CSE5014 CRYPTOGRAPHY AND NETWORK SECURITY

Dr. QI WANG

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

PRGs

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

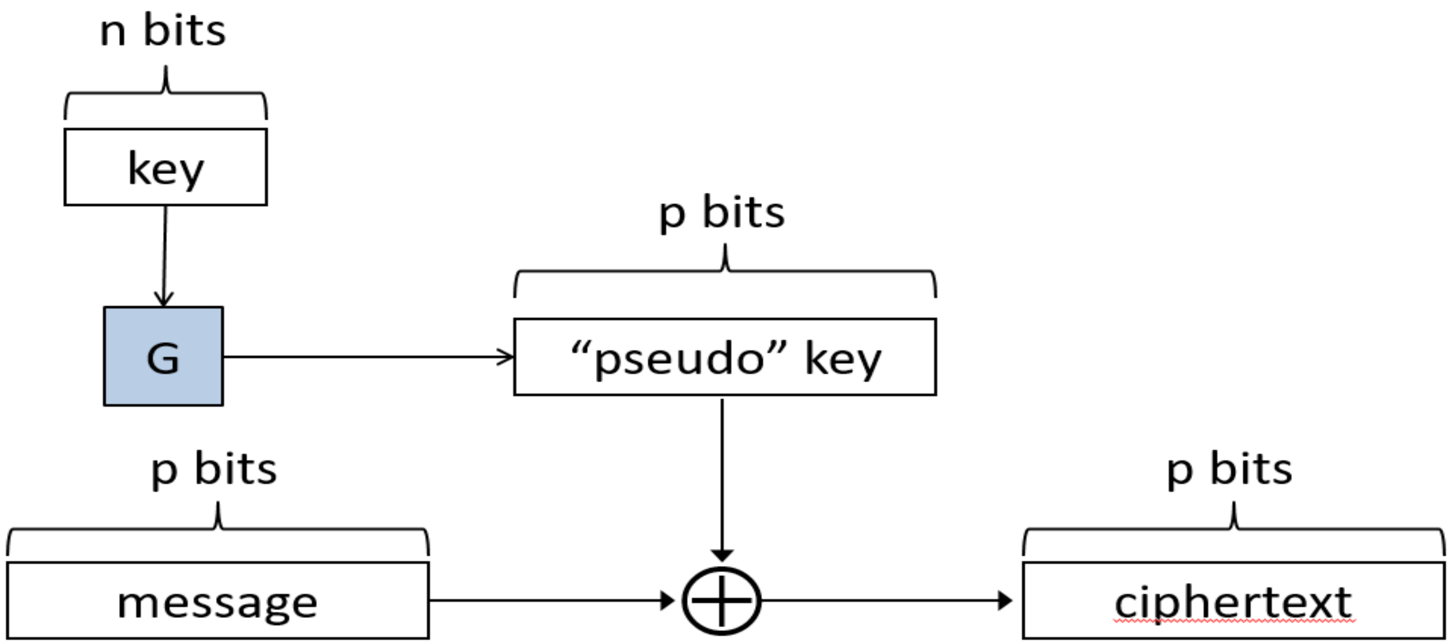
  Let G be a deterministic, poly-time algorithm that is *expanding*, i.e., $|G(x)| = p(|x|) > |x|$.

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

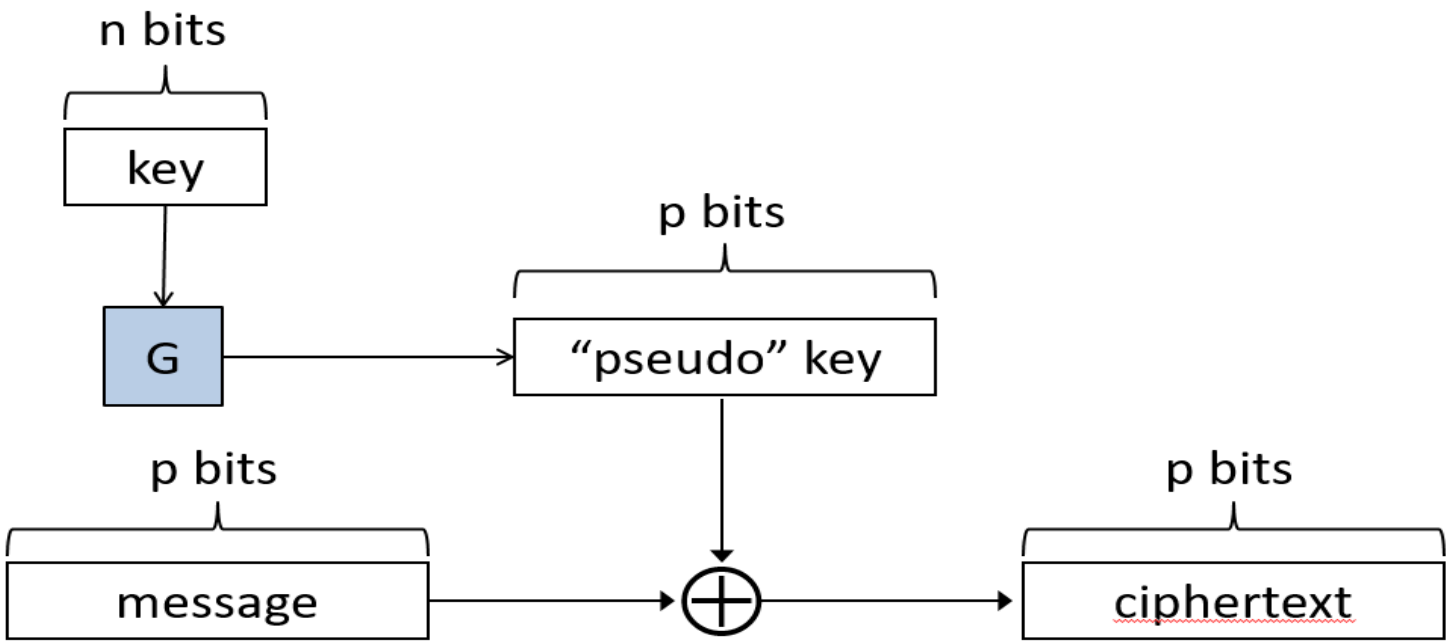
  $$|\Pr_{x \leftarrow U_n}[A(G(x)) = 1] - \Pr_{y \leftarrow U_{p(n)}}[A(y) = 1]| \leq \epsilon(n)$$

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

Pseudo one-time pad



Pseudo one-time pad



- **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*)

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

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

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

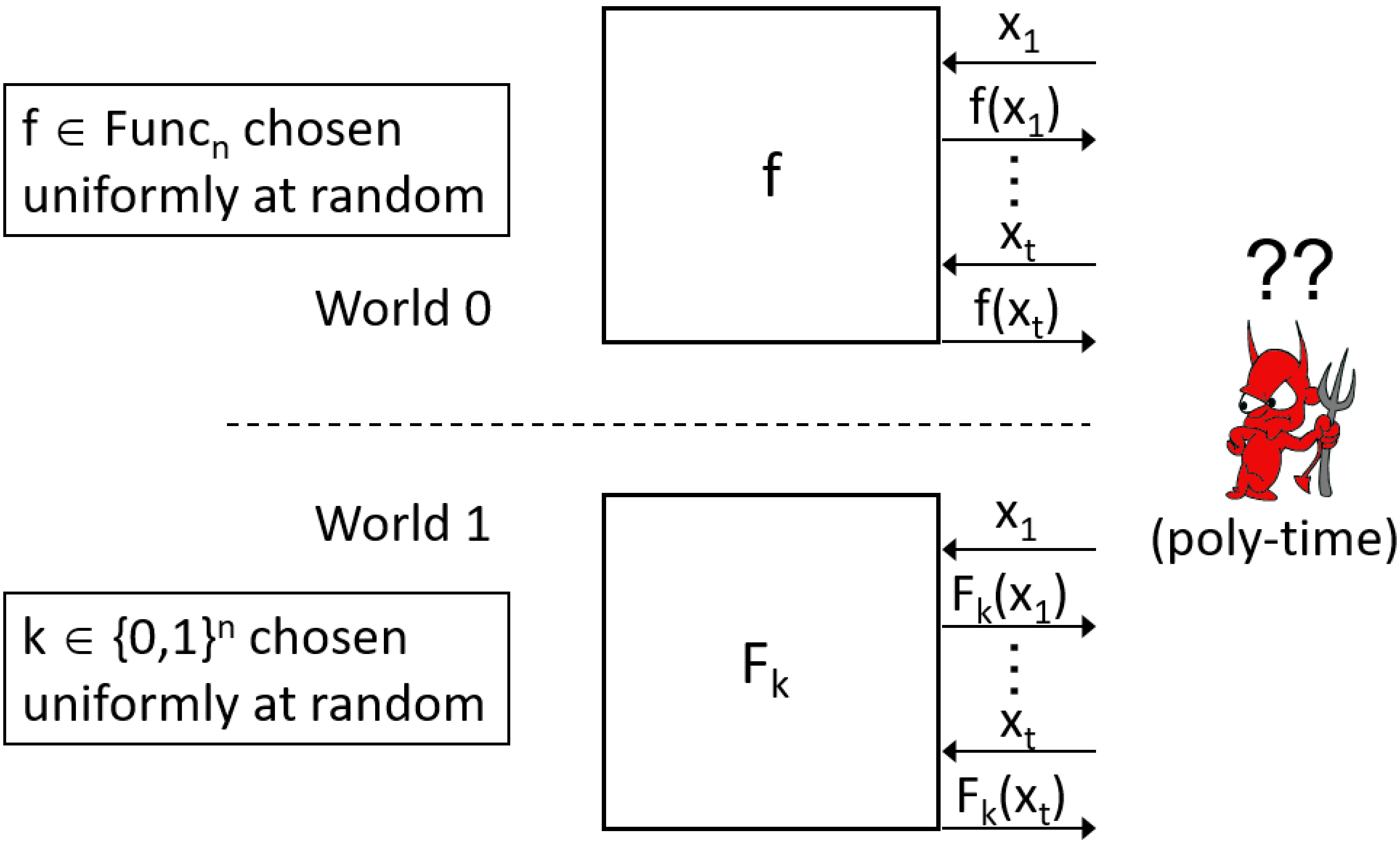
- The number of functions in $Func_n$ is $2^{n \cdot 2^n}$
- $\{F_k\}_{k \in \{0,1\}^n}$ is a subset of $Func_n$
  - The number of functions in $\{F_k\}_{k \in \{0,1\}^n}$ is at most 2^n

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



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

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

- Let $Perm_n \subset Func_n$ be the set of permutations
  – What is $|Perm_n|$?

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

Block ciphers

- Block ciphers are practical constructions of *pseudorandom permutations* (PRPs)

Block ciphers

- Block ciphers are practical constructions of *pseudorandom permutations* (PRPs)

  $F : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$
  - $n =$ "key length"
  - $m =$ "block length"

Block ciphers

- Block ciphers are practical constructions of *pseudorandom permutations* (PRPs)

  $F: \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^m$
  - $n =$ "key length"
  - $m =$ "block length"

  Hard to distinguish F_k from uniform $f \in Perm_m$

AES

- *Advanced encryption standard* (AES)
  - Standardized by NIST in 2000 based on a public, worldwide competation lasting over 3 years
  - Block length $= 128$ bits
  - Key length $= 128, 192$ or 256 bits

AES

- *Advanced encryption standard* (AES)
  - Standardized by NIST in 2000 based on a public, worldwide competation lasting over 3 years
  - Block length $= 128$ bits
  - Key length $= 128, 192$ or 256 bits

- Will discuss details later in the course
  - Rijndael named after Vincent Rijmen and Joan Daemen

History of Block Cipher

- 1972: NIST (then NBS) called for encryption standard proposals

  1976: IBM responded: “Lucifer”

  NSA tweaked Lucifer to get *Data Encryption Standard* (DES) and approved it

  The key length is “short”: 56 bits

  By late 90’s, most commercial applications used 3DES: three applications of DES with independent keys

  It had been used as a standard for encryption until 2000. DES was subject to exhaustive key search attacks.

History of Block Cipher

- 1997: NIST issued call for new ciphers (use for $\geq$ 30 years, protect $\geq$ 100 years)

  1998: 15 candidates accepted in June

  1999: 5 of them were shortlisted in August

  2000: *Rijndael* was selected as the *AES* in October (Daeman, Rijmen)

  2001: issued as FIPS PUB 197 standard in November

History of Block Cipher

- 1997: NIST issued call for new ciphers (use for $\geq$ 30 years, protect $\geq$ 100 years)

  1998: 15 candidates accepted in June

  1999: 5 of them were shortlisted in August

  2000: *Rijndael* was selected as the *AES* in October (Daeman, Rijmen)

  2001: issued as FIPS PUB 197 standard in November

History of Block Cipher

- 1997: NIST issued call for new ciphers (use for $\geq$ 30 years, protect $\geq$ 100 years)

  1998: 15 candidates accepted in June

  1999: 5 of them were shortlisted in August

  2000: *Rijndael* was selected as the *AES* in October (Daeman, Rijmen)

  2001: issued as FIPS PUB 197 standard in November

  Block length: 128 bits, key length: 128/192/256 bits

  Stronger and faster than 3DES

  Efficient in both software and hardware

  Simple in design, suitable for smard cards (memory requirement)

CPA-secure encryption

- Let F be a length-preserving, keyed function

CPA-secure encryption

- Let F be a length-preserving, keyed function

  $Gen(1^n)$: choose a uniform key $k \in \{0,1\}^n$

CPA-secure encryption

- Let F be a length-preserving, keyed function

  $Gen(1^n)$: choose a uniform key $k \in \{0,1\}^n$

  $Enc_k(m)$, for $|m| = |k|$
  - Choose uniform $r \in \{0,1\}^n$ (*nonce/ initialization vector*)
  - Output ciphertext $\langle r, F_k(r) \oplus m \rangle$

CPA-secure encryption

- Let F be a length-preserving, keyed function

  $Gen(1^n)$: choose a uniform key $k \in \{0,1\}^n$

  $Enc_k(m)$, for $|m| = |k|$

  – Choose uniform $r \in \{0,1\}^n$ (*nonce/ initialization vector*)

  – Output ciphertext $\langle r, F_k(r) \oplus m \rangle$

  $Dec_k(c_1, c_2)$: output $c_2 \oplus F_k(c_1)$

CPA-secure encryption

- Let F be a length-preserving, keyed function

  $Gen(1^n)$: choose a uniform key $k \in \{0,1\}^n$

  $Enc_k(m)$, for $|m| = |k|$

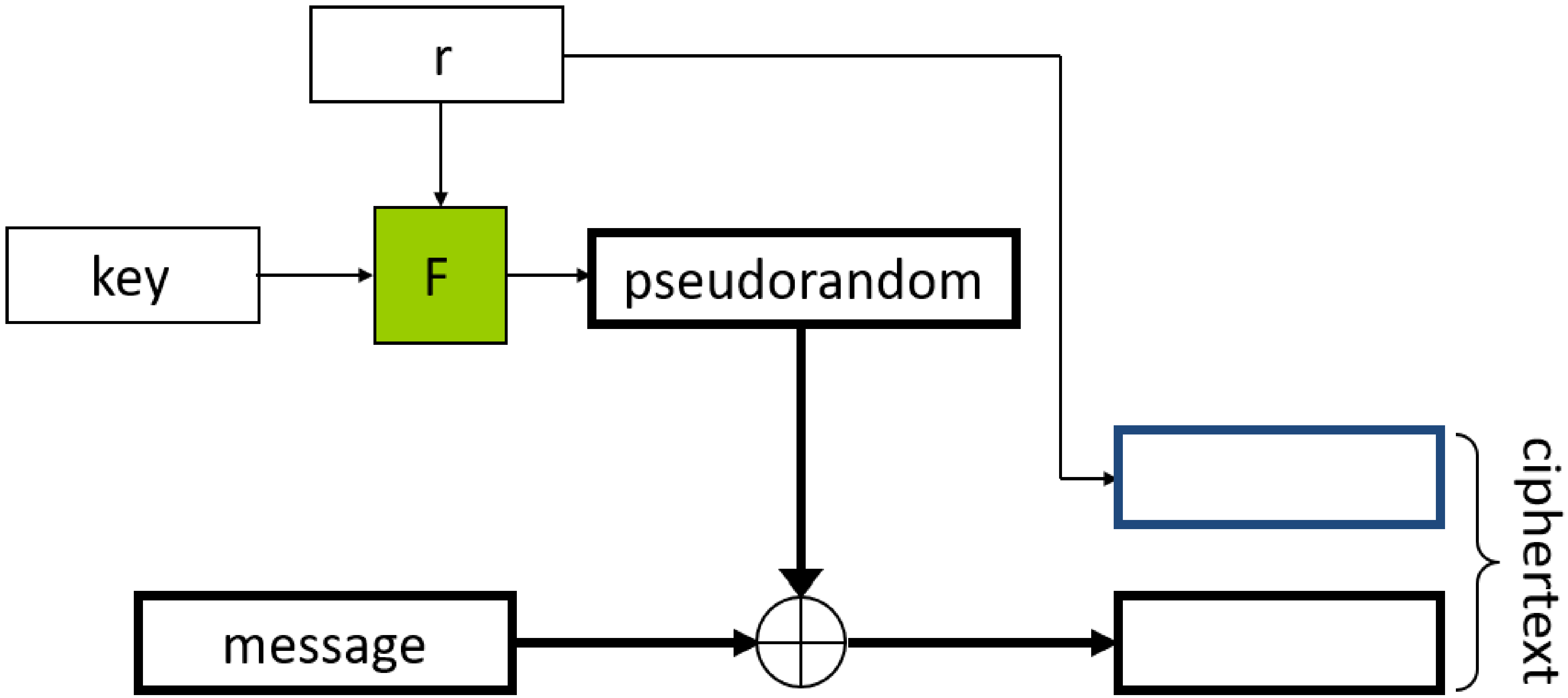
  – Choose uniform $r \in \{0,1\}^n$ (*nonce/ initialization vector*)

  – Output ciphertext $\langle r, F_k(r) \oplus m \rangle$

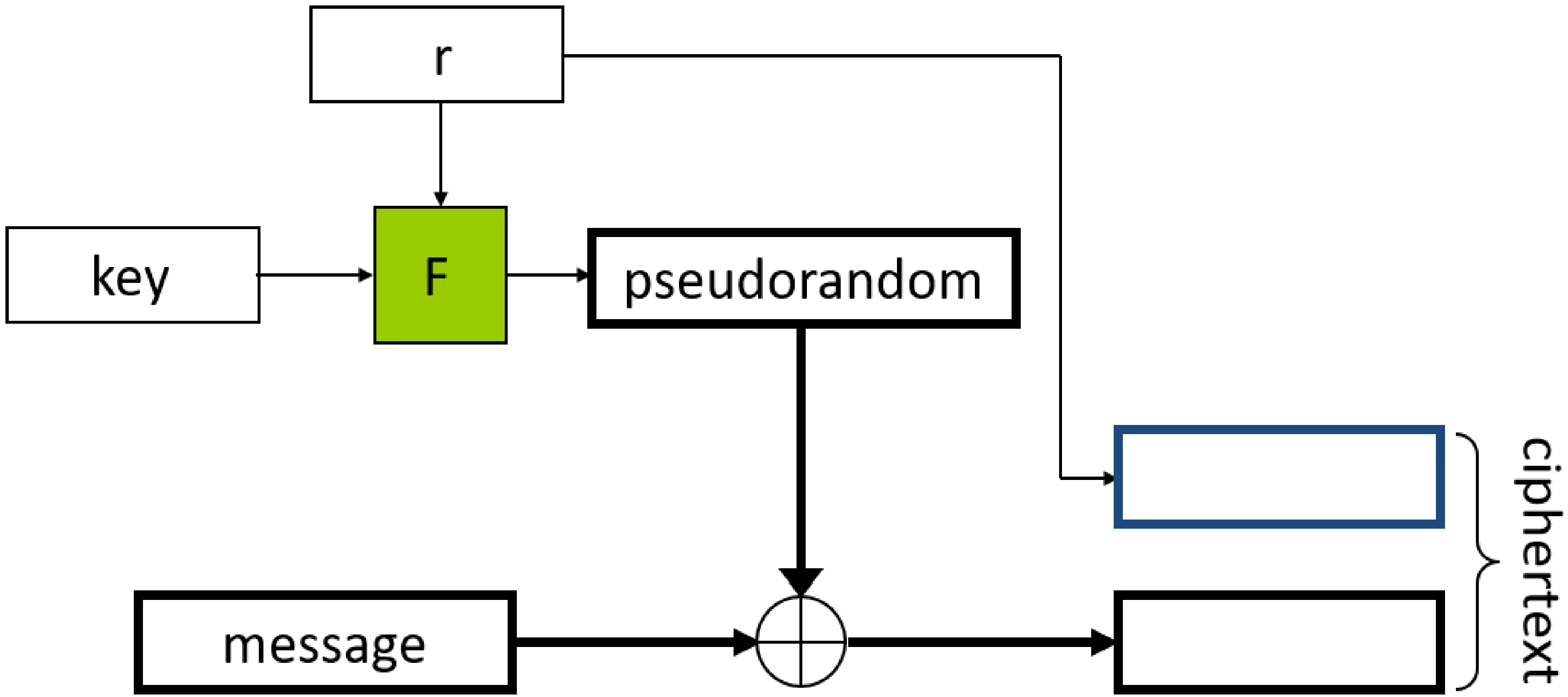
  $Dec_k(c_1, c_2)$: output $c_2 \oplus F_k(c_1)$

  Correctness is immediate

CPA-secure encryption



CPA-secure encryption



Theorem 5.1 If F is a pseudorandom function, then this scheme is *CPA-secure*.

Note

- The key may be as long as the message
- But the same key can be used to safely encrypt *multiple* messages

Security?

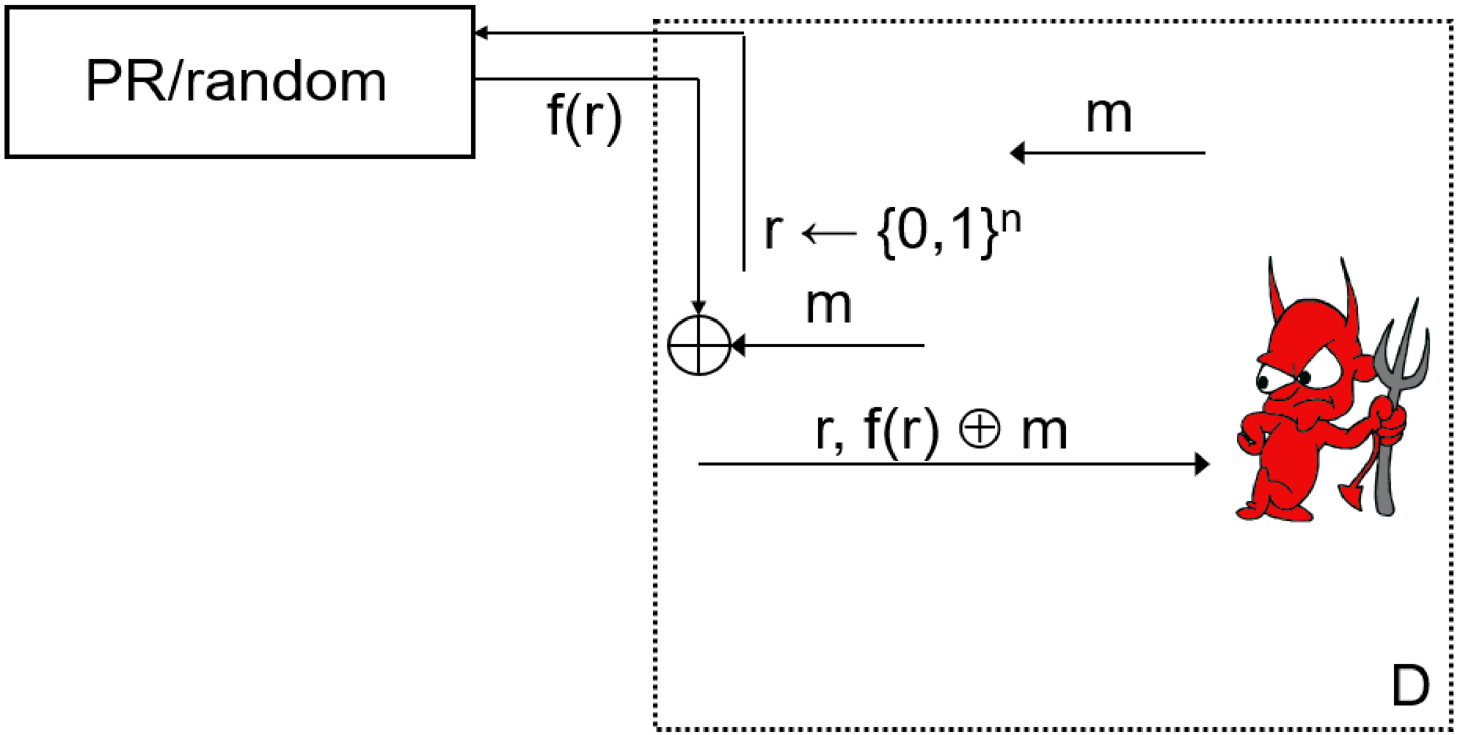
- **Theorem 5.1** If F is a pseudorandom function, then this scheme is *CPA-secure*.

  Proof by reduction
  Let Π denote the scheme

Security?

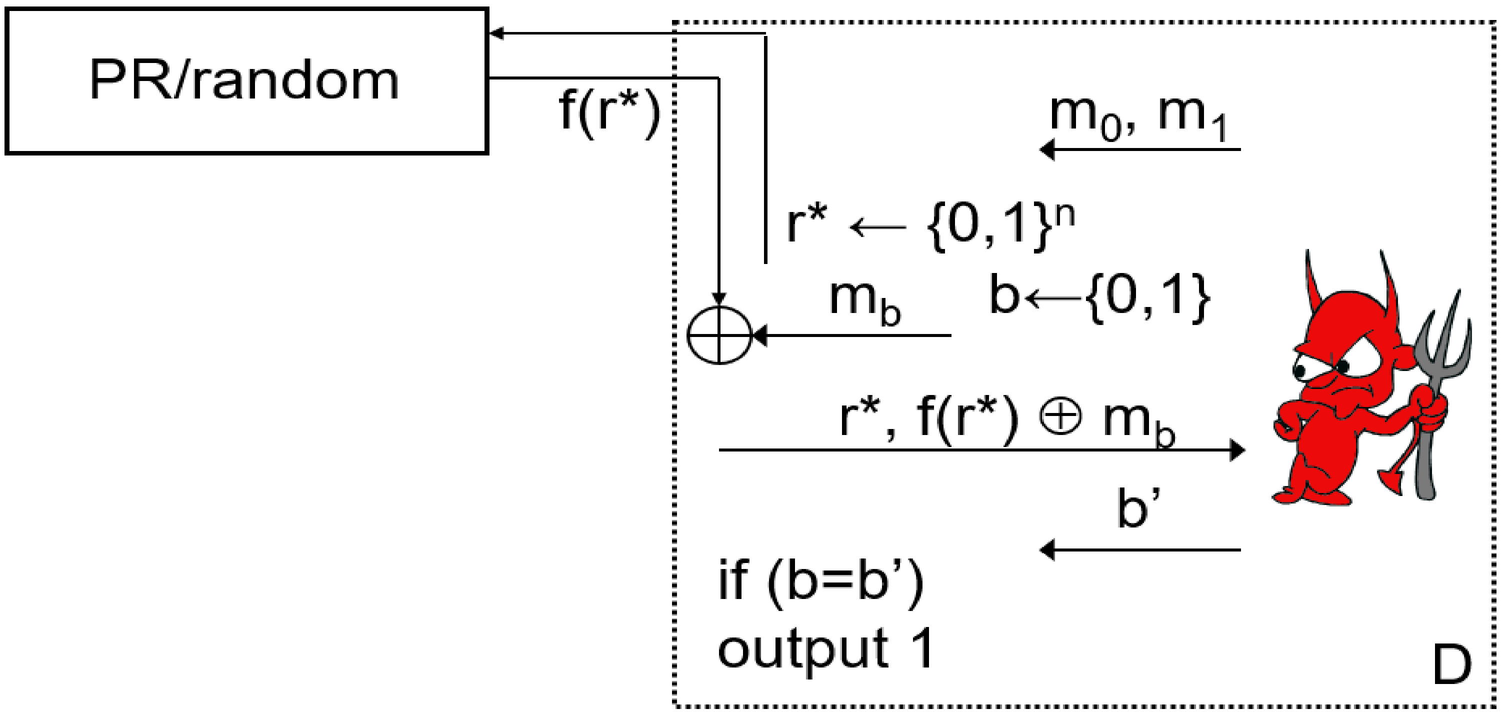
- **Theorem 5.1** If F is a pseudorandom function, then this scheme is *CPA-secure*.

  Proof by reduction
  Let Π denote the scheme



- **Theorem 5.1** If F is a pseudorandom function, then this scheme is *CPA-secure*.

  Proof by reduction
  Let Π denote the scheme



Analysis

- Let $\mu(n) = \Pr[PrivCPA_{Adv,\Pi}(n) = 1]$
  Let $q(n)$ be a bound on the number of encryption queries made by attacker

Analysis

- Let $\mu(n) = \Pr[PrivCPA_{Adv,\Pi}(n) = 1]$
  Let $q(n)$ be a bound on the number of encryption queries made by attacker

  If $f = F_k$ for uniform k, then the view of Adv is exactly as in $PrivCPA_{Adv,\Pi}(n)$

  $\Rightarrow$

  $\Pr_{k \leftarrow \{0,1\}^n}[D^{F_k(\cdot)} = 1] = \Pr[PrivCPA_{Adv,\Pi}(n) = 1] = \mu(n)$

Analysis

- If f is uniform, there are two subcases
  - r^* was used for some other ciphertext (call this event *Repeat*)
  - r^* was not used for some other ciphertext

Analysis

- If f is uniform, there are two subcases
  - r^* was used for some other ciphertext (call this event *Repeat*)
  - r^* was not used for some other ciphertext
- $\Pr_f[D^{f(\cdot)} = 1] \leq \Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] + \Pr[Repeat]$
  - $\Pr[Repeat] \leq q(n)/2^n$
  - $\Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] = 1/2$

Analysis

- If f is uniform, there are two subcases
  - r^* was used for some other ciphertext (call this event *Repeat*)
  - r^* was not used for some other ciphertext
- $\Pr_f[D^{f(\cdot)} = 1] \leq \Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] + \Pr[Repeat]$
  - $\Pr[Repeat] \leq q(n)/2^n$
  - $\Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] = 1/2$
- Since F is pseudorandom

  $\Rightarrow |\mu(n) - \Pr_f[D^{f(\cdot)} = 1]| \leq \epsilon(n)$

Analysis

- If f is uniform, there are two subcases
  - r^* was used for some other ciphertext (call this event *Repeat*)
  - r^* was not used for some other ciphertext
- $\Pr_f[D^{f(\cdot)} = 1] \leq \Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] + \Pr[Repeat]$
  - $\Pr[Repeat] \leq q(n)/2^n$
  - $\Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] = 1/2$
- Since F is pseudorandom

$$\Rightarrow |\mu(n) - \Pr_f[D^{f(\cdot)} = 1]| \leq \epsilon(n)$$

$$\Rightarrow \mu(n) \leq \Pr_f[D^{f(\cdot)} = 1] + \epsilon(n) \leq 1/2 + q(n)/2^n + \epsilon(n)$$

Analysis

- If f is uniform, there are two subcases
  - r^* was used for some other ciphertext (call this event *Repeat*)
  - r^* was not used for some other ciphertext
- $\Pr_f[D^{f(\cdot)} = 1] \leq \Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] + \Pr[Repeat]$
  - $\Pr[Repeat] \leq q(n)/2^n$
  - $\Pr_f[D^{f(\cdot)} = 1 | \neg Repeat] = 1/2$
- Since F is pseudorandom

  $\Rightarrow |\mu(n) - \Pr_f[D^{f(\cdot)} = 1]| \leq \epsilon(n)$

  $\Rightarrow \mu(n) \leq \Pr_f[D^{f(\cdot)} = 1] + \epsilon(n) \leq 1/2 + q(n)/2^n + \epsilon(n)$

  Note: $q(n)/2^n + \epsilon(n) = \epsilon'(n)$ is negligible

Real-world security?

- The security bound we proved is *tight*

Real-world security?

- The security bound we proved is *tight*
- What happens if a nonce r is ever reused?
- What is the probability that the nonce used in some challenge ciphertext is also used for some other ciphertext?
- What happens to the bound if the nonce is chosen *non-uniformly*?

CPA-secure encryption

- We have shown a *CPA-secure* encryption scheme based on any block cipher/ PRF
  - $Enc_k(m) = \langle r, F_k(r) \oplus m \rangle$

CPA-secure encryption

- We have shown a *CPA-secure* encryption scheme based on any block cipher/ PRF
  - $Enc_k(m) = \langle r, F_k(r) \oplus m \rangle$

- Drawbacks?
  - A 1-block plaintext results in a 2-block ciphertext
  - Only defined for encryption of n-bit messages

Encrypting long messages?

- Recall that CPA-security $\Rightarrow$ security for the encryption of multiple messages

Encrypting long messages?

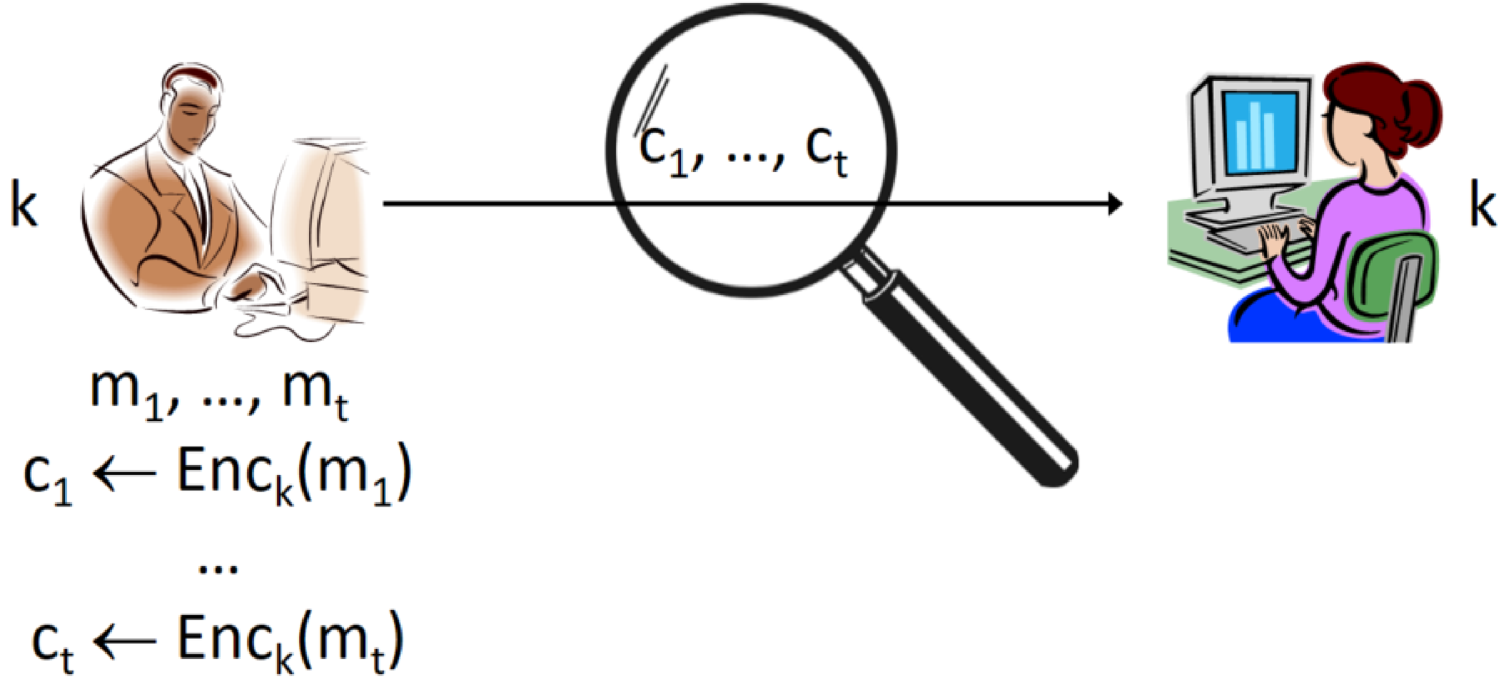
- Recall that CPA-security $\Rightarrow$ security for the encryption of multiple messages

- So, can encrypt the message $m_1, \ldots, m_t$ as $Enc_k(m_1), Enc_k(m_2), \ldots, Enc_k(m_t)$
  – This is also *CPA-secure*!

Encrypting long messages?

- Recall that CPA-security $\Rightarrow$ security for the encryption of multiple messages

- So, can encrypt the message $m_1, \ldots, m_t$ as $Enc_k(m_1), Enc_k(m_2), \ldots, Enc_k(m_t)$
  - This is also *CPA-secure*!



Drawback

- The ciphertext is *twice* the length of the plaintext
  - I.e., ciphertext expansion by a factor of two

Drawback

- The ciphertext is *twice* the length of the plaintext
  – I.e., ciphertext expansion by a factor of two

- Can we do better?

Drawback

- The ciphertext is *twice* the length of the plaintext
  - I.e., ciphertext expansion by a factor of two

- Can we do better?

- Modes of operation
  - *Block-cipher* modes of operation
  - *Stream-cipher* modes of operation

CTR (Counter) mode

- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $ctr \leftarrow_R \{0,1\}^n$, set $c_0 = ctr$
  - For $i = 1$ to t:
    - $c_i = m_i \oplus F_k(ctr + i)$
  - Output $c_0, c_1, \ldots, c_t$

CTR (Counter) mode

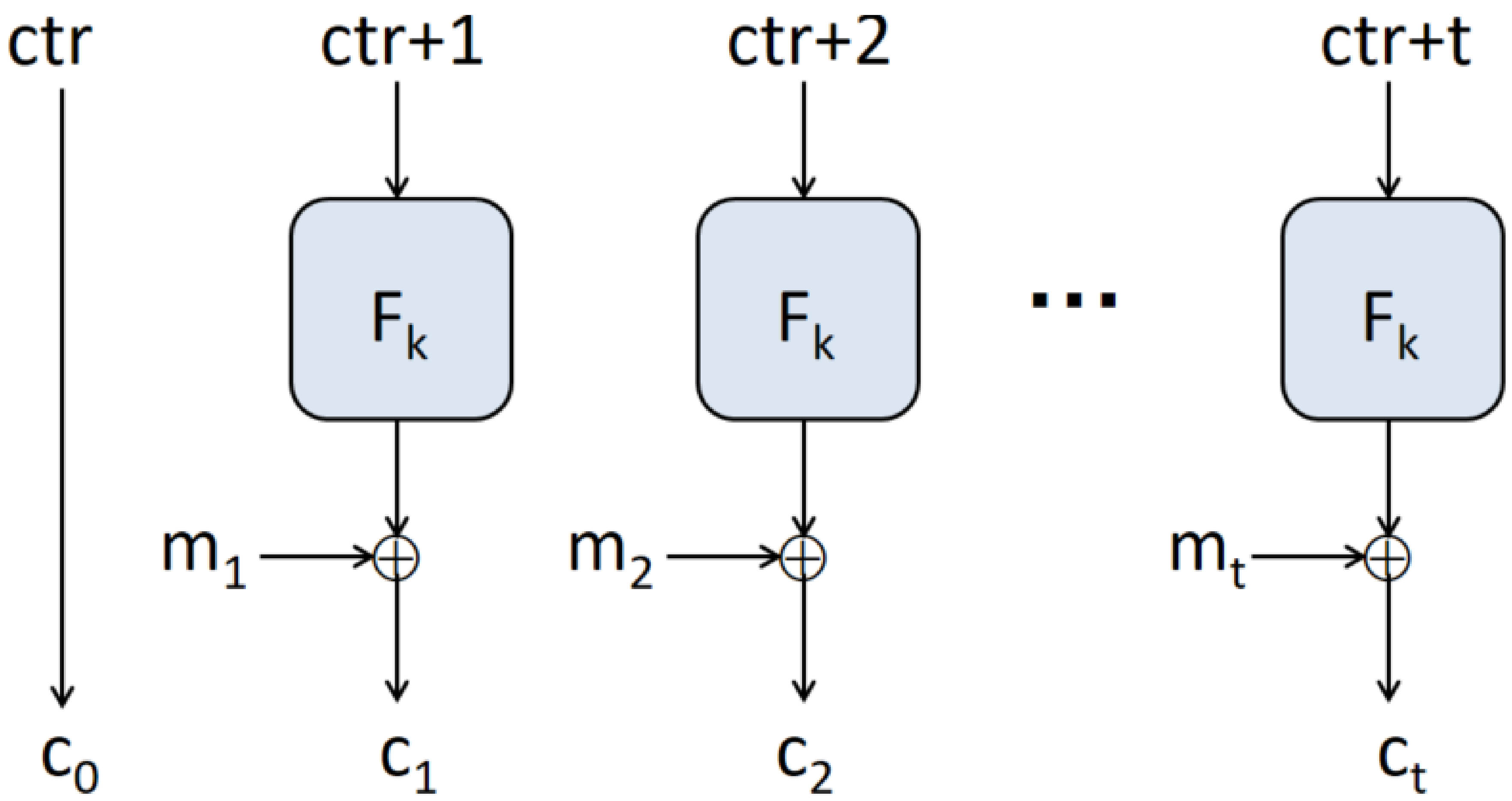
- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $ctr \leftarrow_R \{0,1\}^n$, set $c_0 = ctr$
  - For $i = 1$ to t:
    - $c_i = m_i \oplus F_k(ctr + i)$
  - Output $c_0, c_1, \ldots, c_t$

- Decryption?

CTR (Counter) mode

- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $ctr \leftarrow_R \{0,1\}^n$, set $c_0 = ctr$
  - For $i = 1$ to t:
    - $c_i = m_i \oplus F_k(ctr + i)$
  - Output $c_0, c_1, \ldots, c_t$
- Decryption?
- Ciphertext expansion is just 1 block

CTR mode



CTR mode

- **Theorem 5.2** If F is a pseudorandom function, then CTR mode is *CPA-secure*.

CTR mode

- **Theorem 5.2** If F is a pseudorandom function, then CTR mode is *CPA-secure*.

- **Proof** sketch:
  The sequences $F_k(ctr_i + 1), \ldots, F_k(ctr_i + t)$ used to encrypt the i-th message is pseudorandom

CTR mode

- **Theorem 5.2** If F is a pseudorandom function, then CTR mode is *CPA-secure*.

- **Proof** sketch:
  The sequences $F_k(ctr_i + 1), \ldots, F_k(ctr_i + t)$ used to encrypt the i-th message is pseudorandom
  - Moreover, it is independent of every other such sequence unless $ctr_i + j = ctr_{i'} + j'$ for some i, j, i', j'
  - Just need to bound the probability of that event

CBC (Cipher Block Chaining) mode

- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $c_0 \leftarrow_R \{0,1\}^n$ (also called the IV)
  - For $i = 1$ to t:
    - $c_i = F_k(m_i \oplus c_{i-1})$
  - Output $c_0, c_1, \ldots, c_t$

CBC (Cipher Block Chaining) mode

- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $c_0 \leftarrow_R \{0,1\}^n$ (also called the *IV*)
  - For $i = 1$ to t:
    - $c_i = F_k(m_i \oplus c_{i-1})$
  - Output $c_0, c_1, \ldots, c_t$

- Decryption?
  - Requires F to be *invertible*

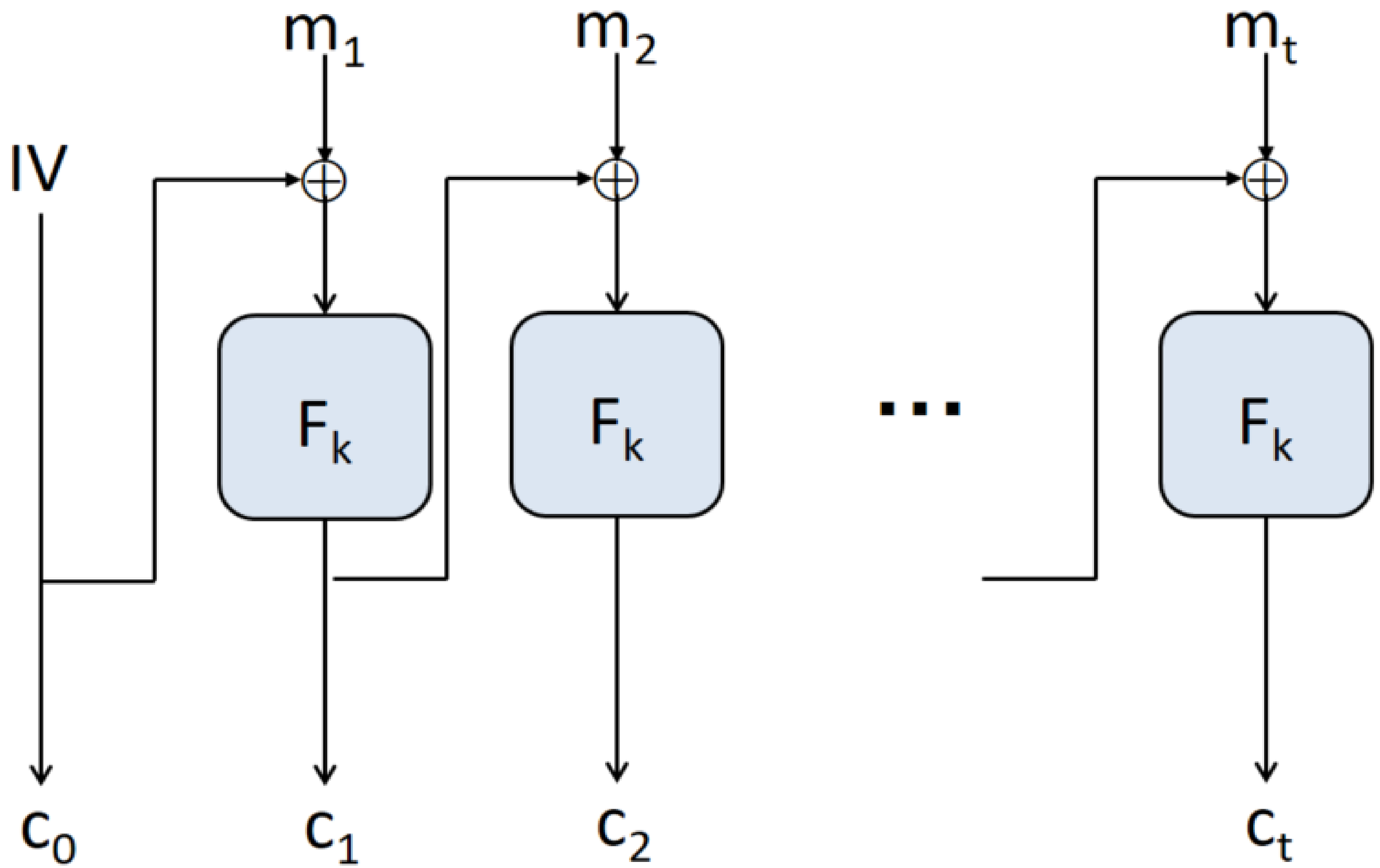
CBC (Cipher Block Chaining) mode

- $Enc_k(m_1, \ldots, m_t)$ // note: t is arbitrary
  - Choose $c_0 \leftarrow_R \{0,1\}^n$ (also called the *IV*)
  - For $i = 1$ to t:
    - $c_i = F_k(m_i \oplus c_{i-1})$
  - Output $c_0, c_1, \ldots, c_t$

- Decryption?
  - Requires F to be *invertible*

- Ciphertext expansion is just 1 block

CBC mode



CBC mode

- **Theorem 5.3** If F is a pseudorandom function, then CBC mode is *CPA-secure*.

CBC mode

- **Theorem 5.3** If F is a pseudorandom function, then CBC mode is *CPA-secure*.

- **Proof** is more complicated than for CTR mode

ECB (Electronic Codebook) mode

- $Enc_k(m_1, \ldots, m_t) = F_k(m_1), \ldots, F_k(m_t)$

ECB (Electronic Codebook) mode

- $Enc_k(m_1, \ldots, m_t) = F_k(m_1), \ldots, F_k(m_t)$
- Deterministic
  - Not CPA-secure!
  - Efficient: online computation

ECB (Electronic Codebook) mode

- $Enc_k(m_1, \ldots, m_t) = F_k(m_1), \ldots, F_k(m_t)$
- Deterministic
  - Not CPA-secure!
  - Efficient: online computation
- Can tell from the ciphertext whether $m_i = m_j$
  - Not even EAV-secure!

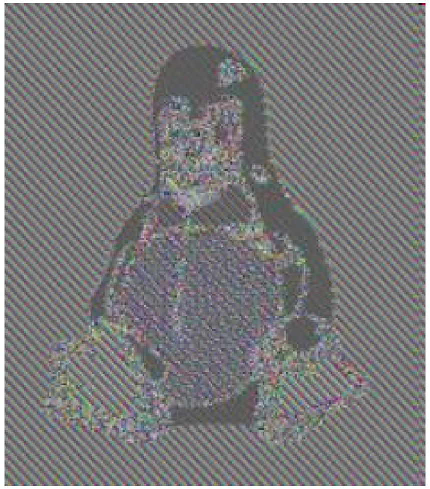
ECB (Electronic Codebook) mode

- $Enc_k(m_1, \ldots, m_t) = F_k(m_1), \ldots, F_k(m_t)$

- Deterministic
  - Not CPA-secure!
  - Efficient: online computation

- Can tell from the ciphertext whether $m_i = m_j$
  - Not even EAV-secure!

original



encrypted using ECB mode

Stream ciphers

- As we defined, PRGs are *limited*
  - They have fixed-length output
  - They produce output in "one shot"

- In practice, PRGs are based on *stream ciphers*
  - Can be viewed as producing an "infinite" stream of pseudorandom bits, on demand
  - More flexible, more efficient

Stream ciphers

- Pair of efficient, deterministic algorithms (Init, GetBits)

Stream ciphers

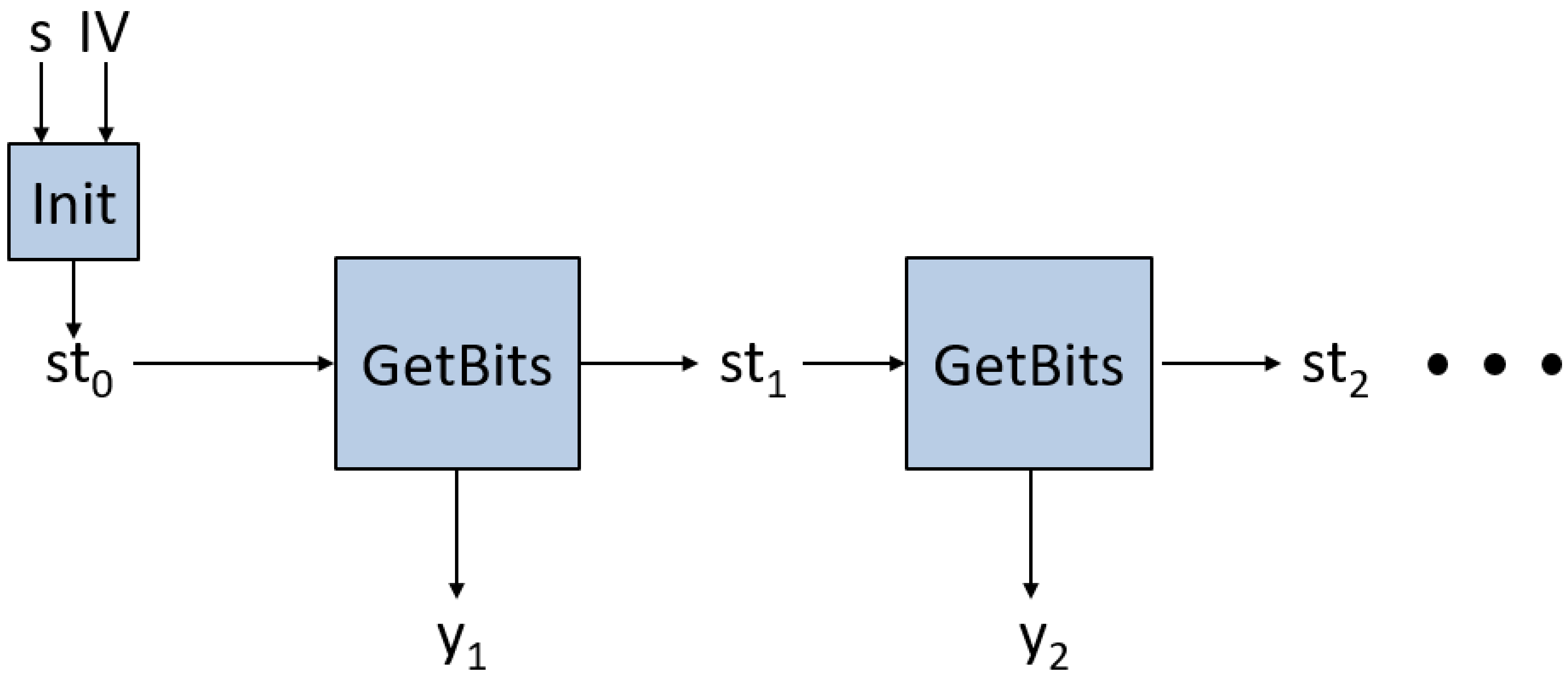
- Pair of efficient, deterministic algorithms (Init, GetBits)
  - Init takes a seed s_0 (and optional IV), and outputs initial state st_0
  - GetBits takes the current state st and outputs a bit y along with updated state st'

Stream ciphers

- Pair of efficient, deterministic algorithms (Init, GetBits)
  - Init takes a seed s_0 (and optional IV), and outputs initial state st_0
  - GetBits takes the current state st and outputs a bit y along with updated state st'
    - In practice, y would be a block rather than a bit

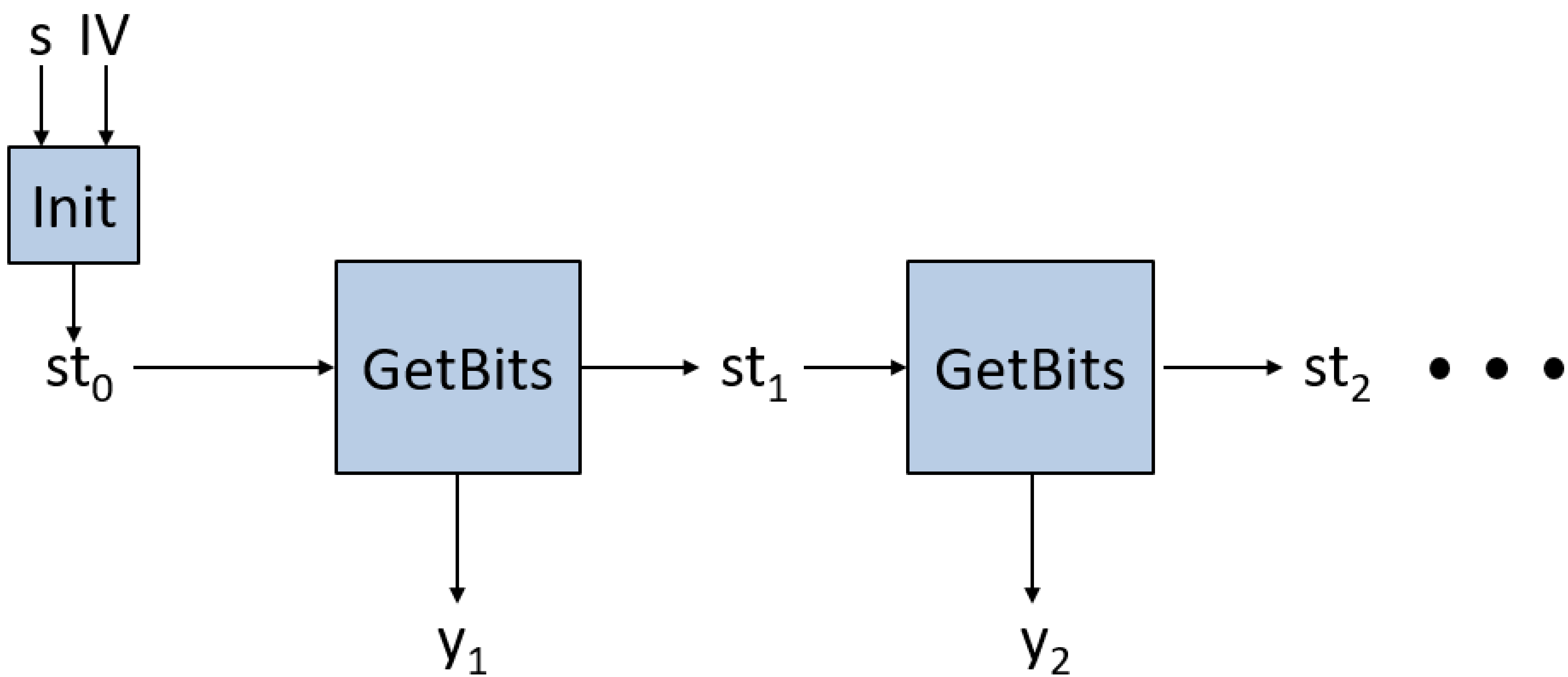
Stream ciphers

- Can use (Init, GetBits) to generate any desired number of output bits from an initial seed



Stream ciphers

- A *stream cipher* is *secure* (informally) if the output stream generated from a uniform seed is pseudorandom
  - I.e., regardless of how long the output stream is (so long as it is polynomial)



Next Lecture

- stream cipher, CCA security ...