Applied Cryptography and Computer Security
CSE 664 Spring 2020

Lecture 4: Symmetric Encryption

Department of Computer Science and Engineering
University at Buffalo

High-Level View

- Previously we talked about:
  - unconstrained adversary
  - achieving perfect secrecy by means of one-time pad
  - using entropy to measure information leakage
- In this lecture we:
  - take a computational approach
  - break the bounds of information-theoretic analysis
  - learn about modern design of encryption algorithms

Symmetric Encryption

- Information-theoretic or perfect security builds on the fact that an attacker doesn't have enough information to recover the message
  - all messages can still happen with original probabilities
- Computational security provides security only in the presence of "practical" adversaries
  - given unlimited resources, such algorithms can be broken
- There are two differences from our previous definition of security:
  - security only holds against adversaries that run in a feasible amount of time
  - adversaries can potentially succeed with a very small probability

Computational Security

- Security of a cipher can often be defined in one of the following ways:
  - exact numbers
    - a scheme is (t, ε)-secure if an adversary running for time at most t has probability of most ε in breaking the security of the scheme
      - what values of t and ε are reasonable today?
    - (t, ε)-security does not imply security in general

Computational Security

- Security of a cipher can be defined in one of the following ways:
  - asymptotic approach
    - cipher is described using a security parameter n
    - a scheme is secure if an efficient adversary has only negligible probability in breaking its security
      - adversary runs in probabilistic polynomial time (PPT)
      - honest parties must be polynomial time as well
    - security guarantees hold only for sufficiently large values of n
      - an adversary running for $2^{25} \cdot n^3$ cycles can break security with probability $2^{20} \cdot 2^{-n/4}$
- Which approach is more common or better?

More on $\mathsf{poly}(n)$ and $\mathsf{negl}(n)$

- We distinguish between polynomial and super-polynomial functions and normally are not very concerned with the exact complexity
  - any super-polynomial function can be made sufficiently large by appropriately setting the security parameter n
  - a negligible function is then the inverse of any super-polynomial function
    - such function can diminish with drastically different rates and should only be used with sufficiently large values of n
    - we assume that events occurring with negligible probability are so unlikely that they can be ignored

More on $\mathsf{poly}(n)$ and $\mathsf{negl}(n)$

- We will use the following closure properties:
  - let f_1 and f_2 be two arbitrary functions of the same type (polynomial, super-polynomial, or negligible)
  - function $f_3(n) = f_1(n) + f_2(n)$ has the same type as f_1 and f_2
  - also, function $f_4(n) = f_1(n) \cdot p(n)$ has the same type as f_1 for any polynomial $p(n)$

String Length and Running Time

- By polynomial-time algorithms we mean functions running in polynomial time in the length of their inputs
  - i.e., on input x, $f(|x|) = f(n)$ must be poly-time
  - since $n = |x| = \log x$, if $f(n)$ takes time x, it is exponential in n
  - if we want $\mathsf{poly}(n)$ and there are no other inputs, we write $f(1^n)$
  - example:
  - the same applies to the complexity of all other algorithms

String Length and Running Time

- By probabilistic algorithms we mean functions that can make unbiased coin tosses
  - choose a bit to be 0 with probability 1/2 and 1 with probability 1/2
  - flip as many coins as necessary
- How do we generate randomness in practice, without a coin?
- Is output of C function `rand()` random? function `random()`?

Computationally-Secure Encryption

- We now define a computationally secure symmetric key encryption scheme
  - a private-key encryption scheme consists of polynomial-time algorithms (Gen, Enc, Dec) such that
    1. Gen: on input the security parameter 1^n, outputs key k
    2. Enc: on input a key k and a message $m \in \{0,1\}^*$, outputs ciphertext c
    3. Dec: on input a key k and ciphertext c, outputs plaintext m
  - we write $k \leftarrow \mathsf{Gen}(1^n)$, $c \leftarrow \mathsf{Enc}_k(m)$, and $m := \mathsf{Dec}_k(c)$
    - this notation means that Gen and Enc are probabilistic and Dec is deterministic

Symmetric Encryption

- The above definition allows us to encrypt message of any length
- In practice, there are two types of symmetric key algorithms:
  - block ciphers
    - the key has a fixed size
    - prior to encryption, the message is partitioned into blocks of certain size
    - each block is encrypted and decrypted on its own
  - stream ciphers
    - the message is processed as a stream
    - pseudo-random generator is used to produce a long key stream from a short fixed-length key

Computationally-Secure Encryption

- Correctness requirement is the same as before
- Definition of security now differs
  - we first model a very weak adversary that observes only one ciphertext
  - recall that we model security using eavesdropping indistinguishability experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\mathcal{E}}$
  - there are three differences in the computational setting:
    1.
    2.
    3.

Computationally-Secure Encryption

- Experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\mathcal{E}}(n)$
  1. $\mathcal{A}$ is given 1^n and chooses two messages m_0, m_1 of the same length
  2. random key k is generated by $\mathsf{Gen}(1^n)$, and random bit $b \leftarrow \{0,1\}$ is chosen
  3. ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$
  4. $\mathcal{A}$ outputs bit b' as its guess for b
  5. experiment outputs 1 if $b' = b$ ($\mathcal{A}$ wins) and 0 otherwise

Computationally-Secure Encryption

- A private-key encryption scheme $\mathcal{E} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has indistinguishable encryptions in the presence of an eavesdropper if for every PPT adversary $\mathcal{A}$ there exists a negligible function negl such that

$$\Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\mathcal{E}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

- The default notion of secure encryption does not hide information about the plaintext length
  - in some cases, we want the length to be protected

Towards Computationally-Secure Encryption

- How do we meet this definition?
  - idea: substitute randomness with pseudorandomness
- What is pseudorandomness?
  - it refers to a distribution of strings rather than a single string
  - given a string, a polynomial-time adversary shouldn't be able to tell whether it was sampled using a distribution of pseudorandom strings or uniformly at random
- Pseudorandom strings are produced using a pseudorandom generators (PRG)
  - a PRG takes a fixed-length key, or seed, and produces a longer string

Pseudorandom Generator

- Let G be a (deterministic) algorithm that on input n-bit string s outputs a string of length $\ell(n)$
- G is a pseudorandom generator if the following is true:
  1. (expansion) for any n, output is longer than input: $\ell(n) > n$
  2. (pseudorandomness) any PPT distinguisher D can't tell the difference with non-negligible probability:

$$|\mathsf{Pr}[D(r) = 1] - \mathsf{Pr}[D(G(s)) = 1]| \leq \mathsf{negl}(n)$$

     where r and s are random strings of size $\ell(n)$ and n
     - this property completely fails if D is computationally unbounded
- The seed s must be treated similar to a key

Pseudorandom Generator Exercises

- Examples on the board

Secure Encryption Scheme

- Private-key encryption scheme for messages from $\mathcal{M} = \{0,1\}^\ell$
  - let PRG G have expansion factor ℓ
  - Gen: on input 1^n, randomly choose key $k \leftarrow \{0,1\}^n$
  - Enc: on input key $k \in \{0,1\}^n$ and message $m \in \{0,1\}^{\ell(n)}$, output ciphertext $c := G(k) \oplus m$
  - Dec: on input key $k \in \{0,1\}^n$ and ciphertext $c \in \{0,1\}^{\ell(n)}$, output message $m = G(k) \oplus c$
- Theorem: The above scheme has indistinguishable encryptions in the presence of an eavesdropper, assuming that G is a PRG

Proof Technique: Reduction

- Reductions are commonly used to prove that a problem is hard
  - suppose problem X is known or believed to belong to a class of "hard" problems
  - we want to prove that Y is also "hard" to solve
  - we construct an "efficient" algorithm to use a solution to problem Y to solve problem X
    - this algorithm is called reduction
  - this implies that Y is "at least as hard" as X and must be within the same class of hard problems
  - example:

Proofs by Reduction

- The actual proof can proceed as proof by contradiction:
  - assume to the contrary that Y can be solved efficiently
  - we use reduction from X to Y to solve X efficiently
  - this is impossible because X is hard $\Rightarrow$ contradiction
  - the assumption that Y can be solved efficiently must be wrong
- In our computational setting
  - hard means "cannot be solved by a polynomial-time adversary with more than negligible probability"
  - efficient means polynomial time

Proofs by Reduction

- Claim: Assuming that no PPT adversary can break construction X with non-negligible probability, our construction Y is secure (cannot be broken by any PPT adversary except with negligible probability)

- Security proof by reduction:

  1. suppose some PPT adversary $\mathcal{A}$ has advantage $\varepsilon(n)$ at breaking our scheme Y

  2. we construct efficient adversary $\mathcal{A}'$ that tries to solve X using $\mathcal{A}$ as a sub-routine

     - $\mathcal{A}'$ simulates environment for $\mathcal{A}$ on an instance of problem X

     - if $\mathcal{A}$ successfully breaks Y, we want $\mathcal{A}'$ to break X at least with probability $1/p(n)$

Proof by Reduction

- Security proof by reduction (cont.)
  3. if $\varepsilon(n)$ is not negligible, X is broken by $\mathcal{A}'$ with non-negligible probability $\varepsilon(n)/p(n) \Rightarrow$ contradiction
  4. it must be that $\varepsilon(n)$ is negligible for *any* $\mathcal{A}$
- In our private-key encryption scheme
  - security of X:
  - security of Y:

Proving Security of Our Encryption Scheme

- To build a proof, we need to construct a distinguisher designed to break the PRG
  - the distinguisher must draw its "security breaking abilities" from an adversary $\mathcal{A}$ attacking our encryption scheme
- Once this is done, we analyze its success and relate it to that of $\mathcal{A}$'s success in breaking encryption experiment

Beyond Simplified Model

- How do we encrypt
  - variable-length messages
  - multiple messages
- Variable-length messages
  - generate pseudorandom string of desired length
  - use variable output-length PRG
- Handling multiple messages is trickier
  - straightforward usage of one-message encryption schemes fails to achieve security
  - what is the definition "security" now?

Multiple Encryptions Security

- Multiple message eavesdropping experiment $\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\mathcal{E}}(n)$
  1. $\mathcal{A}$ is given 1^n and chooses two vectors of t messages M_0 and M_1
  2. random key k is generated by $\mathsf{Gen}(1^n)$, and random bit $b \leftarrow \{0,1\}$ is chosen
  3. ciphertext vector C is computed from M_b and is given to $\mathcal{A}$
  4. $\mathcal{A}$ outputs bit b' as its guess for b
  5. experiment outputs 1 if $b' = b$ ($\mathcal{A}$ wins) and 0 otherwise
- Similar to before, we want

$$\Pr[\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\mathcal{E}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n)$$

Multiple Encryptions Security

- Any deterministic encryption algorithm fails this definition of security
- There are two common ways to achieve multiple encryption security
  - use different portions of the stream for different messages
    - drawback: requires synchronization
  - make PRG take another randomizing parameter as $G(k, IV)$
    - drawback: requires stronger security properties from the PRG

Stream Ciphers

- Stream cipher algorithms
  - Linear Feedback Shift Registers (LFSR)
  - RC4
- Security of practical stream cipher algorithms is less understood than security of block ciphers
- Some implementations also incorrectly use them