Crypto Models

Computer Security II

CS463/ECE424

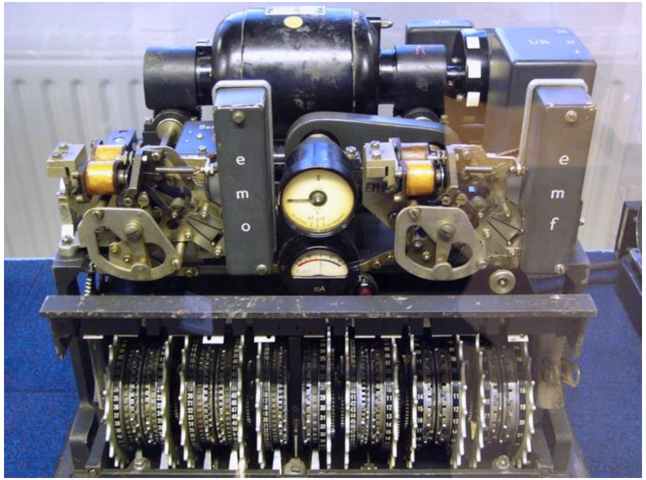
University of Illinois

Outline

Computational Security

Computationally Secure Encryption

Pseudorandomness



Recap: Perfectly Secret Encryption

- Defintion:
  - Message space $\mathcal{M}$ — set of all messages
  - Ciphertext space $\mathcal{C}$— set of all ciphertexts
  - *An encryption scheme (Gen, Enc, Dec) is* perfectly secret *if for every probability distribution over* $\mathcal{M}$*, every message* $m \in \mathcal{M}$*, and every ciphertext* $c \in \mathcal{C}$ *for which* $\Pr[c \in \mathcal{C}] > 0$:

$$\underbrace{\Pr[\boldsymbol{M} = m \mid \boldsymbol{C} = c]} = \underbrace{\Pr[\boldsymbol{M} = m]}$$

a posteriori distribution:
the probability that the message
was m if the ciphertext is c

a priori distribution:
the probability that the
message was m

Recap: One-Time Pad

- Message space $\mathcal{M}$, key space $\mathcal{K}$, ciphertext space $\mathcal{C}$ are $\{0,1\}^l$, for some integer $l > 0$.
- Gen: picks key uniformly at random in $k \in \mathcal{K}$.
- Enc: given key k, message $m \in \mathcal{M}$, output ciphertext $c = m \oplus k$.
- Dec: given key k, ciphertext $c \in \mathcal{C}$, output plaintext $m = c \oplus k$.

Perfectly Secret Encryption

- Perfectly secret encryption provides:
  - Security against an adversary that has unlimited computation power and unlimited time.
  - The adversary gains absolutely no knowledge about the plaintext from looking at the ciphertext.
- But, perfectly secret encryption (e.g., one-time pad) is impractical
  - Key must be at least as long as the message

Computational Security

- One (other) Kerckhoffs' Principle
  - "The [cryptosystem] should be, if not theoretically unbreakab
    unbreakable in practice."

- Basic idea: a scheme does not need to be perfectly secret, but only:
  - Not breakable within a reasonable amount of time
  - Not breakable with a reasonable probability of success

Computational Security

- We require:
  1. Security against an *efficient* adversary running in a *feasible amount of time*.
  2. Adversary may succeed with some *very small probability*.

- We need to precisely define:
  - Efficient adversary
  - Feasible amount of time
  - Very small probability

Computational Security

- There are two common approaches for formalization:
  1. Concrete approach
  2. Asymptotic approach

Concrete Approach

- (Template) Definition:
  - Let $t, \varepsilon > 0$, with $\varepsilon \leq 1$.
  - *A scheme is* (t, ε)-secure *if every adversary running for time at most t breaks the scheme with probability at most ε.*

- What does it mean to "break" the scheme?
- What is the time unit?

Concrete Approach

- Example (optimal security):
  - Modern encryption schemes with key length of n bits and adversary running in time t, are breakable with probability at most $t \times 2^{-n}$
  - Brute-force attack: try to decrypt using all 2^n keys
  - It may be useful to think of the time unit as being either:
    - Clock cycle
    - Time to invoke the decryption function

Concrete Approach

- How to get a feeling for t, ε?

- Today, computation on the order of $t = 2^{64}$ is somewhat within reach
  - A 4GHz computer will take roughly 146 years to execute for 2^{64} cycles.
  - Parallelization: with 146 such computers it would take only about one year

- Assume $n = 128$ bits
  - If an adversary runs a 4GHz computer for 146 years can break a modern encryption scheme (e.g., AES) with probability at most 2^{-64}.

  - Is probability 2^{-64} small enough?

Intuitive Sense: Asteroid Impact

- **What's the probability of us getting killed by an asteroid impact?**
- On average, an asteroid of mass 40 billion kilograms and 325m diameter can impact earth once every 80,000 years
- Probability of impact anywhere on earth at any given second is 2^{-41}
- Probability of impact on Champaign county at any given second is 2^{-59}, i.e., 32 times more than 2^{-64}
- Probability of impact on Champaign county in any given day is 2^{-42}, i.e., 4 million times more than 2^{-64}

Concrete Approach

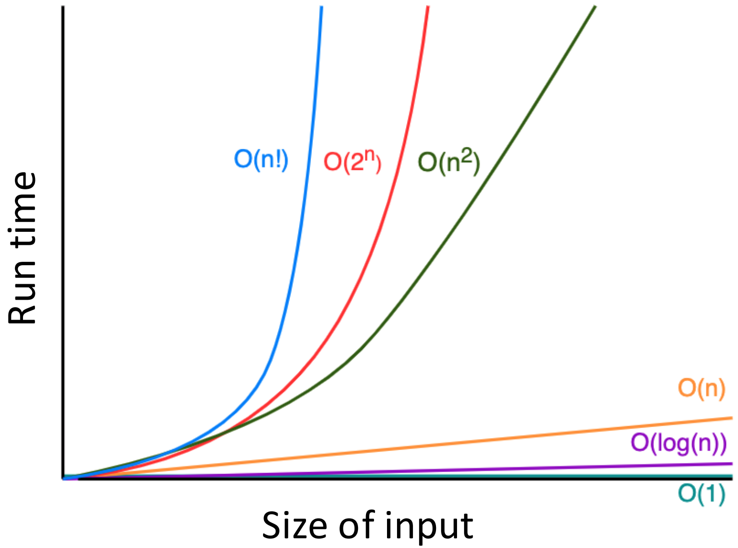
- The approach is useful, but we should be careful:
  1. For what values of t, ε can we say the scheme is "secure"?
  2. What are the capabilities of the adversary?
     - Hardware: "off-the-shelf" or custom-built?
     - Algorithms & Software: "off-the-shelf" or designed for the attack?
  3. What about Moore's law?
     - Computing power doubles every 18 months (?) --- attack now or later?
     - Now: 146 computer (4GHz) run for 1 year to break AES-128 with probability <= 2^{-64}.
     - Later (in 18×64 months $=$ 96 years): 146 computers (of the future) run for one year to break AES-128 with probability 1.

Asymptotic Approach (“Big O”)

Basic idea:

- Use complexity theory
- Running time and success probability of the adversary are *functions* (of some parameters), not concrete numbers

- Formally, we define a security parameter



Asymptotic Approach

- Security parameter:
  - For encryption: the key length, i.e., for encryption with an n-bit key, we say the security parameter is n.
- Efficiency:

Probabilistic Polynomial Time

  - Honest parties and feasible adversaries runs in PPT (in n).
  - PPT stands for *probabilistic polynomial time*, i.e., $a \times n^c$ for some constants a and c.
- Probability of success:
  - Must be smaller than any inverse polynomial, i.e., n^{-c} for every constant c. This is called a *negligible* probability.

Asymptotic Approach

- Concretely (for large enough values of n):
  - Running times of n^2, n^{10}, and even n^{1000} are all feasible.
  - Running times of 2^n, $2^{\sqrt{n}}$, and even $n^{\log n}$ are all infeasible.
  - 2^{-n}, $2^{-\sqrt{n}}$ and $n^{-\log n}$ are all negligible probabilities.

Asymptotic Approach

- (Template) Definition:
  - *A scheme is* secure *if every adversary running in PPT succeeds in breaking the scheme with only negligible probability.*

- Remarks:
  - The security only holds for large enough values of the security parameter.
  - We will use the asymptotic approach in the rest of the lecture.

Asymptotic Approach

- Example:
  - Take a secure scheme with security parameter n
  - Suppose there is an adversary can break the scheme running for n^3 minutes with probability $2^{40} \times 2^{-n}$.
  - Since 2^{-n+40} is a negligible function, the definition is satisfied
  - However, if we take $n = 40$, then the adversary can break the scheme by running for 40^3 minutes $\cong$ 6 weeks, with probability 1.

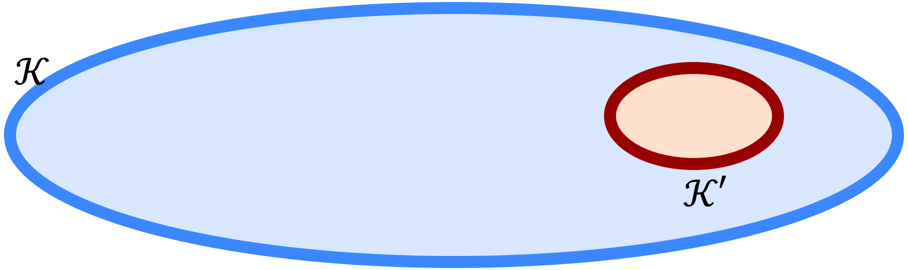
Computational Security

- Observation: we have relaxed the security in two ways
  1. We restrict the running time of the adversary to be polynomial (in terms of the security parameter)
  2. We allow the adversary to successfully break the scheme with some very small (i.e., negligible) probability

- Why?
  – because we are using short keys, i.e., $|\mathcal{K}| \ll |\mathcal{M}|$.

A. Running Time

- Suppose $|\mathcal{K}| \ll |\mathcal{M}|$, and the adversary is attempting a known-plaintext attack
  - Adversary knows pairs (m_i, c_i), such that: $c_i = \mathrm{Enc}_k(m_i)$
  - Adversary can perform a brute-force search:
    - Try all keys $k' \in \mathcal{K}$ until for all i: $\mathrm{Dec}_{k'}(c_i) = m_i$
    - Success probability is 1 and running time is linear in $|\mathcal{K}|$.



- We want the keyspace the adversary can explore $\mathcal{K}' \subset \mathcal{K}$ to be much smaller than $\mathcal{K}$, i.e., $|\mathcal{K}|$ must be super-polynomial in n.

B. Success Probability

- Suppose $|\mathcal{K}| \ll |\mathcal{M}|$, and the adversary is attempting a known-plaintext attack
  - Adversary knows pairs (m_i, c_i), such that: $c_i = \mathrm{Enc}_k(m_i)$
  - Adversary can pick a random key $k' \in \mathcal{K}$ and try to decrypt:
    - If for all i: $\mathrm{Dec}_{k'}(c_i) = m_i$, then key is correct with high probability
    - Success probability is ${}^{1}/_{|\mathcal{K}|}$ and running time is constant.
  - So we must allow for a break with some very small probability (i.e., negligible) and still call the scheme "secure"

Computationally Secure Encryption

Computationally Secure Encryption

- We need to add security parameter n in the definition

1ⁿ: a bit stream of n "1" bits

- Key-generation algorithm: Gen
  - We get the key k by invoking $\mathrm{Gen}(1^n)$, we assume $|k| \geq n$
- Encryption algorithm: Enc
  - Given a key k and a message $m \in \{0,1\}^*$, $\mathrm{Enc}_k(m)$ returns a ciphertext c
  - Enc may be randomized
- Decryption algorithm: Dec
  - Given a key k and a ciphertext c, $\mathrm{Dec}_k(c)$ outputs a message m
  - Dec is deterministic

- For correctness: $\mathrm{Dec}_k\big(\mathrm{Enc}_k(m)\big) = m$

Recall: Perfectly Secret Encryption

- Adversarial indistinguishability game:
  1. Adversary $\mathcal{A}$ chooses messages $m_0, m_1 \in \mathcal{M}$.
  2. Gen outputs random key k, and a random bit $b \in \{0,1\}$ is selected. Then ciphertext $c = \text{Enc}_k(m_b)$ is sent to $\mathcal{A}$.
  3. Adversary $\mathcal{A}$ outputs bit $b' \in \{0,1\}$.
  4. The output is 1 if $b = b'$, and 0 otherwise. If the output is 1 we say adversary $\mathcal{A}$ is successful.

- Definition 4 (Equivalent to Def. 1):
  – *An encryption scheme (Gen, Enc, Dec) is* perfectly secret *if for* ***every*** *adversary* $\mathcal{A}$:

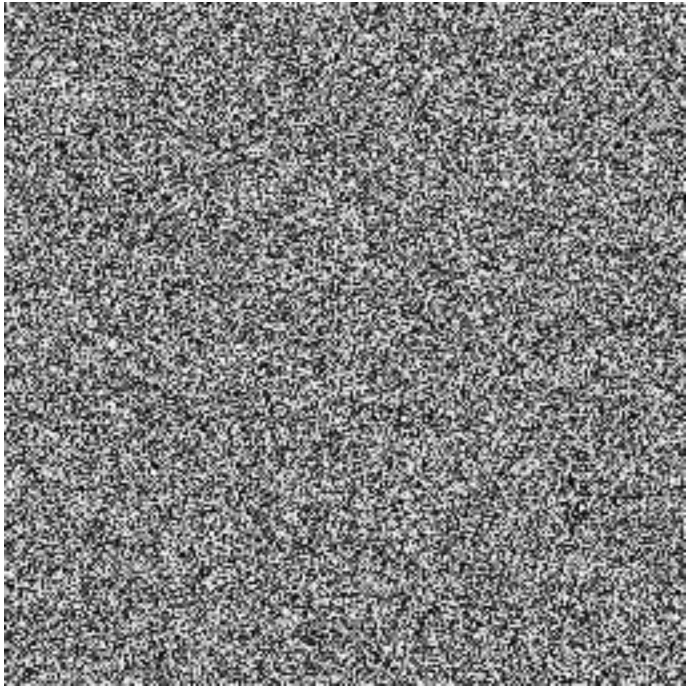
$$\Pr[\mathcal{A} \text{ is successful}] = {}^{1}/_{2}$$

Computationally Secure Encryption

- Eavesdropper indistinguishability game:
  1. Adversary $\mathcal{A}$ is given input 1^n and chooses messages $m_0, m_1 \in \mathcal{M}$ of the same length.
  2. $\mathrm{Gen}(1^n)$ outputs random key k, and a random bit $b \in \{0,1\}$ is selected. Then $c = \mathrm{Enc}_k(m_b)$, called the *challenge ciphertext*, is sent to $\mathcal{A}$.
  3. Adversary $\mathcal{A}$ outputs bit $b' \in \{0,1\}$.
  4. The output is 1 if $b = b'$, and 0 otherwise. If the output is 1 we say adversary $\mathcal{A}$ is successful.
- Definition:
  – *An encryption scheme (*Gen, Enc, Dec*) has* indistinguishable encryption (in the presence of an eavesdropper) *if for **all PPT** adversaries $\mathcal{A}$ there exists a negligible function* negl *such that:*

$$\Pr[\,\mathcal{A} \text{ is successful}] \leq {}^{1}/_{2} + \mathrm{negl}(n)$$

Pseudorandomness



Pseudorandomness

- Roughly speaking, a pseudorandom string "looks" random as long as the "looking" entity runs in PPT.
  - Any PPT algorithm cannot distinguish between a truly random string and a pseudorandom string.

- Remarks:
  - Pseudorandomness is a computational relaxation of randomness
  - If we say a string is pseudorandom we really mean: the process generating this string is pseudorandom

Pseudorandomness & Encryption

- Intuition:
  - With the one-time pad, the ciphertext was truly random because the key was picked uniformly at random
  - If we use pseudorandomness we can make the ciphertext look random to any PPT adversary
  - If we can generate a long pseudorandom string from a small truly random seed, a small key will suffice

Pseudorandom Generators (PRGs)

- Roughly speaking:
  - A PRG is a deterministic algorithm that receives a short truly random seed and stretches it into a long pseudorandom string.
  - In other words, a PRG uses a small amount of true randomness in order to generate a large amount of pseudorandomness.

- To formally define PRGs, we will need the notion of distinguishers
  - A distinguisher D is an algorithm which receives an input string u and outputs a bit $b \in \{0,1\}$. Its goal is to determine whether its input u is random string.

PRGs

- Definition:
  - *Let $\ell(\cdot)$ be a polynomial and G be a deterministic PPT algorithm such that for any input $s \in \{0,1\}^n$: G outputs a string of length $\ell(n)$. G is a* pseudorandom generator *if:*
    1. *(Expansion) For every n, we have: $\ell(n) > n$*
    2. *(Pseudorandomness) For all PPT distinguishers D, there exist a negligible function* negl *such that:*

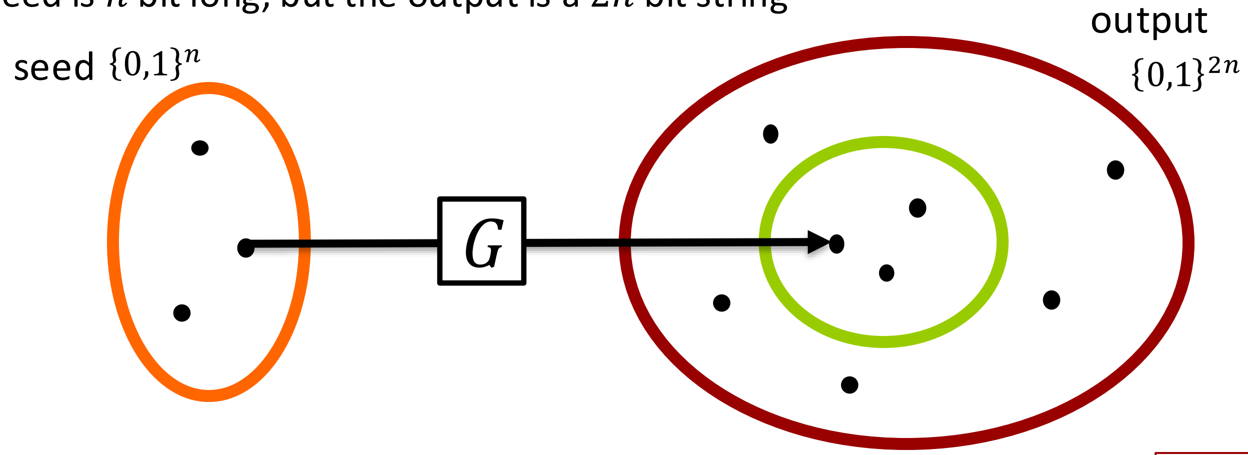
$$\left|\Pr[D(r) = 1] - \Pr\big[D\big(G(s)\big) = 1\big]\right| \leq \mathrm{negl}(n),$$

       where $r \in \{0,1\}^{\ell(n)}$ and $s \in \{0,1\}^n$ are chosen uniformly at random.

  - *$\ell(\cdot)$ is called the expansion factor.*

PRGs: Example

- How random is the output of a PRG?
  - Suppose we have a PRG with $\ell(n) = 2n$
  - The seed is n bit long, but the output is a $2n$ bit string



  - Probability that a random output string is in the range of G: 2^{-n}

$2^n / 2^{2n} = 2^{-n}$

Secure Fixed-Length Encryption

- Let G be a PRG with expansion factor ℓ
- Let key k, message m, and ciphertext c
  - k is n bits; m and c are $\ell(n)$ bits

- Gen:
  - Generate a uniformly random n-bit key k, i.e., with probability 2^{-n}
- Enc:
  - Compute $c = G(k) \oplus m$
- Dec:
  - Compute $m = G(k) \oplus c$

Secure Fixed-Length Encryption

- Theorem:
  - If G is a PRG, then the fixed-length encryption scheme is secure, i.e., it has indistinguishable encryptions in the presence of an eavesdropper.

- Proof (by reduction):
  - If an adversary can break our fixed-length encryption scheme, then we can construct a distinguisher, i.e., an algorithm which can distinguish output of G from a truly random string.

Proof (Sketch)

- If we replace our G with a truly random generator, then our scheme is identical to the one-time pad.
  - And so no adversary can succeed with probability better other than ${}^{1}/_{2}$

- So, if a PPT adversary $\mathcal{A}$ can break the fixed-length encryption scheme, then the adversary **must be** implicitly distinguishing the output of G from a truly random string.
  - This is the key part of the proof, where we construct a distinguisher D using $\mathcal{A}$

- This implies that G is not a PRG, which contradicts our starting assumption.
- Therefore, the scheme is secure.

Computationally Secure Encryption (same as before)

- Eavesdropper indistinguishability game:
  1. Adversary $\mathcal{A}$ is given input 1^n and chooses messages $m_0, m_1 \in \mathcal{M}$ of the same length.
  2. $\text{Gen}(1^n)$ outputs random key k, and a random bit $b \in \{0,1\}$ is selected. Then $c = \text{Enc}_k(m_b)$, called the *challenge ciphertext*, is sent to $\mathcal{A}$.
  3. Adversary $\mathcal{A}$ outputs bit $b' \in \{0,1\}$.
  4. The output is 1 if $b = b'$, and 0 otherwise. If the output is 1 we say adversary $\mathcal{A}$ is successful.
- Definition:
  - *An encryption scheme (*Gen, Enc, Dec*) has* indistinguishable encryption (in the presence of an eavesdropper) *if for* ***all PPT*** *adversaries* $\mathcal{A}$ *there exists a negligible function* negl *such that:*

$$\Pr[\mathcal{A} \text{ is successful}] \leq {}^{1}/_{2} + \text{negl}(n)$$

Reduction

- We are given a PPT adversary $\mathcal{A}$
- Distinguisher D:
  - On input $u \in \{0,1\}^{\ell(n)}$, do the following:
    1. Run $\mathcal{A}(1^n)$ to obtain messages $m_0, m_1 \in \{0,1\}^{\ell(n)}$
    2. Select a random bit $b \in \{0,1\}$, and compute $\boldsymbol{c} = \boldsymbol{u} \oplus \boldsymbol{m_b}$
    3. Send ciphertext c to $\mathcal{A}$ and obtain bit b'. Output is 1 if $b = b'$, and 0 otherwise

- Observation:
  - If u is a random string then $\mathcal{A}$ is being run against the one-time pad, and so $\mathcal{A}$ will be successful with probability $^1/_2$
  - If $u = G(k)$, for some key k, then $\Pr\left[D\left(G(k)\right) = 1\right] = \Pr[\,\mathcal{A}\text{ is successful}\,]$

Reduction

- So far, we have a PPT adversary $\mathcal{A}$ and a distinguisher D
- Observation:
  - If u is a random string then $\mathcal{A}$ is being run against the one-time pad, and so $\mathcal{A}$ will be successful with probability $^1/_2$
  - If $u = G(k)$, for some key k, then $\Pr\left[D\big(G(k)\big) = 1\right] = \Pr[\,\mathcal{A} \text{ is successful}\,]$

Recall: PPT adversary $\mathcal{A}$ can break the encryption scheme:

$$\Pr[\,\mathcal{A} \text{ is successful}] > {}^1/_2 + \mathrm{negl}(n)$$

$$\left|\,\Pr\left[D\big(G(k)\big) = 1\right] - \Pr[D(r) = 1]\,\right| > {}^1/_2 + \mathrm{negl}(n) - {}^1/_2$$

$$\left|\,\Pr\left[D\big(G(k)\big) = 1\right] - \Pr[D(r) = 1]\,\right| > \mathrm{negl}(n)$$

Distinguisher D can tell a truly random string r and G(k) → G(k) is not a PRG

Computationally Secure Encryption

- In practice we need a few more steps:
  - variable output-length PRGs $\longrightarrow$ stream ciphers
  - pseudorandom permutations $\longrightarrow$ block ciphers
  - modes of operations
  - chosen-plaintext attacks $\longrightarrow$ non-deterministic encryption

References

- Jonathan Katz, and Yehuda Lindell. "Introduction to modern cryptography." CRC Press, 2014. Chapters 2 & 3. (Mostly chapter 3.)

Discussion Questions

1. What do you think:
   - Is the C++ rand() function or Java Random class secure PRGs?
   - Do you know of any alternatives?

2. Is it a good idea to roll your own crypto?
   - Design your own algorithms?
   - Implement standardized algorithms (e.g., AES) yourself?