Computational Security

CS 5158/6058 Data Security and Privacy

Spring 2018

Instructor: Boyang Wang

Limitations of OTP

- Key is as long as message
  - Cannot decide message size in advance
  - Why not share the message directly while sharing the key

- Use each key only once

$$
\begin{aligned}
c \oplus c' &= (m \oplus k) \oplus (m' \oplus k) = m \oplus (k \oplus k) \oplus m' \\
&= m \oplus \{0\}^\lambda \oplus m' = m \oplus m'
\end{aligned}
$$

$$k = m \oplus c$$

OTP is Optimal

- OTP is optimal for perfect secrecy
  - Key size is the <u>smallest</u> we can get

- If perfectly secret, then key space size $|\mathcal{K}|$ must be greater than or equal to message space size $|\mathcal{M}|$

- Prove $|\mathcal{K}| < |\mathcal{M}|$ cannot be perfectly secret
  - Uniformly distribution over message space $\mathcal{M}$
  - A ciphertext c occurs with non-zero probability

Computational Security

- OTP is optimal but still not practical

- Relax the security (but slightly weaker still sufficient)
  - Computational security instead of perfect security
  - Can win with 1/2+p, but p is extremely small
  - It will take a very, very, very long time.

- Not perfect but good enough for real applications
  - E.g., win with 1/2 + 1/1000000000 using 200 years

Computational Security

- Assumptions on an <u>adversary</u>:
  - Knows distribution over message space $\mathcal{M}$
  - Knows Enc and Dec algorithm, can eavesdrop
  - Does not know key k

  - Has <u>limited</u> computational power
    - Run efficient (polynomial-time) algorithms
  - Can win <u>negligibly better than 1/2</u>
    - Negligible probability: extremely small

Computational Security

- Big O notation:
  - The performance/complexity of an algorithm
  - O(n): n steps in worst case scenario
  - Keep dominating factor:
    - $n^3 + n^2 + n$ steps —> $O(n^3)$
    - n + log(n) steps —> O(n)
  - Remove constants:
    - 3n + 200 steps —> O(n)
    - 1000000*n steps —> O(n)
    - 100 steps —> O(1)

Computational Security

- Practice: what are the complexity of following algo?
  - Algo 1: $4n^3 + 20n^2$ steps
  - Algo 2: $3000000*n + 2^n$ steps
  - Algo 3: $20*\log(n) + 3*n$ steps

- Answer:
  - Algo 1: $O(n^3)$
  - Algo 2: $O(2^n)$
  - Algo 3: $O(n)$

Computational Security

- Polynomial-time (efficient) algorithm
  - Complexity: $O(n^k)$, $k>1$,
    - Algorithm takes n^k steps in worst case
  - E.g., $O(n^2)$, $O(n^5)$, $O(n^{20})$ are polynomial-time
  - E.g., $O(2^n)$ is not polynomial-time

- Probabilistic polynomial-time algorithm
  - Complexity is polynomial-time
  - Outputs are probabilistic

Negligible Function

- A negligible function is one that is asymptotically smaller than any inverse polynomial-time function.

Def. A function f is negligible if for every positive polynomial p there is an N such that for all integers $n > N$ it holds that $f(n) < \frac{1}{p(n)}$.

- E.g., $2^{-n}, 2^{-\sqrt{n}}, n^{-\log n}$
- Approach zero given a large n

Negligible Function

$$2^{-n},\ 2^{-\sqrt{n}},\ n^{-\log n}$$

- Negligible functions approach zero at different rates
  - The min value of n, s.t. function is smaller than n^{-5}
    - $2^{-n} < n^{-5}$, get $n > 5\log n$, min is 23
    - $2^{-\sqrt{n}} < n^{-5}$, get $n > 25\log^2 n$, min is 3500
    - $n^{-\log n} < n^{-5}$, get $\log n > 5$, min is 33
  - Asymptotically, they are the same

Negligible Function

- Properties of negligible functions

Let $\mathtt{negl}_1$ and $\mathtt{negl}_2$ be negligible functions. Then

- If $\mathtt{negl}_3(n) = \mathtt{negl}_1(n) + \mathtt{negl}_2(n)$, $\mathtt{negl}_3(n)$ is negligible.
- For any positive polynomial p, if $\mathtt{negl}_4(n) = p(n) \cdot \mathtt{negl}_1(n)$, $\mathtt{negl}_4(n)$ is negligible.

Negligible Function

- Practice: Assume $p_1(n)$, $p_2(n)$ are negligible
  - Are the following algo. negligible?
  - Algo 1: $q(n) = p_1(n) - p_2(n) > 0$
  - Algo 2: $q(n) = 3000*p_1(n)$
  - Algo 3: $q(n) = 2^n*p_1(n)$

  - Algo 1: negligible
  - Algo 2: negligible
  - Algo 3: not negligible (2^n is not polynomial time)

Computational Security

- Security: for a sufficiently large n
  - Example: if adversary runs n^3 minutes can break a system with probability of $2^{40}2^{-n}$
  - $n=40$, 40^3 minutes (6 weeks), break with 1
  - $n=50$, 50^3 minutes (3 months), break with 1/1000
  - Practice: what if $n=500$?
    - 500^3 minutes (200 years), break with 2^{-460}
- Increase n (key length) to defend against the increase on computational power

Computational Security

- A faster computer makes adversary's job harder
- Example: An encryption scheme
  - Alice/Bob: encryption time 10^6n^2 cycles
  - Attacker: break with 10^8n^4 cycles with 2^{-n}
  - All use 2GHz computers, n=80(1GHz=10^9cycles/s)

2GHz, n=80	**Cycles**	**Time**	**Probability**
Alice/Bob	$80^2 10^6$	3.2 seconds	NA
Attacker	$80^4 10^8$	3 weeks	2^{-80}

Computational Security

- A faster computer makes adversary's job harder
- Example: An encryption scheme
  - Alice/Bob: encryption time 10^6n^2 cycles
  - Attacker: break with 10^8n^4 cycles with 2^{-n}
  - All update to 8GHz computers, n=160

8GHz, n=160	Cycles	Time	Probability
Alice/Bob	$160^2 10^6$	3.2 seconds	NA
Attacker	$160^4 10^8$	13 weeks	2^{-160}

Computational Security

- A faster computer makes adversary's job harder
- Practice: An encryption scheme
  - Alice/Bob: encryption time $10^6 n^2$ cycles
  - Attacker: break with $10^8 n^4$ cycles with 2^{-n}
  - 16GHz computers, n=240 (1GHz=10^9 cycles/s)

16GHz, n=240	Cycles	Time	Probability
Alice/Bob	?????	?????	NA
Attacker	?????	?????	?????

Computational Security

- A faster computer makes adversary's job harder
- Practice: An encryption scheme
  - Alice/Bob: encryption time 10^6n^2 cycles
  - Attacker: break with 10^8n^4 cycles with 2^{-n}
  - All update to 16GHz computers, n=240

16GHz, n=240	Cycles	Time	Probability
Alice/Bob	$240^2 10^6$	3.6 seconds	NA
Attacker	$240^4 10^8$	34 weeks	2^{-240}

Computational Security Game

- Ciphertexts of m_0, m_1 are <u>negligibly distinguishable</u>.

Given $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$, **security game** $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

1. Adversary $\mathcal{A}$ outputs m_0, $m_1 \in \mathcal{M}$ with $|m_0| = |m_1|$
2. Challenger flips a coin $b \in \{0, 1\}$, compute $c_b \leftarrow \mathsf{Enc}_k(m_b)$, where $k \leftarrow \mathsf{KeyGen}(1^n)$, an return c_b to $\mathcal{A}$
3. $\mathcal{A}$ guesses a bit b'
4. Output 1 if $b' = b$, otherwise 0; $\mathcal{A}$ wins if it is 1

Computational Security Game

- Random guess is 1/2, and can do <u>negligibly</u> better

Def. $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ is **indistinguishable** if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function $\texttt{negl}$ such that, for all n

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

- Adversary's advantage is <u>negligible</u>

$$\mathrm{Adv}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = \left| \Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - \frac{1}{2} \right| \leq \texttt{negl}(n)$$

Computational v.s. Perfect

	Computational security	Perfect security
A's computation power	Polynomial-time	Unlimited
Prob. of Win	1/2 + negligible	1/2
Practical?	Yes ($\|K\| << \|M\|$)	No ($\|K\| = \|M\|$)

Pseudorandom

- Pseudorandom
  - Is not random (i.e., not uniformly distributed)
  - Negligibly distinguish from random
  - More practical than random in crypto design

- Building Blocks for Symmetric-Key Encryption
  - Pseudorandom Generator (PRG)
  - Pseudorandom Function (PRF)

Pseudorandom Generator

- Pseudorandom Generator (PRG)
  - Efficient (polynomial-time), deterministic function
  - Use a <u>short</u> random string to generate a <u>long</u> pseudorandom string
  - Polynomial-time adversary can only negligibly distinguish PRG's output from random

Pseudorandom Generator

Def. Let $G(s) \to \{0,1\}^{l(n)}$ be a deterministic polynomial-time function, where $s \in \{0,1\}^n$. We say G is a PRG if

- **Expansion:** For every n, $l(n) > n$.
- **Pseudorandomness:** For any PPT algorithm D, there is a negligible function such that

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

  where s is uniformly selected from $\{0,1\}^n$ and r is uniformly selected from $\{0,1\}^{l(n)}$.

Pseudorandom Generator

- Why a PRG's output is not random (uniformly distributed)?
  - E.g. input size n and output size $l(n) = 2n$
  - Uniform distribution on $\{0,1\}^{2n}$, 2^{2n} strings with a probability 2^{-2n} each
  - Input $\{0,1\}^n$ and deterministic function, 2^n possible outputs at most; In other words, at least $2^{2n} - 2^n = 2^n(2^n - 1)$ strings in $\{0, 1\}^{2n}$ will not occur.

Pseudorandom Generator

- Practice: input: $n=20$ and output $l(n)=32$
  1. How many strings in $\{0, 1\}^{32}$?
  2. How many strings output by G(s), s in $\{0,1\}^{20}$?
  3. At least how many strings in $\{0, 1\}^{32}$ will not occur in the outputs of G(s)?

- (1) 2^{32} = 4,294,967,296; (2) 2^{20} = 1,048,576;
- (3) $2^{32} - 2^{20}$ = 4,293,918,720 or $(2^{32}-2^{20})/2^{32}$ = 0.99976

Additional Reading

Chapter 3, *Introduction to Modern Cryptography, Drs. J. Katz and Y. Lindell, 2nd edition*