CS306: Introduction to IT Security

Fall 2020

Lecture 4: Ciphers in Practice (I)

Instructor: **Nikos Triandopoulos**

September 22, 2020

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Assignment Project Exam Help

https://powcoder.com

4.0 Announcements

Add WeChat powcoder

CS306: Other announcements

- Homework assignment HW1 is out
  - due in almost two weeks: Friday, October 2
  - covering perfect secrecy, classical ciphers and OTP
    - relevant materials
      - Lectures 2.2, 2.3, 3.1, 3.2, lecture 4.demo
      - Lab 2
  - please
    - start early
    - ask for help if needed
    - respect the non-collaboration policy

CS306: Tentative Syllabus

Week	Date	Topics	Reading	Assignment
1	**Sep 1**	**Introduction**	**Lecture 1**	**-**
2	**Sep 8**	**Symmetric-key encryption**	**Lecture 2**	**Lab 1**
3	**Sep 15**	**Perfect secrecy**	**Lecture 3**	**Lab 2, HW 1**
4	Sep 22	Public-key crypto I		
5	Sep 29	Public-key crypto II		
6	Oct 6	Access control & authentication		
-	Oct 13	**No class (Monday schedule)**		
7	Oct 20	**Midterm**	All materials covered	

CS306: Tentative Syllabus (continued)

Week	Date	Topics	Reading	Assignment
8	Oct 27	Software & Web security		
9	Nov 3	Network security		
10	Nov 10	Database security		
11	Nov 17	Cloud security		
12	Nov 24	Privacy		
13	Dec 1	Economics		
14	Dec 8	Legal & ethical issues		
15	Dec 10 (or later)	**Final** (closed “books”)	All materials covered*	

* w/ focus on what covered after midterm

Last week

- Symmetric-key Cryptography
  - Perfect secrecy
  - The One-Time Pad cipher
- Demo
  - Why encryption matters?
  - Using the Wireshark packet analyser

Today

- Ciphers in practice
  - The big picture
  - Computational security
  - Pseudo-randomness
    - stream ciphers, pseudorandom generators

- Demo
  - The Caesar and Vigenère ciphers and their cryptanalysis
  - Pseudo-randomness in practice

4.1 Introduction to modern cryptography

Cryptography / cryptology

- Etymology
  - two parts: “crypto” + “graphy” / “logy”
  - original meaning: κρυπτός + γράφω / λόγος (**in Greek**)
  - English translation: secret + write / speech, logic
  - meaning: secret writing / the study of secrets
- Historically developed/studied for secrecy in communications
  - message encryption in the symmetric-key setting
  - main application area: use by military and governments

Classical Vs. modern cryptography

antiquity – ~70s

*“the **art** of **writing** and **solving codes**”*

- approach
  - ad-hoc design
  - trial & error methods
  - empirically evaluated

~80s – today

*“the study of **mathematical techniques** for **securing** information, systems, and distributed computations against **adversarial attacks**”*

- approach
  - systematic design & analysis
  - formal notions of security (or adversary)
  - rigorous proofs of security (or insecurity)

Example: Classical Vs. modern cryptography for encryption

antiquity – ~70s

*"the **art** of **writing** and **solving codes**"*

- **ad-hoc study**
  - vulnerabilities/insecurity of
    - Caesar's cipher
    - shift cipher
    - mono-alphabetic substitution cipher
    - Vigenère cipher

~80s – today

*"the study of **mathematical techniques** for **securing** information, systems, and distributed computations against **adversarial attacks**"*

- **rigorous study**
  - **problem statement**: secret communication over insecure channel
  - **abstract solution concept**: symmetric encryption, Kerckhoff's principle, perfect secrecy
  - **concrete solution & analysis**: OTP cipher, proof of security

Example: Differences of specific ciphers

Caesar's/shift/mono-alphabetic cipher

- substitution ciphers
  - Caesar's cipher
    - **shift is always 3**
  - shift cipher
    - **shift is unknown and the same for all characters**
  - mono-alphabetic substitution/Vigènere cipher
    - **shift is unknown and the same for all/many character occurrences**

The one-time pad

- also, a substitution cipher
  - **shift is unknown and independent for each character occurrence**

Formal treatment in modern cryptography

Problem is formulated as an abstract crypto primitive

- captures the essence of the problem at hand, provides clarity and focus

Design & evaluation of crypto primitives follows a systematic process

- (A) **formal definitions** (what it means for a crypto primitive to be secure?)

- (B) **precise assumptions** (which forms of attacks are allowed – and which aren't?)
- (C) **provable security** (why a candidate solution is secure – or not)?

(A) Formal definitions

abstract but rigorous description of security problem

- **computing setting** (to be considered)
  - involved parties, communication model, core functionality
- **underlying cryptographic scheme** (to be designed)
  - e.g., symmetric-key encryption scheme
- **desired properties** (to be achieved)
  - security related
  - non-security related
    - e.g., correctness, efficiency, etc.

(A) Why formal definitions are important?

- **successful project management**
  - good design requires clear/specific security goals
    - helps to avoid critical omissions or over engineering

- **provable security**

  - rigorous evaluation requires a security definition
    - helps to separate secure from insecure solutions

- **qualitative analysis/modular design**
  - thorough comparison requires an exact reference
    - helps to secure complex computing systems

Example: Problem at hand

abstract but rigorous description of security problem (to be solved)

secret communication

Insecure channel

Example: Formal definitions (1)

- computing setting (to be considered)
  - e.g., involved parties, communication model, core functionality

⇒ Alice, Bob, Eve

⇒ Alice wants to send a message m to Bob; Eve can eavesdrop sent messages

⇒ Alice/Bob may transform the transmitted/received message and share info

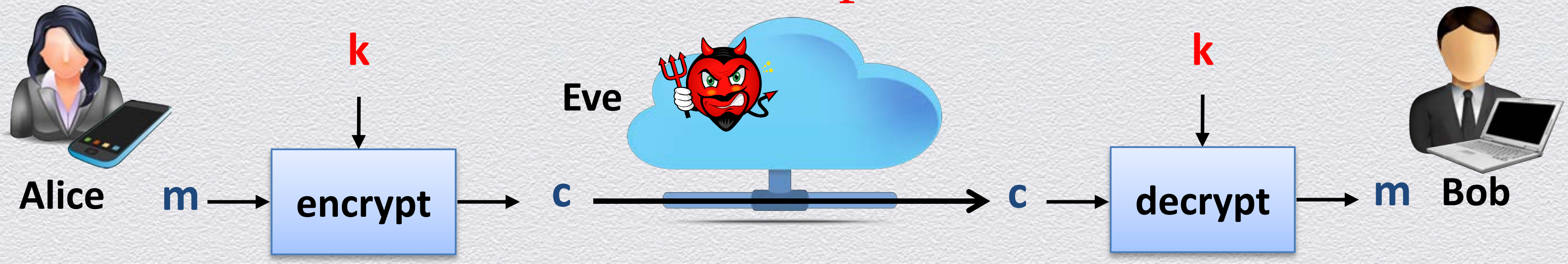
Example: Formal definitions (2)

- **underlying cryptographic scheme** (to be designed)

symmetric-key encryption scheme

- Alice and Bob share and use a key k
- Alice encrypts plaintext m to ciphertext c and sends c instead of m

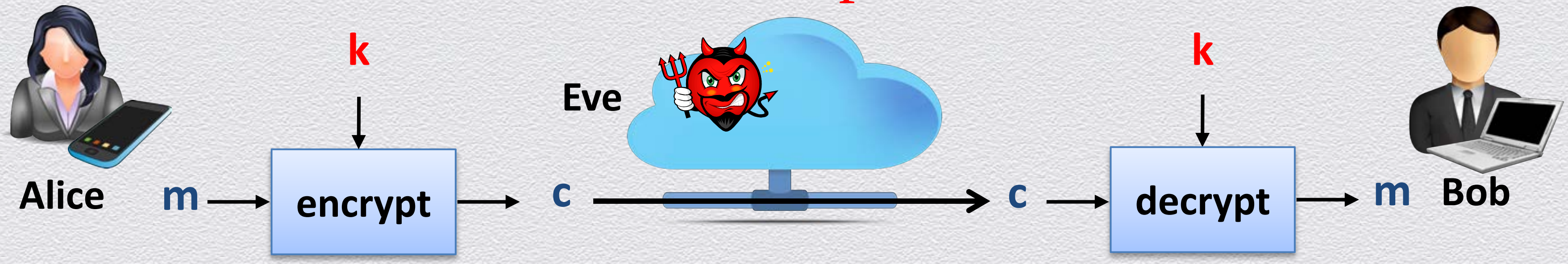
- Bob decrypts received c to get a message m’



Example: Formal definitions (3)

- **desired properties** (to be achieved)
  - **security (informal)** Eve “cannot learn” m (from c)
  - **correctness (informal)**

If Alice encrypts m to c, then Bobs decrypts c to (the original message) m



Example: Formal definitions (4)

Perfect correctness

- for any k ∈ $\mathcal{K}$, m ∈ $\mathcal{M}$ and any ciphertext c output of $\mathrm{Enc}_k(m)$, it holds that

$$\mathbf{Pr[\ Dec_k(c) = m\ ] = 1}$$

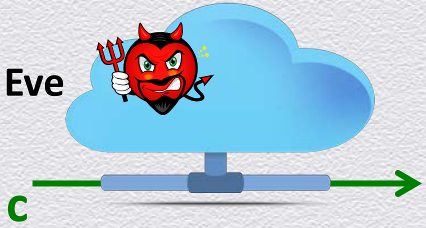
Perfect security (or information-theoretic security)

- the adversary should be able to learn **no additional** information on m

random experiment
$\mathcal{D}_\mathcal{M} \rightarrow \mathbf{m}$
$\mathcal{D}_\mathcal{K} \rightarrow$ k
$\mathbf{Enc_k(m) \rightarrow c}$

$$\mathbf{m} = \begin{cases} \texttt{attack} & \text{w/ prob. } 0.8 \\ \texttt{no attack} & \text{w/ prob. } 0.2 \end{cases}$$

Eve's view remains the same!



$$\mathbf{m} = \begin{cases} \texttt{attack} & \text{w/ prob. } 0.8 \\ \texttt{no attack} & \text{w/ prob. } 0.2 \end{cases}$$

(B) Precise assumptions

precise description of all relevant problem components

- **adversary / attacker**
  - type of attacks – a.k.a. **threat model**
  - **capabilities** (e.g., a priori knowledge, access to information, party corruptions)
  - **limitations** (e.g., bounded memory, passive Vs. active)
- **computational assumptions** (about hardness of certain tasks)
  - e.g., factoring of large composite numbers is hard
- **computing setting**
  - system set up, initial state, key distribution, randomness…
  - means of communication (e.g., channels, rounds, …)
  - timing assumptions (e.g., synchronicity, epochs, …)

(B) Why precise assumptions are important?

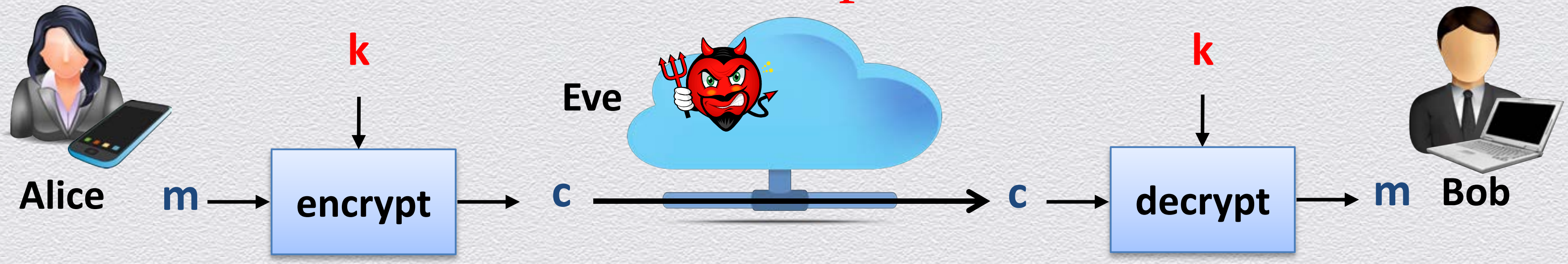
- **basis** for proofs of security
  - security holds under specific assumptions
- **comparison** among possible solutions
  - relations among different assumptions
    - stronger/weaker (i.e., less/more plausible to hold), “A implies B” or “A and B are equivalent”
    - refutable Vs. non-refutable
- **flexibility** (in design & analysis)
  - **validation** – to gain confidence or refute
  - **modularity** – to choose among concrete schemes that satisfy the same assumptions
  - **characterization** – to identify simplest/minimal/necessary assumptions

Example: Precise assumptions (1)

- **adversary**
  - type of attacks – a.k.a. **threat model** → eavesdropping
  - **capabilities** (e.g., a priori knowledge, access to information, party corruptions)
  - **limitations** (e.g., bounded memory, passive Vs. active)

→ Eve may know the a priori distribution of messages sent by Alice

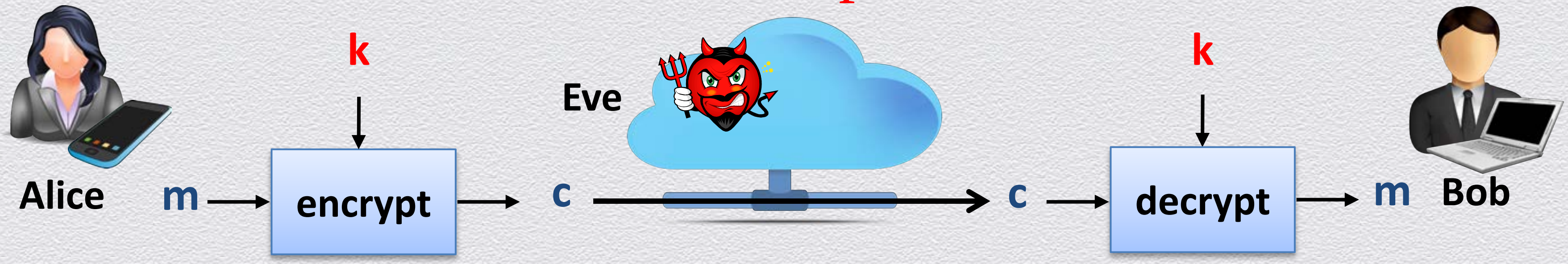
→ Eve doesn't know/learn the secret k (shared by Alice and Bob)



Example: Precise assumptions (2)

- **computational assumptions** (about hardness of certain tasks)
  - e.g., factoring of large composite numbers is hard

⇒ no computational assumptions
– a.k.a. perfect secrecy (or information-theoretic security)



Example: Precise assumptions (3)

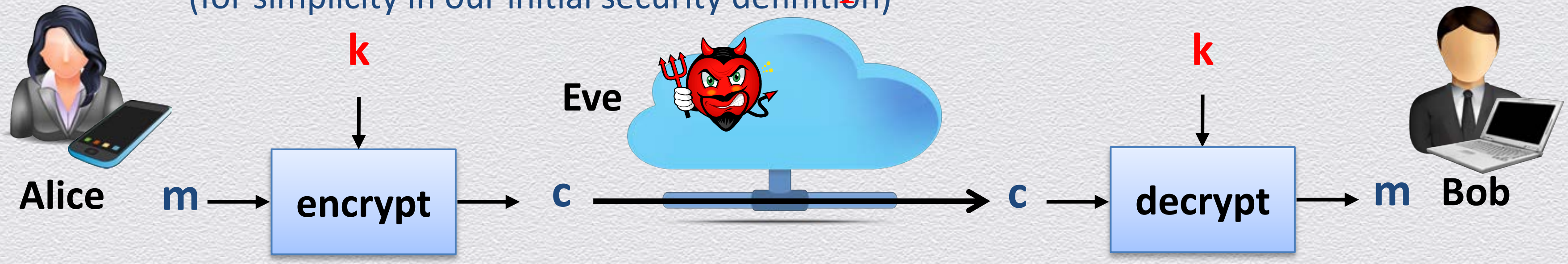
- **computing setting**
  - **system set up**, initial state, **key distribution**, **randomness**...
  - means of **communication** (e.g., channels, rounds, messages...)
  - timing assumptions (e.g., synchronicity, epochs, ...)

key k is generated randomly using the uniform distribution

key k is securely distributed to and securely stored at Alice and Bob

one message m is only communicated (for simplicity in our initial security definition)

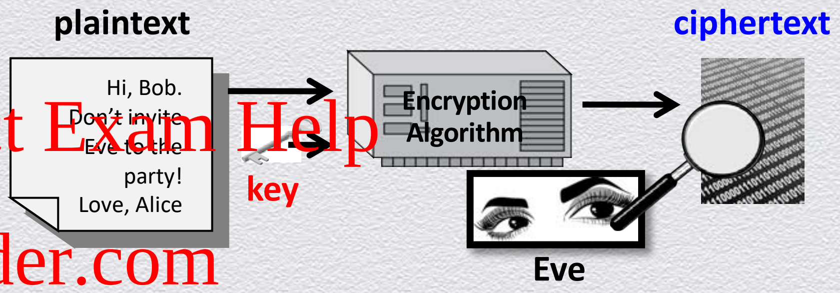
k, m are chosen independently



Possible eavesdropping attacks (I)

An attacker may possess a

- **collection of ciphertexts**
  - ciphertext only attack (or simply EAV)

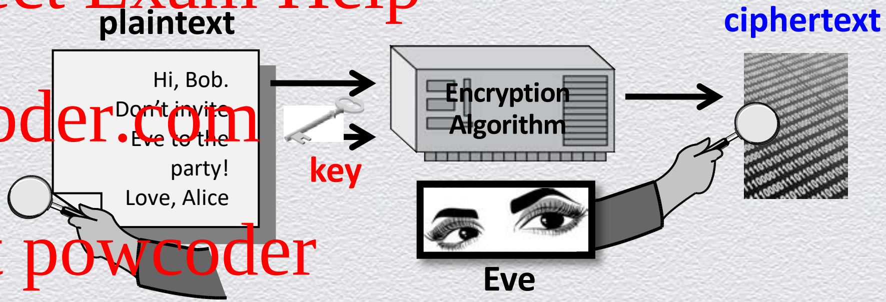


Plain thread model

Possible eavesdropping attacks (II)

An attacker may possess a

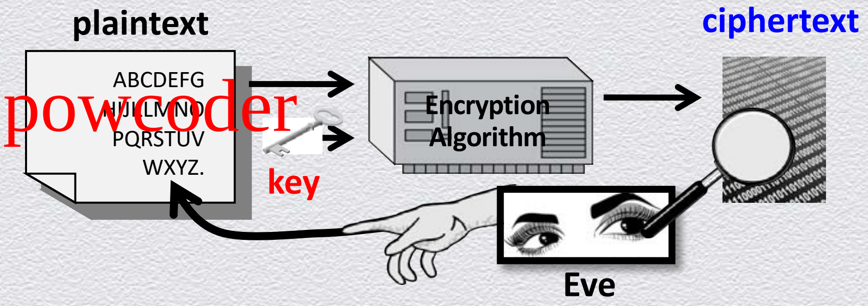
- **collection of plaintext/ciphertext pairs**
  - known plaintext attack



Possible eavesdropping attacks (III)

An attacker may possess a

- **collection of plaintext/ciphertext pairs for plaintexts selected by the attacker**
  - chosen plaintext attack (CPA)

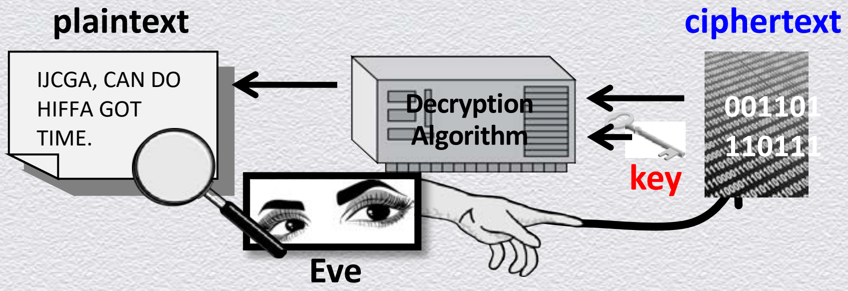


Advanced threat model

Possible eavesdropping attacks (IV)

An attacker may possess a

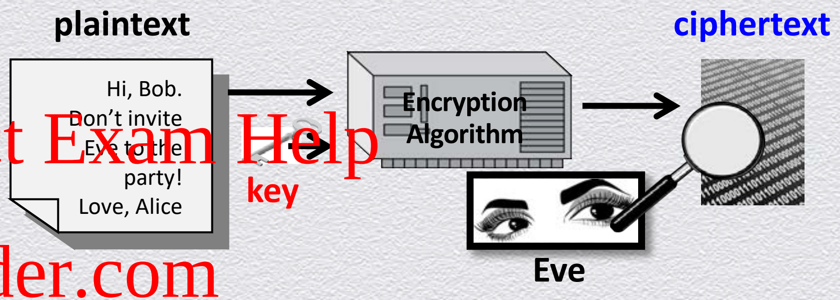
- **collection of plaintext/ciphertext pairs for ciphertexts selected by the attacker**
  - chosen ciphertext attack (CCA)



Main security properties against eavesdropping

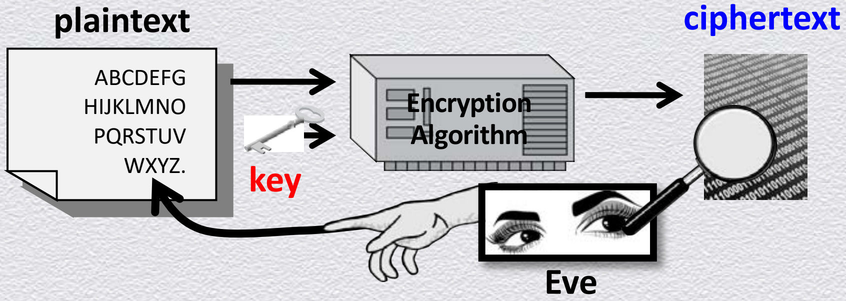
“plain” security

- protects against ciphertext-only attacks



“advanced” security

- protects against chosen plaintext attacks



(C) Provably security

Security

- subject to certain **assumptions**, a scheme is proved to be **secure** according to a specific **definition**, against a specific **adversary**
  - in practice the scheme may break if
    - some assumptions do not hold or the attacker is more powerful

Insecurity

- a scheme is proved to be **insecure** with respect to a specific **definition**
  - it suffices to find a **counterexample attack**

(C) Why provable security is important?

Typical performance

- in some areas of computer science
  formal proofs may not be essential
  - behavior of hard-to-analyze algorithms is simulated to experimentally study their performance on "typical" inputs
- in practice, **typical/average case** occurs

Worst case performance

- in cryptography and secure protocol design
  formal proofs are essential
  - "experimental" security analysis is not possible
  - the notion of a "typical" adversary makes little sense and is unrealistic
- in practice, **worst case attacks will occur**
  - an adversary will use any means in its power to break a scheme

4.2 Computational security

The big picture: OPT is perfect but impractical!

We formally defined and constructed the perfectly secure OTP cipher

- This scheme has some major drawbacks
  - it employs a very large key which can be used only once!
- Such limitations are unavoidable and make OTP not practical
  - why?

Now, what?

Our approach: Relax “perfectness”

Initial model

- the **perfect secrecy** (or security) requires that
  - the ciphertext **leaks absolutely no extra information** about the plaintext
  - to adversaries of **unlimited computational power**

Refined model

- a relaxed notion of security, called **computational security**, requires that
  - the ciphertext leaks **a tiny amount of extra information** about the plaintext
  - to adversaries with **bounded computational power**

Computational security

- to be contrasted against information-theoretic security
  - *de facto* way to model security in most settings
  - an integral part of modern cryptography w/ rigorous mathematical proofs
- entails two relaxations
  - security is guaranteed against **efficient** adversaries
    - if an attacker invests in **sufficiently large resources**, it may break security
    - goal: make required resources larger than those available to any realistic attacker!
  - security is guaranteed in a **probabilistic** manner
    - with some **small probability**, an attacker may break security
    - goal: make attack probability sufficiently small so that it can be practically ignored!

Towards a rigorous definition of computational security

Concrete approach

- “A scheme is (t,ε)-secure if any attacker A, running for time at most t, succeeds in breaking the scheme with probability at most ε”

Asymptotic approach

- “A scheme is secure if any efficient attacker A succeeds in breaking the scheme with at most negligible probability”

Examples

- almost optimal security guarantees
  - if key length n, the number of possible keys is 2^n
  - attacker running for time t succeeds w/ prob. at most ~ $t/2^n$ (brute-force attack)
- if n = 60, security is enough for attackers running a desktop computer
  - 4 GHz (4×10^9 cycles/sec), checking all 2^{60} keys require about 9 years
  - if n = 80, a supercomputer would still need ~2 years
- today's recommended security parameter is at least n = 128
  - large difference between 2^{80} and 2^{128}; e.g., #seconds since Big Bang is ~2^{58}
  - a once-in-100-years event corresponds to probability 2^{-30} of happening at a particular sec
  - if within 1 year of computation attack is successful w/ prob. $1/2^{60}$ then it is more likely that Alice and Bob are hit by lighting

4.3 Symmetric encryption, revisited: Security

Three equivalent "looks" of perfect secrecy

1) a posteriori = a priori

For every $\mathcal{D}_{\mathcal{M}}$, m ∈ $\mathcal{M}$ and c ∈ $\mathcal{C}$, for which Pr [C = c] > 0, it holds that

$$\Pr[\, M = m \mid C = c \,] = \Pr[\, M = m \,]$$

2) C is independent of M

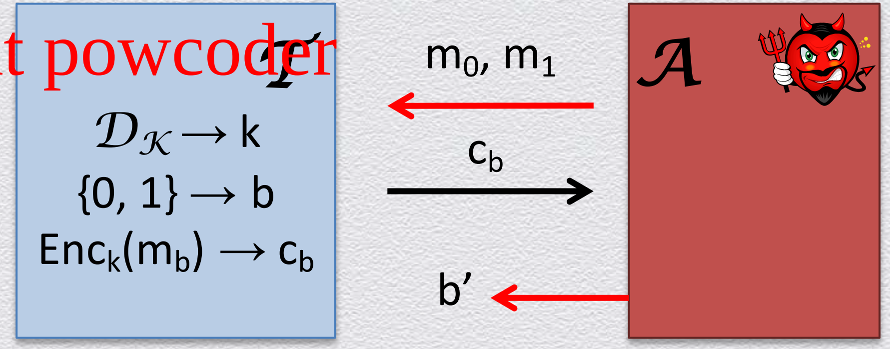
For every m, m' ∈ $\mathcal{M}$ and c ∈ $\mathcal{C}$, it holds that

$$\Pr[\, \mathrm{Enc}_K(m) = c \,] = \Pr[\, \mathrm{Enc}_K(m') = c \,]$$

3) indistinguishability

For every $\mathcal{A}$, it holds that

$$\Pr[\, b' = b \,] = 1/2$$



ciphertext looks **completely random**

Security relaxation

Perfect security: M, $Enc_K(M)$ are independent, **unconditionally**

- no extra information is leaked to any attacker

Computational security: M, $Enc_K(M)$ are independent, **for all practical purposes**

- no extra information is leaked but a tiny amount
  - e.g., with prob. 2^{-128} (or much less than the likelihood of being hit by lighting)

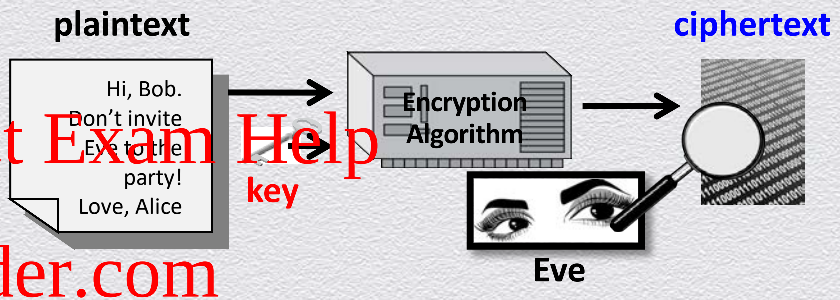
- to computationally bounded attackers
  - e.g., who cannot count to 2^{128} (or invest work of more than one century)

- attacker's best strategy remains **ineffective**
  - random guess a secret key or exhaustive search over key space (brute-force attack)

Recall: Main security properties against eavesdropping

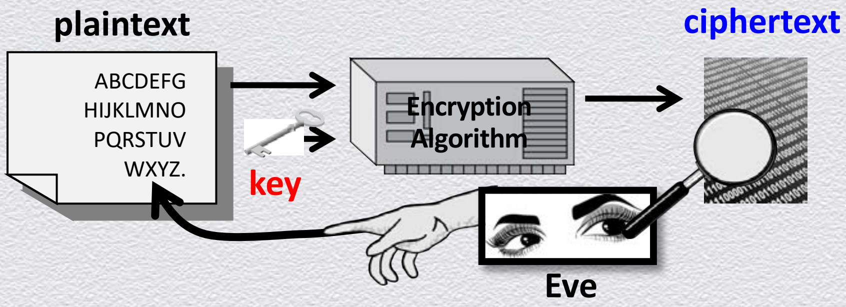
“plain” security

- protects against ciphertext-only attacks



“advanced” security

- protects against chosen plaintext attacks



Computational EAV-security or indistinguishability

Relax the definition of perfect secrecy that is based on indistinguishability

- require that target messages m_0, m_1 are chosen by a **PPT** attacker
- require that no such attacker can distinguish $Enc_k(m_0)$ from $Enc_k(m_1)$

non-negligibly better than guessing

3) indistinguishability

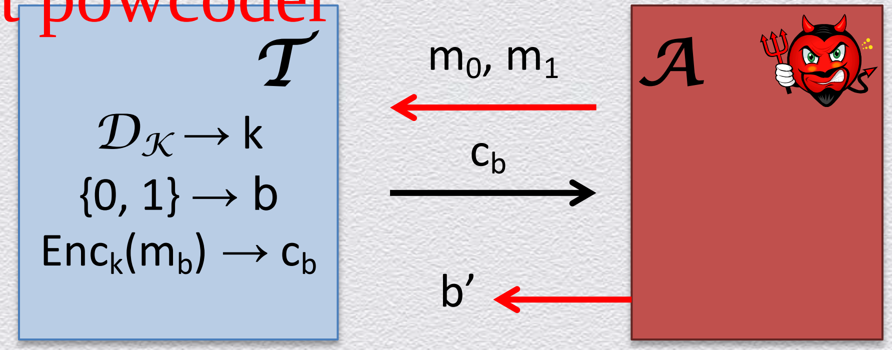
For every $\mathcal{A}$, it holds that

Pr[b' = b] = 1/2 + negligible

PPT

something that can be safely ignored

PPT



Computational CPA-security

Advanced security implies probabilistic encryption – why?

Strengthen the definition of computational plain-security

- allow attacker to have access to an **encryption "box"**
- allow the attacker to select m_0, m_1 **after** using this "box" (as many times as desired)

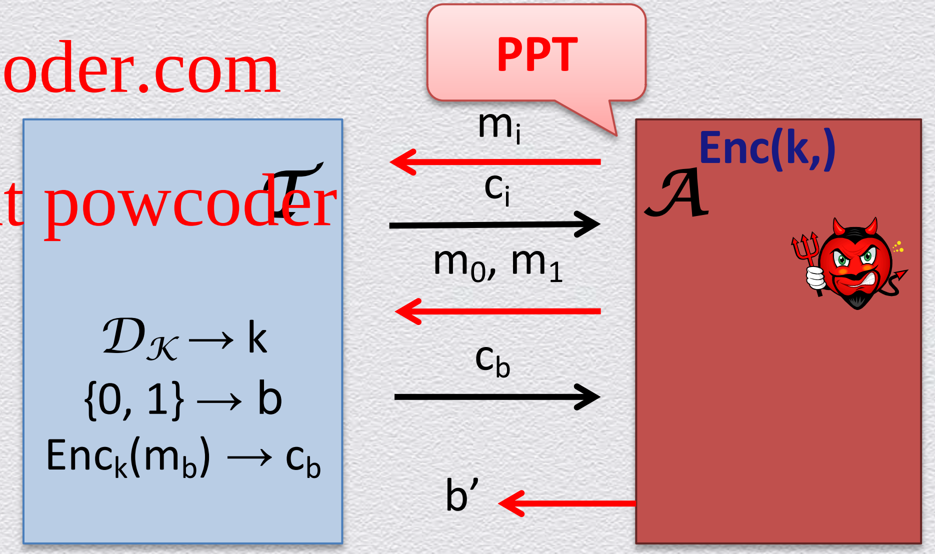
3) indistinguishability

For every PPT $\mathcal{A}$, it holds that

Pr[b' = b] = 1/2 + negligible

PPT

something that can be safely ignored



4.4 Symmetric encryption, revisited: OTP with pseudorandomness

Perfect secrecy & randomness

Role of randomness in encryption is **integral**

- in a perfectly secret cipher, the ciphertext **doesn't depend** on the message
  - the ciphertext appears to be **truly random**
  - the uniform key-selection distribution **is imposed also onto** produced ciphertexts
    - e.g., c = k XOR m (for uniform k and any distribution over m)

When security is computational, randomness is **relaxed** to "pseudorandomness"

- the ciphertext appears to be "**pseudorandom**"
  - it **cannot be efficiently distinguished** from truly random

Symmetric encryption as “OPT with pseudorandomness”

Stream cipher

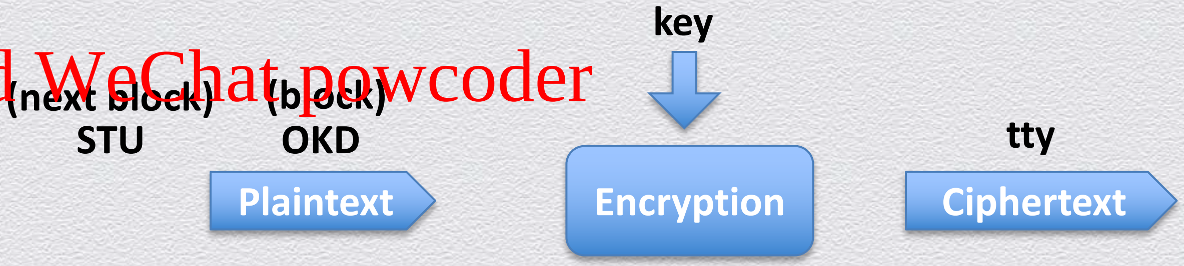
Uses a **short** key to encrypt **long** symbol **streams** into a **pseudorandom** ciphertext

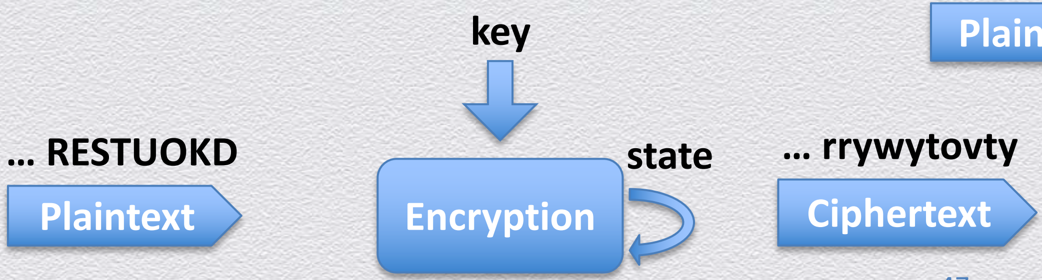
- based on abstract crypto primitive of **pseudorandom generator (PRG)**

Block cipher

Uses a **short** key to encrypt **blocks** of symbols into **pseudorandom** ciphertext blocks

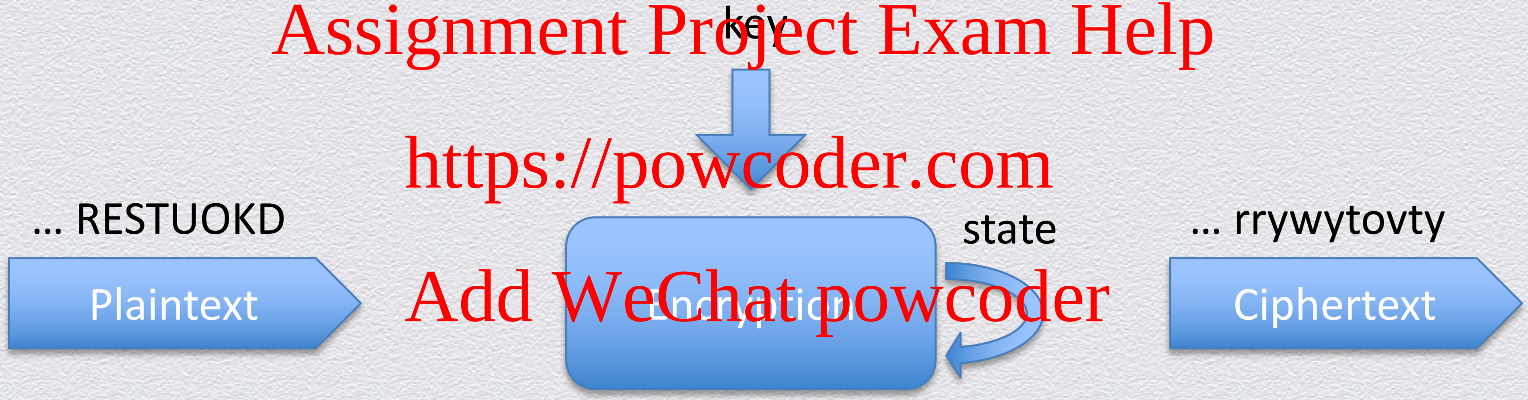
- based on abstract crypto primitive of **pseudorandom function (PRF)**



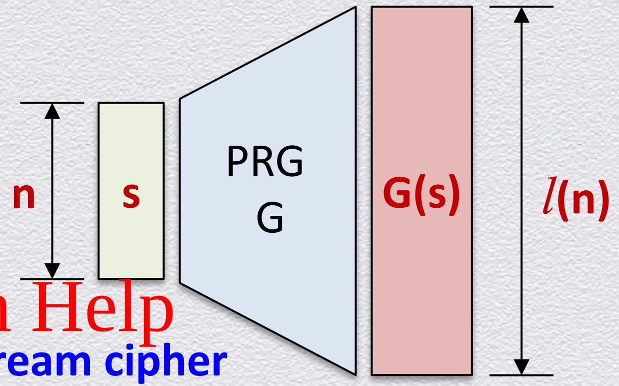


4.4.1 Pseudorandom generators

Stream ciphers



Pseudorandom generators (PRGs)



Deterministic algorithm G that

on input a **seed** $s \in \{0,1\}^t$, outputs $G(s) \in \{0,1\}^{l(t)}$

a.k.a. **stream cipher**

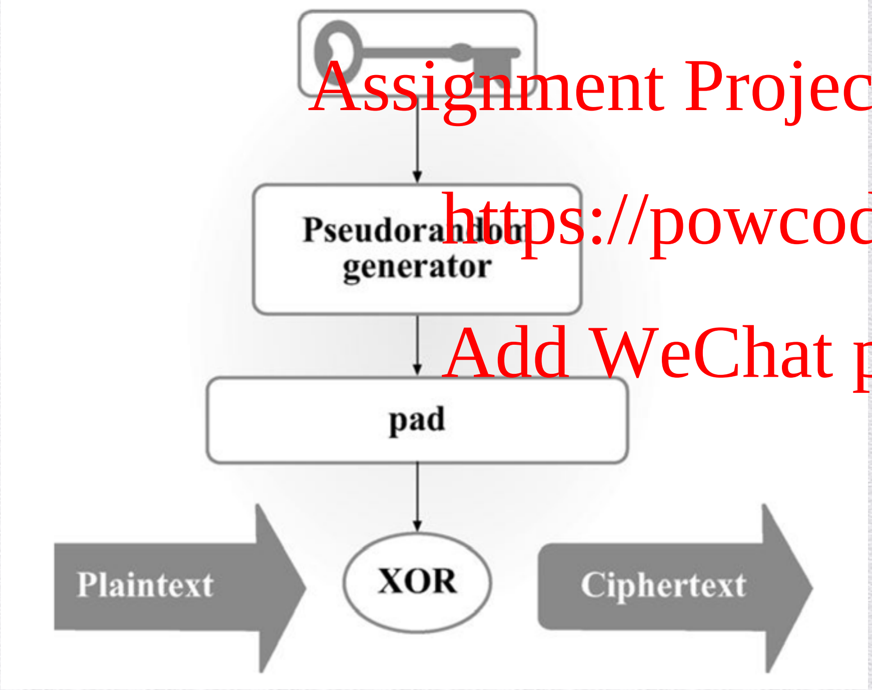
G is a PRG if:

- **expansion**

  - for polynomial l, it holds that for any n, $l(n) > n$
  - models the process of **extracting** randomness from a short random string
- **pseudorandomness**
  - **no efficient statistical test can tell apart** G(s) from a **truly random** string

Generic PRG-based symmetric encryption

- **Fixed-length** message encryption



encryption scheme is plain-secure as long as the underlying PRG is secure

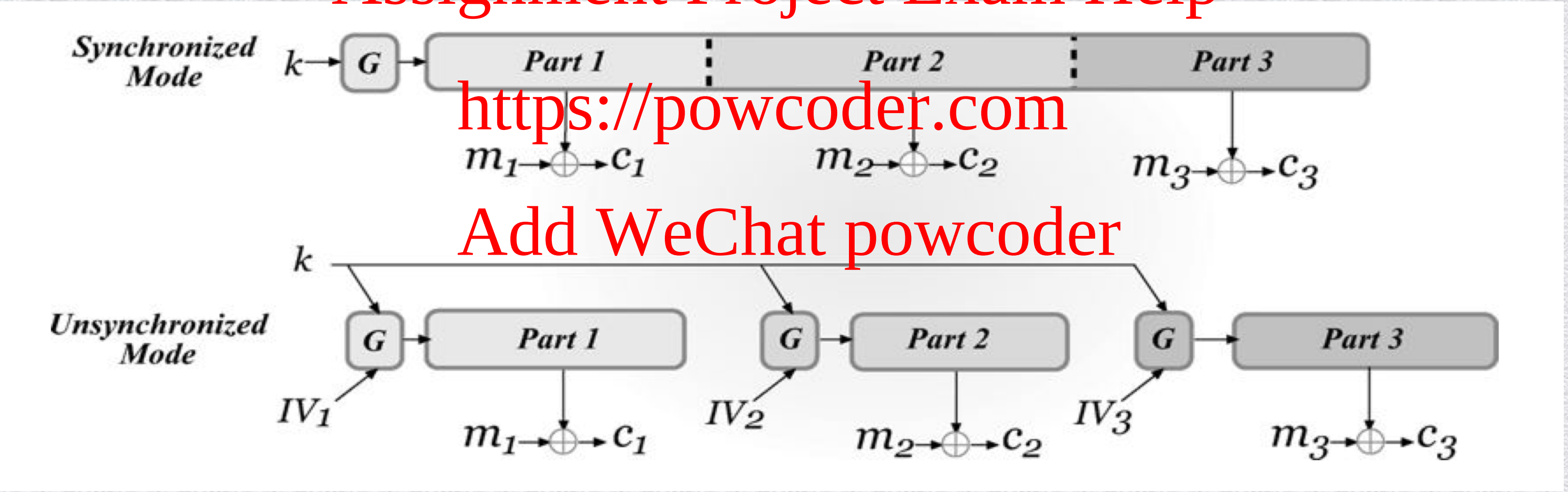
Generic PRG-based symmetric encryption (cont.)

- **Bounded- or arbitrary-length** message encryption
  - specified by a mode of operation for using an underlying stateful stream cipher, repeatedly, to encrypt/decrypt a stream of symbols

Stream ciphers: Modes of operations

- **Bounded or arbitrary-length** message encryption

on-the-fly computation of new pseudorandom bits, no IV needed, plain-secure



random IV used for every new message is sent along with ciphertext, advanced-secure