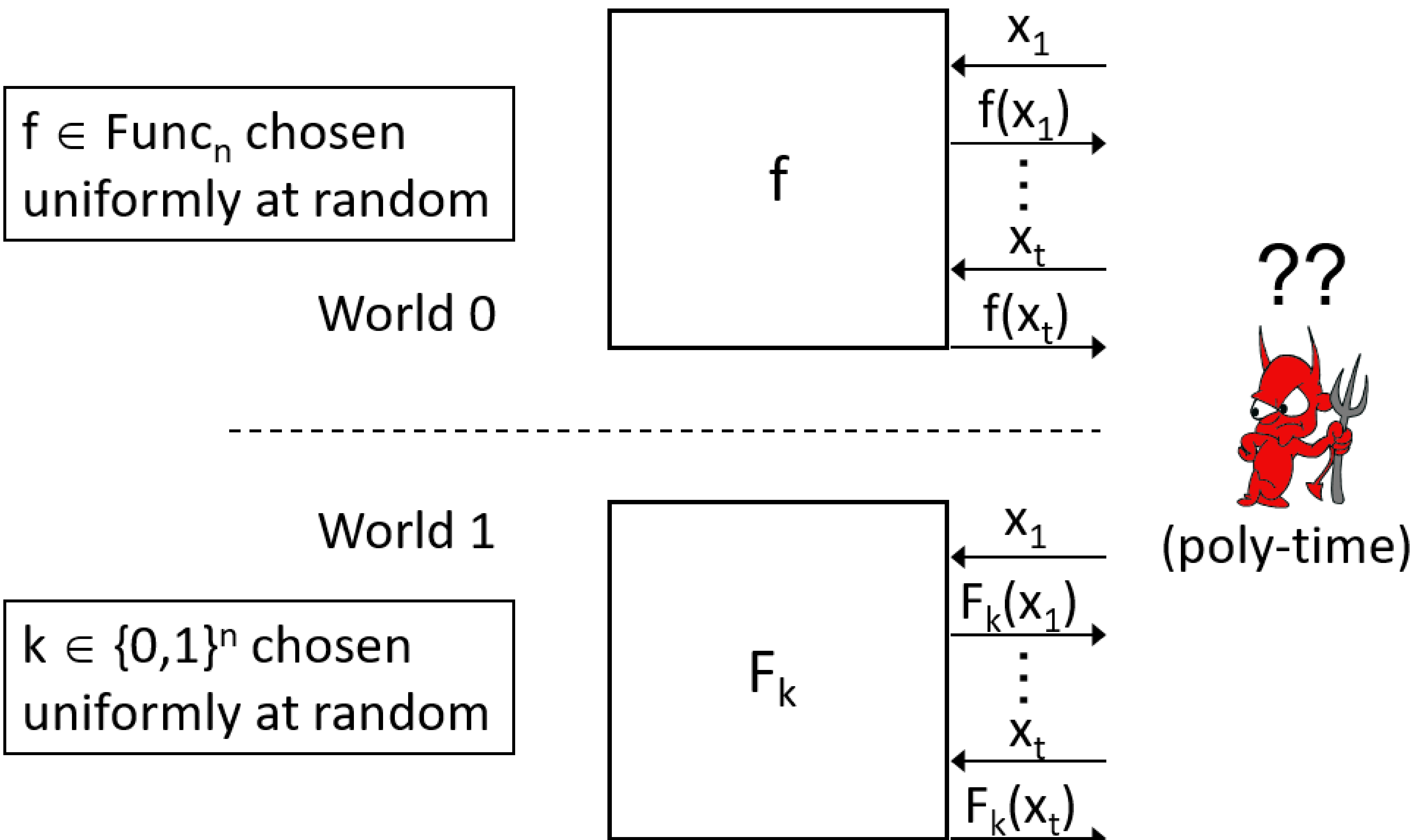
- The number of functions in $Func_n$ is $2^{n \cdot 2^n}$
- $\{F_k\}_{k \in \{0,1\}^n}$ is a subset of $Func_n$
  - The number of functions in $\{F_k\}_{k \in \{0,1\}^n}$ is at most 2^n

Definition 4.2 F is a *pseudorandom function* if F_k, for uniform $k \in \{0,1\}^n$ is indistinguishable from a uniform function $f \in Func_n$

Formally, for all poly-time distinguishers D:

$$\left| \Pr_{k \leftarrow \{0,1\}^n}[D^{F_k(\cdot)}(1^n) = 1] - \Pr_{f \leftarrow Func_n}[D^{f(\cdot)}(1^n) = 1] \right| \leq \epsilon(n)$$

PRFs



$f \in \text{Func}_n$ chosen uniformly at random
World 0
f
x_1
$f(x_1)$
x_t
$f(x_t)$
??
(poly-time)
World 1
F_k
x_1
$F_k(x_1)$
x_t
$F_k(x_t)$
$k \in \{0,1\}^n$ chosen uniformly at random

Pseudorandom permutations (PRPs)

- Let $f \in Func_n$

Pseudorandom permutations (PRPs)

- Let $f \in Func_n$
  f is a *permutation* if it is a bijection
  - This means that the inverse f^{-1} exists

Pseudorandom permutations (PRPs)

- Let $f \in \mathit{Func}_n$
  f is a *permutation* if it is a bijection
  – This means that the inverse f^{-1} exists

- Let $\mathit{Perm}_n \subset \mathit{Func}_n$ be the set of permutations
  – What is $|\mathit{Perm}_n|$?

Pseudorandom permutations

- Let F be a length-preserving, keyed function

Pseudorandom permutations

- Let F be a length-preserving, keyed function
- F is a *keyed permutation* if
  - F_k is a permutation for every k
  - F_k^{-1} is *efficiently computable* (where $F_k^{-1}(F_k(x)) = x$)

Pseudorandom permutations

- Let F be a length-preserving, keyed function
- F is a *keyed permutation* if
  - F_k is a permutation for every k
  - F_k^{-1} is *efficiently computable* (where $F_k^{-1}(F_k(x)) = x$)
- **Definition 4.3** F is a *pseudorandom permutation* if F_k, for uniform key $k \in \{0,1\}^n$, is indistinguishable from a uniform permutation $f \in Perm_n$

Pseudorandom permutations

- Let F be a length-preserving, keyed function
- F is a *keyed permutation* if
  - F_k is a permutation for every k
  - F_k^{-1} is *efficiently computable* (where $F_k^{-1}(F_k(x)) = x$)
- **Definition 4.3** F is a *pseudorandom permutation* if F_k, for uniform key $k \in \{0,1\}^n$, is indistinguishable from a uniform permutation $f \in Perm_n$
- For large enough n, a random permutation is indistinguishable from a random function.
  - In practice, PRPs are also good PRFs

PRFs vs. PRGs

- PRF F immediately implies a PRG G:
  - Define $G(k) = F_k(0\ldots0)|F_k(0\ldots1)$
  - I.e., $G(k) = F_k(\langle 0\rangle)|F_k(\langle 1\rangle)|F_k(\langle 2\rangle)|\ldots$, where $\langle i\rangle$ denotes the n-bit encoding of i

PRFs vs. PRGs

- PRF F immediately implies a PRG G:
  - Define $G(k) = F_k(0\ldots0)|F_k(0\ldots1)$
  - I.e., $G(k) = F_k(\langle 0\rangle)|F_k(\langle 1\rangle)|F_k(\langle 2\rangle)|\ldots$, where $\langle i\rangle$ denotes the n-bit encoding of i

- PRF can be viewed as a PRG with random access to exponentially long output
  - The function F_k can be viewed as the $n2^n$-bit string $F_k(0\ldots0)|\ldots|F_k(1\ldots1)$

Do PRFs/PRPs exist?

- They are a stronger primitive than PRGs
  - though can be built from PRGs

Do PRFs/PRPs exist?

- They are a stronger primitive than PRGs
  – though can be built from PRGs

Theorem (Goldreich, Goldwasser, Micali 1984)
If the PRG Axiom is true, then there exist PRFs.

How to Construct Random Functions

ODED GOLDREICH, SHAFI GOLDWASSER,
AND SILVIO MICALI

Massachusetts Institute of Technology, Cambridge, Massachusetts

Abstract. A constructive theory of randomness for functions, based on computational complexity, is developed, and a pseudorandom function generator is presented. This generator is a deterministic polynomial-time algorithm that transforms pairs (g, r), where g is *any* one-way function and r is a random k-bit string, to polynomial-time computable functions f_r: $\{1, \ldots, 2^k\} \rightarrow \{1, \ldots, 2^k\}$. These f_r's cannot be distinguished from *random* functions by any probabilistic polynomial-time algorithm that asks and receives the value of a function at arguments of its choice. The result has applications in cryptography, random constructions, and complexity theory.

Categories and Subject Descriptors: F.0 [**Theory of Computation**]: General; F.1.1 [**Computation by Abstract Devices**]: Models of Computation—*computability theory*; G.0 [**Mathematics of Computing**]: General; G.3 [**Mathematics of Computing**]: Probability and Statistics—*probabilistic algorithms*; *random number generation*

General Terms: Algorithms, Security, Theory

Additional Key Words and Phrases: Cryptography, one-way functions, prediction problems, randomness

> *I have set up on a Manchester computer a small programme using only 1000 units of storage, whereby the machine supplied with one sixteen figure number replies with another within two seconds. I would defy anyone to learn from these replies sufficient about the programme to be able to predict any replies to untried values.*
>
> A. TURING

Do PRFs/PRPs exist?

- They are a stronger primitive than PRGs
  – though can be built from PRGs

- In practice, block ciphers are used

Next Lecture

- block cipher ...