AADS – Lecture 9

Algorithms in Cryptography and Other Miscellaneous Topics

Intro

On the menu today!

1. Algorithms and Cryptography (*potential exam material*)
    - Symmetric versus asymmetric
    - Meet-in-the-middle
    - RSA, exponentiation by squaring
    - Key sizes

Intro

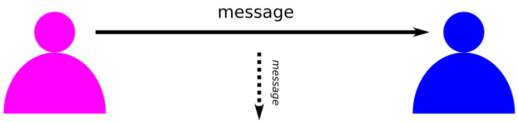
On the menu today!

1. Algorithms and Cryptography (*potential exam material*)
   - Symmetric versus asymmetric
   - Meet-in-the-middle
   - RSA, exponentiation by squaring
   - Key sizes
2. Miscellaneous advanced topics in AADS (*not exam material*)
   - Exponential algorithms
   - Approximate algorithms
   - Parallel algorithms
   - Space complexity
   - Average case complexity
   - Worst-case revisited
   - Space complexity

Algorithms in Cryptography

Why talk about crypto?

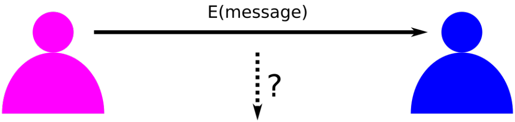
- Loaded with parallels to algorithms!
- Bridges between different disciplines/topics 👍👍👍
- Not a crypto class, different angle
- We simplify things



Problem

- Eavesdroppers can listen in on the channel
- We have to assume channel is insecure / message is "public"

Algorithms in Cryptography – Encryption



Problem

- Eavesdroppers can listen in on the channel
- We have to assume channel is insecure / message is "public"

Solution

- The sender *encrypts* the message
- Unintelligible with the correct *key*

Symmetric encryption

- Same key used to encrypt and decrypt
- "Padlock"
- Fast(er)
- In a network of n users, each pair need a distinct key: $n(n+1)/2 = O(n^2)$ keys
- Security (typically) based on established principles
- No proof of security! But not broken after many years...
- AES, 3DES, Blowfish, ChaCha20, ...

Asymmetric encryption

- A *public* key to encrypt, a *private* key to decrypt
- "Mailbox"
- Slow(er)
- Each user has only one (pair of) key(s), no matter how many other agents
- Security (typically) based on hardness assumptions
- Reduction proofs (e.g. reducing to $P = NP$)
- RSA, ElGamal, Pailler, ...

Breaking Symmetric Encryption

- Finding a weakness in the cipher
- A lot of (smart, well-motivated) people have *really* tried
- No proof, but well understood *structures*
- Trust built over time

Breaking Symmetric Encryption

- Finding a weakness in the cipher
- A lot of (smart, well-motivated) people have *really* tried
- No proof, but well understood *structures*
- Trust built over time

Guessing the Key for a Given Instance

- Keys are k bits (e.g. $k = 128$ for AES)
- All keys equally likely
- All keys produce independent plaintext-ciphertext relationships
- Search through all 2^{128} possibilities
- $T(k) = 2T(k-1)$
- "Exponential algorithm"

Double Encryption

- Encrypt a message twice (with different keys)
- Rationale: larger search space, "patches" cipher if weakness, or if poorly implemented
- Is it useful? Yes and no...

Double Encryption

- Encrypt a message twice (with different keys)
- Rationale: larger search space, "patches" cipher if weakness, or if poorly implemented
- Is it useful? Yes and no...

Meet-in-the-middle Attack

- From a pair x, y such that $y = E_{K_2}(E_{K_1}(x))$
- Compute candidates ciphertexts $E_{K1}(x)$ for all $K_1 \in \{0,1\}^k$
- Insert them in a dictionary
- Compute candidates plaintexts $E_{K_2}^{-1}(y)$ for all $K_2 \in \{0,1\}^k$
- For each, lookup (expected $O(1)$) in dictionary for collision
- Collision found = correct key pair found!

Modular Exponentiation

- Computation of the form $c = b^e \pmod{N}$
- Example: $4^3 = 64 = 4 \pmod{15}$

Modular Exponentiation

- Computation of the form $c = b^e \pmod{N}$
- Example: $4^3 = 64 = 4 \pmod{15}$
- What about $284028164950672937520345 70^{123456789}$ $\pmod{70329520026147529388 25}$...?
- Requires $O(e)$ (long) multiplications $\rightarrow$ expensive!

Modular Exponentiation

- Computation of the form $c = b^e \pmod{N}$
- Example: $4^3 = 64 = 4 \pmod{15}$
- What about $284028164950672937520345701^{23456789}$ (mod 7032952002614752938825)...?
- Requires $O(e)$ (long) multiplications $\rightarrow$ expensive!

Solution: Square-and-Multiply

- Exponent in binary form: $e = \overline{e_n \ldots e_1 e_0}$
- From $c = 1$, for each bit in the exponent:
  1. If e_0, $c \leftarrow c \cdot b \bmod N$
  2. In any case, $c \leftarrow c^2 \bmod N$, $e \leftarrow e \gg 1$
- Requires $O(\log e)$ “short” multiplications $\rightarrow$ OK!

S&M makes modular exponentiation feasible on large numbers!

Addition-Chain Exponentiation

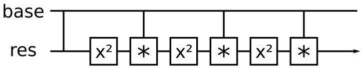
- Can we do better than Square-and-Multiply?
- Example: x^{15}

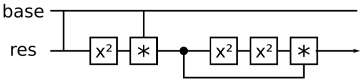
Addition-Chain Exponentiation

- Can we do better than Square-and-Multiply?
- Example: x^{15}
- Square-and-Multiply: $x \cdot (x \cdot (x \cdot x^2)^2)^2$ (6 multiplications)

Addition-Chain Exponentiation

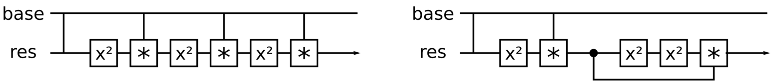
- Can we do better than Square-and-Multiply?
- Example: x^{15}
- Square-and-Multiply: $x \cdot (x \cdot (x \cdot x^2)^2)^2$ (6 multiplications)
- Optimal: $x^3 \cdot ((x^3)^2)^2$ (5 multiplications)





Addition-Chain Exponentiation

- Can we do better than Square-and-Multiply?
- Example: x^{15}
- Square-and-Multiply: $x \cdot (x \cdot (x \cdot x^2)^2)^2$ (6 multiplications)
- Optimal: $x^3 \cdot ((x^3)^2)^2$ (5 multiplications)



But... finding optimal is hard! Also: only constant factor reduction
Exercise: can DP help here?

RSA

- $N = pq$ with p, q two (large) primes
- Compute $\lambda(N) = lcm(p-1, q-1)$
- Choose e (small) coprime to $\lambda(N)$ and compute $d = e^{-1} \bmod \lambda(N)$; $pk = (e, N)$, $sk = (d, p, q, \lambda(N))$

RSA

- $N = pq$ with p, q two (large) primes
- Compute $\lambda(N) = lcm(p-1, q-1)$
- Choose e (small) coprime to $\lambda(N)$ and compute $d = e^{-1} \bmod \lambda(N)$; $pk = (e, N)$, $sk = (d, p, q, \lambda(N))$
- $c = m^e \bmod N$
- $m = c^d \bmod N$

RSA

- $N = pq$ with p, q two (large) primes
- Compute $\lambda(N) = lcm(p-1, q-1)$
- Choose e (small) coprime to $\lambda(N)$ and compute $d = e^{-1} \bmod \lambda(N)$; $pk = (e, N)$, $sk = (d, p, q, \lambda(N))$
- $c = m^e \bmod N$
- $m = c^d \bmod N$
- $m = m^{ed} \bmod N$ because $ed = 1 \bmod \lambda(N)$ ("Carmichael's generalization of Euler's theorem")

RSA

- $N = pq$ with p, q two (large) primes
- Compute $\lambda(N) = lcm(p-1, q-1)$
- Choose e (small) coprime to $\lambda(N)$ and compute $d = e^{-1} \bmod \lambda(N)$; $pk = (e, N)$, $sk = (d, p, q, \lambda(N))$
- $c = m^e \bmod N$
- $m = c^d \bmod N$
- $m = m^{ed} \bmod N$ because $ed = 1 \bmod \lambda(N)$ ("Carmichael's generalization of Euler's theorem")

Security

- (Assumed) hardness of *RSA problem* (e-th root modulo N)
- (Assumed) hardness of *factoring large composites* (sufficient)

Algorithms in Cryptography – Key Sizes

Symmetric Keys

- Each key in the key space is possible and a priori equally likely
- Brute-force attack takes up to 2^k steps
- We choose k to be well over what computers can achieve today (but not *too* much – why?); for AES, $k = 128$

Algorithms in Cryptography – Key Sizes

Symmetric Keys

- Each key in the key space is possible and a priori equally likely
- Brute-force attack takes up to 2^k steps
- We choose k to be well over what computers can achieve today (but not *too* much – why?); for AES, $k = 128$

Asymmetric Keys

- Not all keys in key space possible
- Some attacks better than brute force, depending on scheme
- For RSA: "general number field sieve" for integer factorization (complexity: $O(\exp((\sqrt[3]{64/9} + o(1))(\ln N)^{\frac{1}{3}}(\ln \ln N)^{\frac{2}{3}})))$
- To remain "equivalent" to 2^{128}, N is 2048 or 4096 bits

Miscellaneous Topics

Miscellaneous Topics – Exponential Algorithms

Algorithms we discussed are "efficient" (some more than others)

Miscellaneous Topics – Exponential Algorithms

Algorithms we discussed are “efficient” (some more than others)
Generally, “efficient” means sub-exponential (depends on context)

Miscellaneous Topics – Exponential Algorithms

Algorithms we discussed are "efficient" (some more than others)
Generally, "efficient" means sub-exponential (depends on context)
Some problems are hard: $\nexists$ efficient algorithm (conjecture)

Miscellaneous Topics – Exponential Algorithms

Algorithms we discussed are "efficient" (some more than others)
Generally, "efficient" means sub-exponential (depends on context)
Some problems are hard: $\nexists$ efficient algorithm (conjecture)

Example 1: Traveling Salesman Problem (TSP)

- Given $G = (V, E)$, find $\pi = (v_1, \ldots, v_n)$, a permutation of V, that minimizes $C = \sum_{i=1}^{n} w(v_i, v_{i+1}) + w(v_n, v_1)$
- "Shortest path that visits all cities"

Algorithms we discussed are "efficient" (some more than others)
Generally, "efficient" means sub-exponential (depends on context)
Some problems are hard: $\nexists$ efficient algorithm (conjecture)

Example 1: Traveling Salesman Problem (TSP)

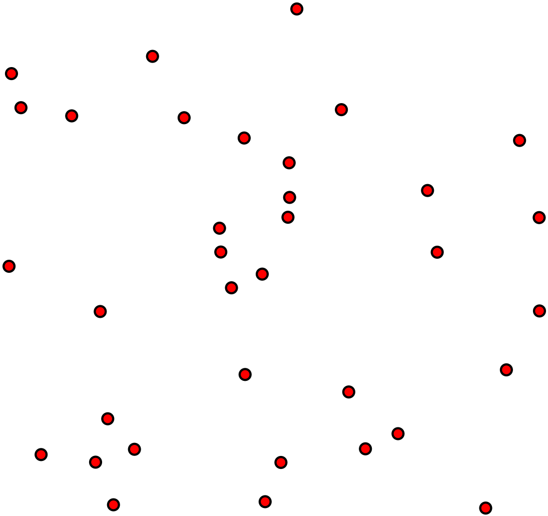
- Given $G = (V, E)$, find $\pi = (v_1, \ldots, v_n)$, a permutation of V, that minimizes $C = \sum_{i=1}^{n} w(v_i, v_{i+1}) + w(v_n, v_1)$
- "Shortest path that visits all cities"
- Naive algorithm: try all π, compute C for each $\rightarrow O(n \cdot n!)$

Algorithms we discussed are "efficient" (some more than others)
Generally, "efficient" means sub-exponential (depends on context)
Some problems are hard: $\nexists$ efficient algorithm (conjecture)

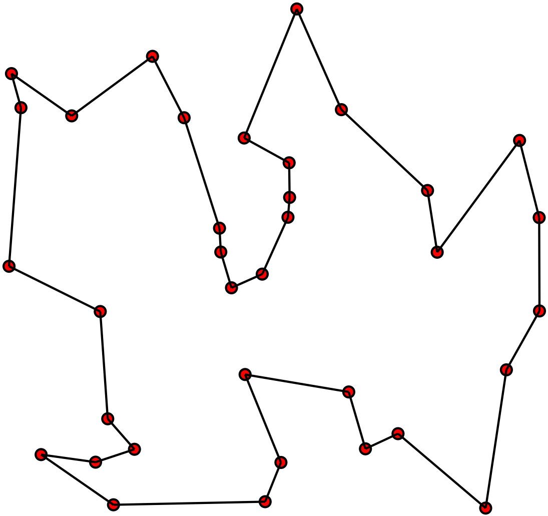
Example 1: Traveling Salesman Problem (TSP)

- Given $G = (V, E)$, find $\pi = (v_1, \ldots, v_n)$, a permutation of V, that minimizes $C = \sum_{i=1}^{n} w(v_i, v_{i+1}) + w(v_n, v_1)$
- "Shortest path that visits all cities"
- Naive algorithm: try all π, compute C for each $\rightarrow O(n \cdot n!)$
- Slightly better exists: $O(n^2 \cdot 2^n)$
- Also, other approaches (branch-and-bound, linear programming)... but nothing "efficient"

Miscellaneous Topics – Traveling Salesman Problem



Miscellaneous Topics – Traveling Salesman Problem



Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime

Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime
- Typically framed as *integer factorization* (finding p and q)
- Both are "difficult"

Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime
- Typically framed as *integer factorization* (finding p and q)
- Both are "difficult"
- … and it's a good thing!

Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime
- Typically framed as *integer factorization* (finding p and q)
- Both are “difficult”
- … and it’s a good thing!

What to do then?

- Small instances

Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime
- Typically framed as *integer factorization* (finding p and q)
- Both are "difficult"
- … and it's a good thing!

What to do then?

- Small instances
- Special cases

Example 2: RSA

- Find p given $c = p^e \bmod N$ where $N = pq$ semiprime
- Typically framed as *integer factorization* (finding p and q)
- Both are “difficult”
- … and it’s a good thing!

What to do then?

- Small instances
- Special cases
- Approximate solutions

Miscellaneous Topics – Approximate Algorithms

Sometimes, finding an approximate solution is "good enough"
Compromise between cost and quality of the approximation

Miscellaneous Topics – Approximate Algorithms

Sometimes, finding an approximate solution is “good enough”
Compromise between cost and quality of the approximation

Methods

- **Greedy** algorithm: e.g. in the coin change algorithm

Miscellaneous Topics – Approximate Algorithms

Sometimes, finding an approximate solution is “good enough”
Compromise between cost and quality of the approximation

Methods

- **Greedy** algorithm: e.g. in the coin change algorithm
- **Relaxation**: e.g. integer minimization $\rightarrow$ real minimization

Miscellaneous Topics – Approximate Algorithms

Sometimes, finding an approximate solution is “good enough”
Compromise between cost and quality of the approximation

Methods

- **Greedy** algorithm: e.g. in the coin change algorithm
- **Relaxation**: e.g. integer minimization $\rightarrow$ real minimization
- **Local search** and other incremental optimizations: e.g. starting with a random sampling of edges in TSP, and perturbing iteratively

Miscellaneous Topics – Approximate Algorithms

Sometimes, finding an approximate solution is “good enough”
Compromise between cost and quality of the approximation

Methods

- **Greedy** algorithm: e.g. in the coin change algorithm
- **Relaxation**: e.g. integer minimization $\rightarrow$ real minimization
- **Local search** and other incremental optimizations: e.g. starting with a random sampling of edges in TSP, and perturbing iteratively
- **Randomness**, sampling, etc.

Parallel Algorithms

- Some are naturally parallelizable, or even distributable
- Some are not ("serial"/"sequential")
- May matter in current computational paradigm

Parallel Algorithms

- Some are naturally parallelizable, or even distributable
- Some are not ("serial"/"sequential")
- May matter in current computational paradigm
- Limitations: intrinsic algorithmic sequencing, communication, balancing, etc.
- "Parallel slowdown"

Parallel Algorithms

- Some are naturally parallelizable, or even distributable
- Some are not ("serial"/"sequential")
- May matter in current computational paradigm
- Limitations: intrinsic algorithmic sequencing, communication, balancing, etc.
- "Parallel slowdown"

Example: Matrix Multiplication

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$
$$= \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

Space Complexity

- Memory required when executing an algorithm
- Analogous to time complexity (big O notation, etc.)
- Always bounded by time complexity! Exercise: why?

Space Complexity

- Memory required when executing an algorithm
- Analogous to time complexity (big O notation, etc.)
- Always bounded by time complexity! Exercise: why?

When Memory Matters!

- Sometimes memory is precious, or even limiting
- Already touched upon once aspect: *implicit data structures*
- In-place versus out-of-place (esp. for *sorting*)
  - e.g. Bubble sort versus Quicksort
- Time-space trade-offs: caching, compression, in crypto, etc.

Other related limitations: $\neq$ kinds of memory, network, etc.

Average Case Complexity

- Measure of the “average” case

Average Case Complexity

- Measure of the “average” case
- **. . . but what does average mean??**

Miscellaneous Topics – Average Case Complexity

Average Case Complexity

- Measure of the "average" case
- . . . **but what does average mean??**
- Needs assumption on *input distribution*

Average Case Complexity

- Measure of the "average" case
- ... **but what does average mean??**
- Needs assumption on *input distribution*
- You already tackled this in labs when *testing*
- Meaningful when average case $\neq$ worst case (e.g. Quicksort)

Average Case Complexity

- Measure of the "average" case
- **. . . but what does average mean??**
- Needs assumption on *input distribution*
- You already tackled this in labs when *testing*
- Meaningful when average case $\neq$ worst case (e.g. Quicksort)
- Amortized complexity: *related* but different

Average Case Complexity

- Measure of the “average” case
- **… but what does average mean??**
- Needs assumption on *input distribution*
- You already tackled this in labs when *testing*
- Meaningful when average case $\neq$ worst case (e.g. Quicksort)
- Amortized complexity: *related* but different

Examples

- Sorting: all permutations equally likely
- BST Insert: all *possible* BSTs of a given size n
  - Potentially very hard to consider
  - Often: random uniform *sampling* (with care...)

Why Worst Case so Prevalent?

- Easier to work with
- In many cases, *tight* bound
- Sometimes it is precisely what we want to capture
- Real-time and user-facing applications
  - User experience: e.g. web, apps, etc.
  - Safety: e.g. air/train/road traffic signalling
  - Cascading effects: e.g. scheduling