

# Introduction to Cryptography

## Lecture 2

Benny Pinkas

# Perfect Cipher

- What type of security would we like to achieve?
- In an “ideal” world, the message will be delivered in a magical way, out of the reach of the adversary
  - An encryption system will therefore be called *secure* if no adversary can learn any partial information about the plaintext from the ciphertext.
- Definition: a *perfect cipher*
  - $Pr(\text{plaintext} = P \mid \text{ciphertext} = C) = Pr(\text{plaintext} = P)$
  - The ciphertext does not reveal any information about the plaintext
  - Sometimes called “*semantic security*”.

- “Perfect cipher” is a definition of a security property
- In the previous lecture, we saw an example of a perfect cipher, the one-time pad.
- When we want to discuss or prove general properties of perfect ciphers, we must refer to every encryption scheme that satisfies the definition.
  - Not only the one-time pad.

# Perfect Ciphers

- A simple criteria for perfect ciphers. :
- The cipher is perfect if, and only if,  
 $\forall m_1, m_2 \in M, \forall \text{cipher } c,$   
 $\Pr(\text{Enc}(m_1)=c) = \Pr(\text{Enc}(m_2)=c).$   
(let's prove it)
- This criterion is called *"indistinguishability"*.
- Idea: Regardless of the plaintext, the adversary sees the same distribution of ciphertexts and cannot distinguish between encryptions of different plaintexts.

# Proof

- Note that the proof cannot assume that the cipher is the one-time-pad
- We can only assume that  $Pr(\textit{plaintext} = P \mid \textit{ciphertext} = C) = Pr(\textit{plaintext} = P)$

## Proof (of one direction)

- Perfect security:
  - $\forall m \in M, \forall \text{cipher } c, \Pr(\text{plaintext}=m / \text{ciphertext}=c) = \Pr(\text{plaintext}=m)$ .
- Indistinguishability criterion:
  - $\forall m_1, m_2 \in M, \forall \text{cipher } c, \Pr(\text{Enc}(m_1)=c) = \Pr(\text{Enc}(m_2)=c)$ .
- Perfect security  $\Rightarrow$  Indistinguishability criterion
$$\begin{aligned} \Pr(\text{Enc}(m_1)=c) &= \Pr(\text{ciphertext}=c / \text{plaintext}=m_1) \\ &= \Pr(\text{ciphertext}=c \text{ and } \text{plaintext}=m_1) / \Pr(\text{plaintext}=m_1) \\ &= \Pr(\text{plaintext}=m_1 / \text{ciphertext}=c) \cdot \Pr(\text{ciphertext}=c) / \\ &\quad \Pr(\text{plaintext}=m_1) \\ &= 1 \cdot \Pr(\text{ciphertext}=c) / 1 = \Pr(\text{ciphertext}=c) \end{aligned}$$

## Size of key space

- Perfect security holds even against an adversary that has unlimited computational powers. It is also called “information theoretic security” or “unconditional security”.
- However, the key size is inefficient.
- Theorem: For a perfect encryption scheme, the number of possible keys is at least the number of possible plaintexts.
- Proof:
  - Given in class last week
- Corollary: Key length of one-time pad is optimal ☹️

# Computational security

- The computation approach to security is more relaxed
  - It only worries about polynomial adversaries
  - Adversaries may succeed with very small probability
- Why are these relaxations required ?
  - We want the number of possible keys to be smaller than the number of possible plaintexts  $|K| < |M|$ .
  - (*brute force attack*) Given a ciphertext, an adversary can decrypt it with all keys. Since  $|K| < |M|$ , results cannot contain all messages and this leaks some information about the plaintext.
  - (*key guess*) Given a ciphertext  $c$  and a plaintext  $m$ , the adversary can guess at random a key  $k$  and check if  $E_k(m) = c$ .



# Computational security

- How this works
  - Define a family of cryptosystems, based on a parameter  $n$  (often the key length).
  - Each choice of  $n$  defines a specific cryptosystem.
  - Encryption and decryption run in time polynomial in  $n$ .
  - “negligible probability” = smaller than any inverse polynomial in  $n$ . (see below)
  - The system is secure if any polynomial time adversary has a negligible probability of success.

## An example

- A cryptosystem
  - Encryption and decryption take  $10^6 n^2$  cycles.
  - An adversary (who doesn't have the key) that runs  $10^8 n^4$  cycles, decrypts with probability at most  $2^{20} 2^{-n}$
- Suppose  $n=50$ , and 1Ghz computer
  - Encryption and decryption take 2.5 seconds.
  - Adversary runs 1 week and decrypts with probability  $2^{-30}$
- Suppose we have 16Ghz computers, and set  $n=100$ .
  - Encryption and decryption take 0.625 seconds.
  - Adversary runs 1 week and decrypts with probability  $2^{-80}$ .

## Negligible success probability

- A function  $f()$  is *negligible* if  $\forall$  polynomial  $p()$ ,  $\exists N$ , s.t.  $\forall n > N$  it holds that  $f(n) < 1/p(n)$ .
- The functions  $2^{-n}$ ,  $2^{-n^{0.5}}$ , and  $2^{-\log^2(n)}$  are all negligible.
  - $2^{-n}$  is smaller than  $10^{-6}$  for all  $n > 20$
  - $2^{-n^{0.5}}$  is smaller than  $10^{-6}$  for all  $n > 400$
  - $2^{-\log^2(n)}$  is smaller than  $10^{-6}$  for all  $n > \approx 10^6$
  - (Note however that for  $n > 65536$ ,  $2^{-n^{0.5}} < 2^{-\log^2(n)}$  )

# Computational security

- We should only worry about polynomial adversaries
- Idea: Generate a string which “looks random” to any polynomial adversary. Use it instead of a OTP.
- What does it mean for a string to look random?
  - Fraction of bits set to 1 is  $\approx 50\%$
  - Longest run of 0's is of length  $\approx \log(n)$ ,
  - Is that sufficient?...
- Enumerating a set of statistical tests that the string should pass is not enough.

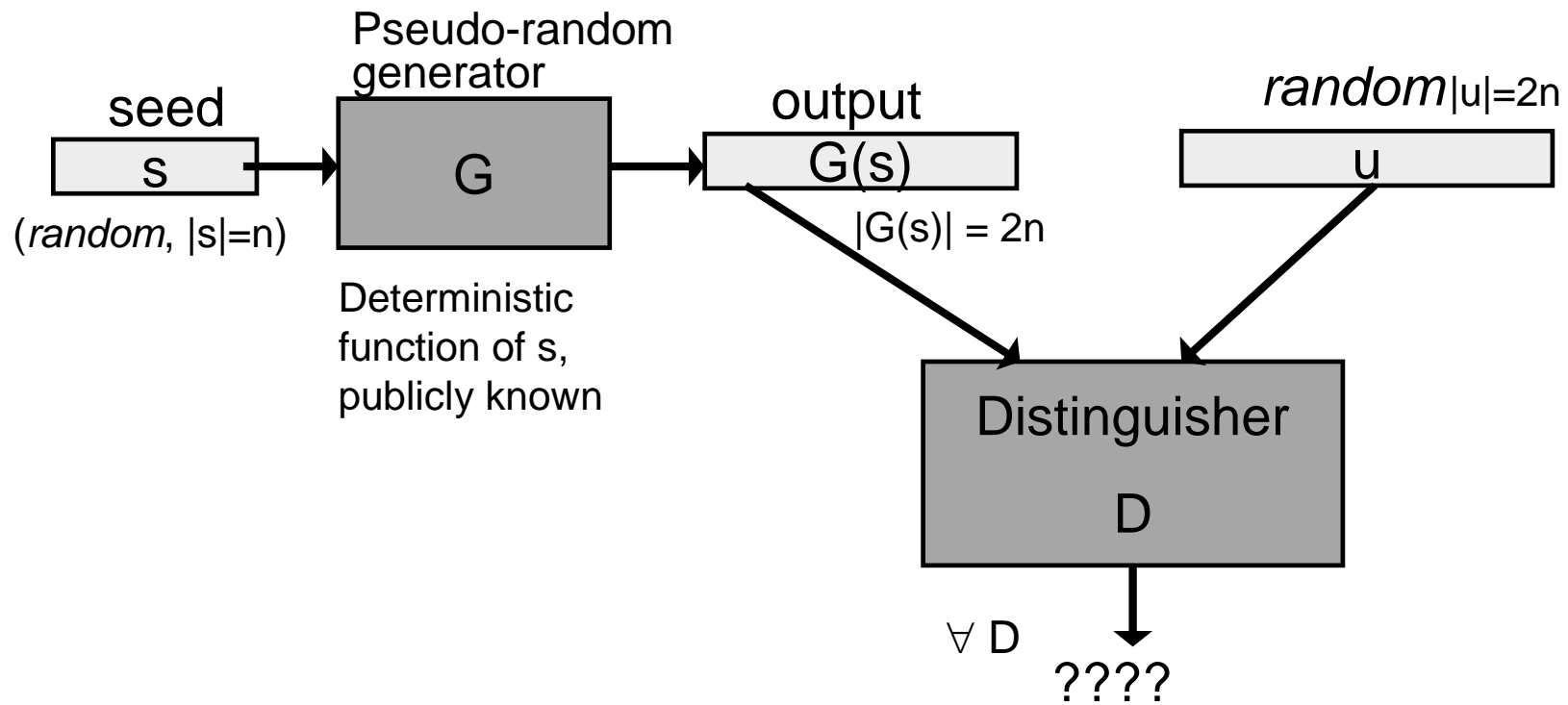
## Computational security – Pseudo-randomness

- Pseudo-random string:
  - No *efficient* observer can *distinguish* it from a uniformly random string of the same length
  - It “looks” random as long as the observer runs in polynomial time
- Motivation: *Indistinguishable objects are equivalent*
  - So, can use the pseudo-random string instead of a random one
- The foundation of modern cryptography
- (Note that no fixed string can be pseudo-random, or random. We consider a distribution of strings. A distribution of strings of length  $m$  is pseudo-random if it is indistinguishable from the uniform distribution of  $m$  bit strings.)

# Pseudo-random generators

- Pseudo-random generator (PRG)
  - $G: \{0,1\}^n \Rightarrow \{0,1\}^m$ 
    - A deterministic function, computable in polynomial time.
    - It must hold that  $m > n$ . Let us assume  $m=2n$ .
    - The function has only  $2^n$  possible outputs.
- Pseudo-random property:
  - $\forall$  polynomial time adversary  $D$ , (whose output is 0/1) if we choose inputs  $s \in_R \{0,1\}^n$ ,  $u \in_R \{0,1\}^m$ , (in other words, choose  $s$  and  $u$  uniformly at random), then it holds that  $D(G(s))$  is similar to  $D(u)$  namely,  $|\Pr[D(G(s))=1] - \Pr[D(u)=1]|$  is negligible

# Pseudo-random generator



# Properties of PRGs

- How can the adversary distinguish the PRG's output from a random one? (Exhaustive search?)
- Claim: If  $G$  is a PRG then it passes all statistical tests (e.g., the probability that the number of 1 bits in the PRG's output is  $< |m|/3$  is negligible).
- Can the output of  $G$  contain its input?
  - $G(\text{seed}) = \text{seed} \mid G'(\text{seed})$
- Implementation of PRGs:
  - Based on mathematical/computational assumptions
  - Ad-hoc constructions



## Using a PRG for Encryption

- Replace the one-time-pad with the output of the PRG
- Key: a (short) random key  $k \in \{0,1\}^{|k|}$ .
- Message  $m = m_1, \dots, m_{|m|}$ .
- Use a PRG  $G : \{0,1\}^{|k|} \rightarrow \{0,1\}^{|m|}$
- Key generation: choose  $k \in \{0,1\}^{|k|}$  uniformly at random.
- Encryption:
  - Use the output of the PRG as a one-time pad. Namely,
  - Generate  $G(k) = g_1, \dots, g_{|m|}$
  - Ciphertext  $C = g_1 \oplus m_1, \dots, g_{|m|} \oplus m_{|m|}$
- This is an example of a *stream cipher*.

## Security of encryption against polynomial adversaries

- Perfect security (previous equivalent defs):
  - (indistinguishability)  $\forall m_0, m_1 \in M, \forall c$ , the probability that  $c$  is an encryption of  $m_0$  is equal to the probability that  $c$  is an encryption of  $m_1$ .
  - (semantic security) The distribution of  $m$  given the encryption of  $m$  is the same as the a-priori distribution of  $m$ .
- Security of pseudo-random encryption (equivalent defs):
  - (indistinguishability)  $\forall m_0, m_1 \in M$ , no *polynomial time* adversary  $D$  can distinguish between the encryptions of  $m_0$  and of  $m_1$ . Namely,  $\Pr[D(E(m_0))=1] \approx \Pr[D(E(m_1))=1]$
  - (semantic security)  $\forall m_0, m_1 \in M$ , a polynomial time adversary which is given  $E(m_b)$ , where  $b \in_r \{0, 1\}$ , succeeds in finding  $b$  with probability  $\approx 1/2$ .

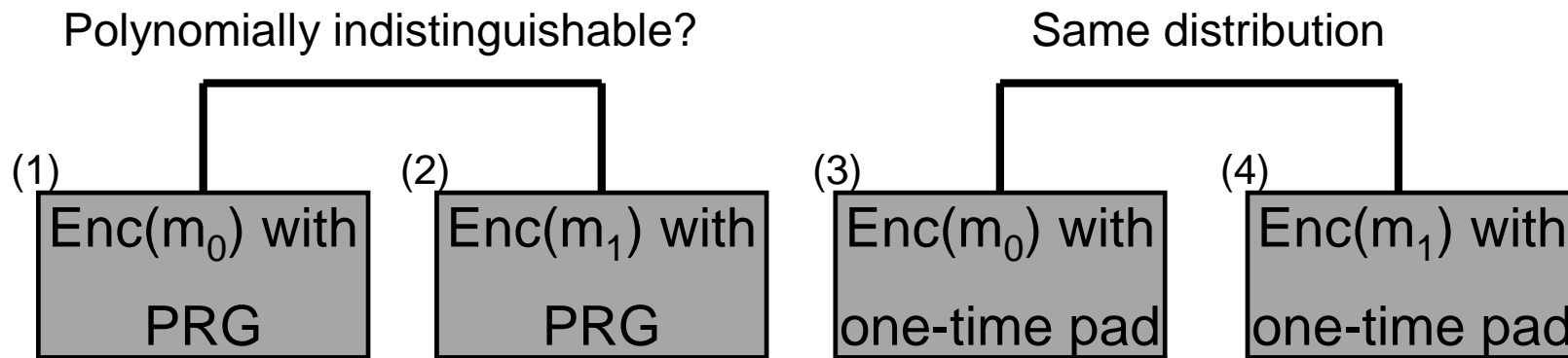
# Proofs by reduction

- We don't know how to prove unconditional proofs of computational security; we must rely on assumptions.
  - We can simply assume that the encryption scheme is secure. This is bad.
  - Instead, we will assume that some low-level problem is hard to solve, and then prove that the cryptosystem is secure under this assumption.
  - (For example, the assumption might be that a certain function  $G$  is a pseudo-random generator.)
  - Advantages of this approach:
    - It is easier to design a low-level function.
    - There are (very few) “established” assumptions in cryptography, and people prove the security of cryptosystem based on these assumptions.

## Using a PRG for Encryption: Security

- The output of a pseudo-random generator is used for the encryption.
- Proof of security by reduction:
  - The assumption is that the PRG is strong (its output is indistinguishable from random).
  - We want to prove that in this case the encryption is strong (it satisfies the indistinguishability definition above).
  - In other words, prove that if one can break the security of the encryption (distinguish between encryptions of  $m_0$  and of  $m_1$ ), then it is also possible to break the security of the PRG (distinguish its output from random).

# Proof of Security



- Suppose that there is a  $D()$  which distinguishes between (1) and (2)
- We know that no  $D()$  can distinguish between (3) and (4)
- We are given a string  $S$  and need to decide whether it is drawn from a pseudorandom distribution or from a uniformly random distribution
- Choose a random  $b \in \{0,1\}$  and compute  $m_b \oplus S$ . Give the result to  $D()$ .
  - if  $S$  was chosen uniformly,  $D()$  must distinguish (3) from (4). (impossible)
  - if  $S$  is pseudorandom,  $D()$  must distinguish (1) from (2). (easy)
- If  $D()$  outputs  $b$  then declare “pseudorandom”, otherwise declare “random”.

# Stream ciphers

- Stream ciphers are based on pseudo-random generators.
  - Usually used for encryption in the same way as OTP
- Examples: A5, SEAL, RC4.
  - Very fast implementations.
  - RC4 is popular and secure when used correctly, but it was shown that its first output bytes are biased. This resulted in breaking WEP encryption in 802.11.
- Some technical issues:
  - Stream ciphers require *synchronization* (for example, if some packets are lost in transit).