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(plaintext = P | ciphertext = C) = Pr(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,

  $\forall$ $m_1, m_2 \in M$, $\forall$cipher c,

  $$\Pr(Enc(m_1)=c) = \Pr(Enc(m_2)=c).$$

  (one direction was proved in the recitation)

- 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.
- *Indistinguishability* is equivalent to *semantic security*.

Proof

- Note that the proof cannot assume that the cipher is the one-time-pad

- We can only assume that $Pr(\ plaintext = P \mid ciphertext = C) = Pr(\ plaintext = P)$

Proof (of one direction; the other direction was proved in the recitation)

- Perfect security:
  - $\forall\ m \in M$, $\forall$cipher c, Pr(plaintext=m / ciphertext=c) = Pr(plaintext=m).
- Indistinguishability criterion:
  - $\forall\ m_1, m_2 \in M$, $\forall$cipher c, $\Pr(Enc(m_1)=c) = \Pr(Enc(m_2)=c)$.

- Perfect security $\Rightarrow$ Indistinguishability criterion

  $\Pr(Enc(m_1)=c) = \Pr(\text{ciphertext}=c \,/\, \text{plaintext}=m_1)$

  $= \Pr(\text{ciphertext}=c \text{ and plaintext}=m_1) \,/\, \Pr(\text{plaintext}=m_1)$

  $= \Pr(\text{plaintext}=m_1 \,/\, \text{ciphertext}=c) \cdot \Pr(\text{ciphertext}=c) \,/\, \Pr(\text{plaintext}=m_1)$

  $= 1\cdot \Pr(\text{ciphertext}=c) \,/\, 1 = \Pr(\text{ciphertext}=c)$

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, namely $|K|<|M|$.
  - *(brute force attack)* Given a ciphertext, an adversary can try to decrypt it with all possible keys. Since $|K|<|M|$, the 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 $E_k(m)=c$. If this holds, the adversary can decrypt other ciphertexts which use k.

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.

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} is smaller than n^{-4} for all n>16

  - $2^{-n^{0.5}}$ is smaller than 10^{-6} for all n>400
  - $2^{-n^{0.5}}$ is smaller than n^{-4} for all n>1900

  - $2^{-\log^2(n)}$ is smaller than 10^{-6} for all n> $\approx 10^3$
  - $2^{-\log^2(n)}$ is smaller than n^{-4} for all n>16

An example

- A cryptosystem
  - Encryption and decryption take $2^{20}n^2$ cycles.
  - An adversary (who doesn't have the key) that runs 10^8n^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

- In practice
  - An event that happens with probability 2^{-30} is non-negligible (likely to happen over 1GB of data)

  - An event that happens with probability 2^{-80} is negligible

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 ≈ 50%
  - Longest run of 0’s is of length ≈ 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

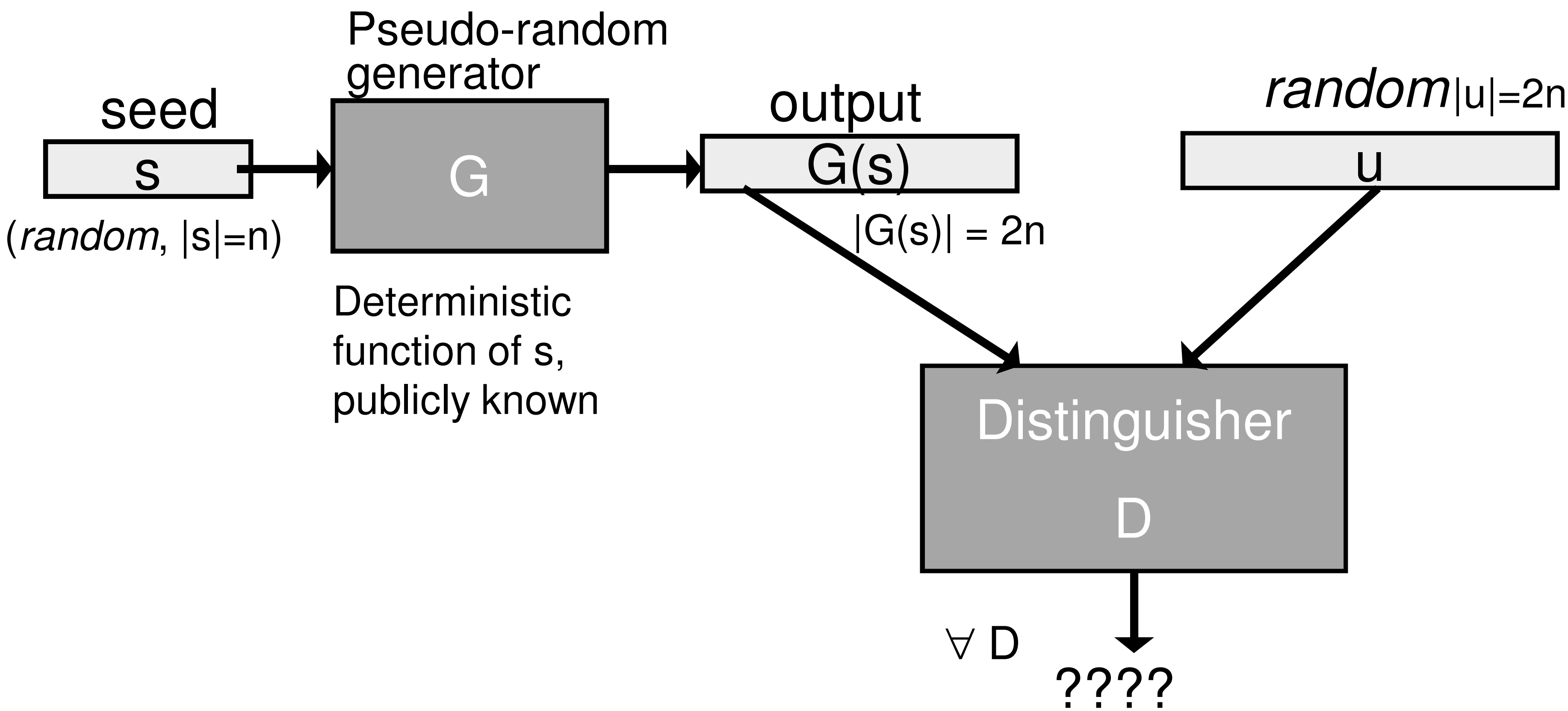
- (Note that no fixed string can be pseudo-random, or random. We consider a distribution of strings. A distribution of strings of length m is pseudo-random if it is indistinguishable from the uniform distribution of m bit strings.)

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:
  - 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 no polynomial adversary can distinguish between G(*s*) and *u*.
  - In other words, it holds $\forall$ polynomial time adversary D, (whose output is 0/1) that D(G(*s*)) is indistinguishable from D(*u*))

    $| \Pr[D(G(s))=1] - \Pr[D(u)=1] |$ is negligible.

Pseudo-random generator



Properties of PRGs

- How can the adversary distinguish the PRG's output from a random one? (Exhaustive search?)
- Claim (to be proved in the recitation): 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).

Properties of PRGs

- The value | Pr[D(G(s))=1] - Pr[D(u)=1] | is called the advantage of the algorithm D.
- The PRG is secure if $\forall$poly D the advantage is negligible.

- Can G(seed) be such that the xor of all its bits is always 1?
- Can the output of G contain its input?
  - G(seed)= seed | G'(seed)

Properties of PRGs

- The value | Pr[D(G(s))=1] - Pr[D(u)=1] | is called the advantage of the algorithm D.
- The PRG is secure if $\forall$poly D the advantage is negligible.

- Can G(seed) be such that the xor of all its bits is always 1?
- Can the output of G contain its input?
  - G(seed)= seed | G'(seed)

- Implementation of PRGs:
  - Based on mathematical/computational assumptions
  - Ad-hoc constructions

Predictability

- The output of a PRG is unpredictable
- There is no efficient alg A() that given the first *j* bits of G() can predict the next bit with non-negligible prob.

- Proof:
  - Suppose that $\exists$poly A() s.t. $\text{Prob}_{seed}(A(\, G(seed)|_{1\ldots j}\,) = G(seed)|_{j+1}\,)$ is $½+\delta$, where δ is non-negligible.
  - Define a distinguisher, as D(X)=1 iff $X|_{j+1} = A(\, X|_{1\ldots j}\,)$.
    - If X is uniform, then Prob(D(X)=1) = ½.
    - If X=G(seed) then Prob(D(X)=1) = $½+\delta$.
    - The advantage of D() is δ and is non-negligible.

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 $m = m_1,\ldots,m_{|m|}$.
- 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 $G(k) = g_1,\ldots,g_{|m|}$
  - Ciphertext $C = g_1 \oplus m_1,\ldots, g_{|m|} \oplus m_{|m|}$

- This is an example of a *stream cipher.*

Definitions of 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 $\approx$ ½.

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.

Proofs by reduction

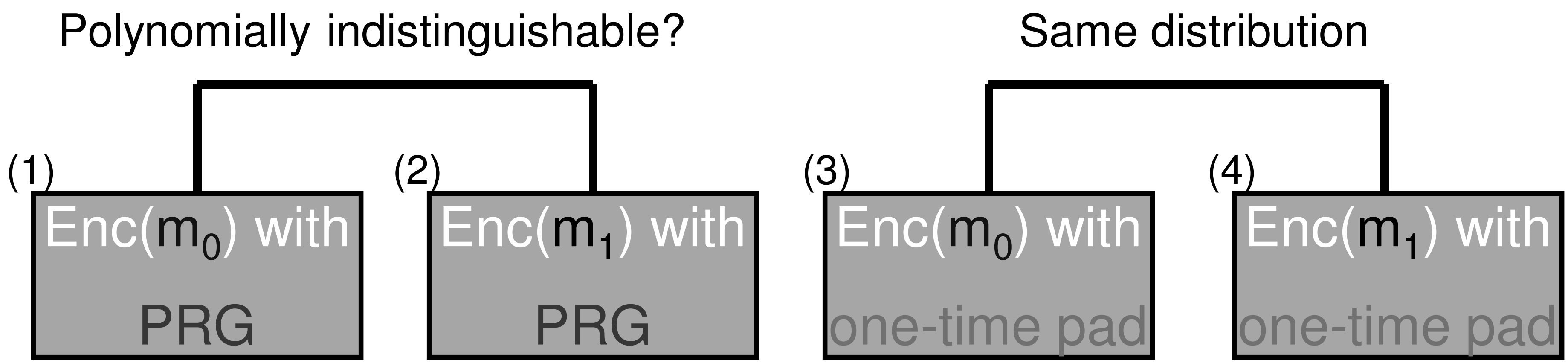
- We don't know how to prove unconditional proofs of computational security; we must rely on assumptions.
  - We can simply assume that the encryption scheme is secure. This is bad.
  - Instead, we will assume that some low-level problem is hard to solve, and then prove that the cryptosystem is secure 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).

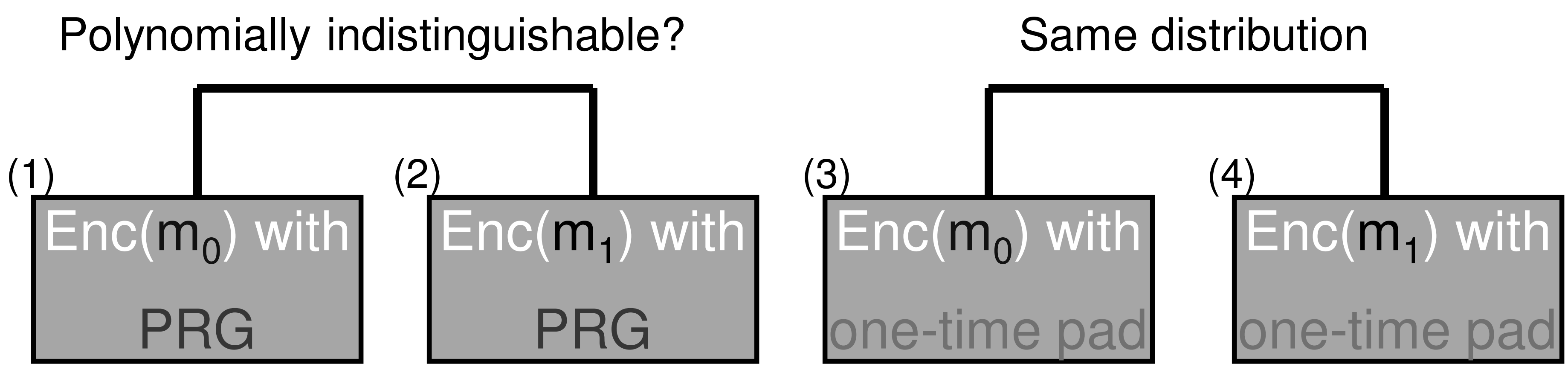
  - In other words, prove that if one can break the security of the encryption (distinguish between encryptions of m_0 and of m_1), then it is also possible to break the security of the PRG (distinguish its output from random).

Proof of Security



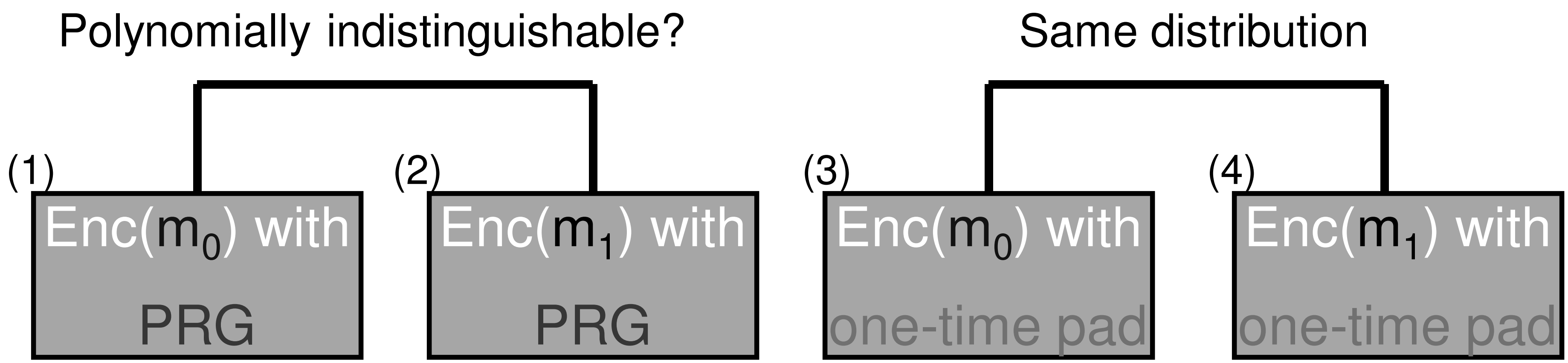
- Suppose that there is a distinguisher algorithm D'() which distinguishes between (1) and (2) (for now, assume that D' always succeeds)
- We know that no D'() can distinguish between (3) and (4)
- We are given a string S and need to decide whether it is drawn from a pseudorandom distribution or from a uniformly random distribution
- We will use S as a pad to encrypt a message.

Proof of Security



- Recall: we assume that there is a D’() which always distinguishes between (1) and (2). D’ cannot distinguish between (3) and (4) with probability > ½.
- 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). (prob=½)
  - if S is pseudorandom, D’() must distinguish (1) from (2). (prob=1)
- If D’() outputs b then declare “pseudorandom”, otherwise declare “random”.
  - if S was chosen uniformly we output “pseudorandom” with prob ½.
  - if S is pseudorandom we output “pseudorandom” with prob 1.

Proof of Security



- Recall: we assume that there is a D’() which distinguishes between (1) and (2) with prob ½+δ. D’ cannot distinguish between (3) and (4) with probability>½
- 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). (prob=½)
  - if S is pseudorandom, D’() must distinguish (1) from (2). (prob=½+δ)
- If D’() outputs b then declare “pseudorandom”, otherwise declare “random”.
  - if S was chosen uniformly we output “pseudorandom” with prob ½.
  - if S is pseudorandom we output “pseudorandom” with prob ½+δ.

ADD MY USUAL BLOCK DIAGRAM

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).