review, crypto
FINGERPRINTING

Alice

Bob

Disk

Disk
STRING MATCHING

PATTERN

CORPUS
RELIABLE COMMUNICATION
GOAL:

DEVISE A RELIABLE METHOD FOR NODES TO SEND MESSAGE TO THE SERVER WITH AS LITTLE COORDINATION AS POSSIBLE.
**SIMPLE ALGORITHM**

1. At time $t$, flip a coin that is heads with probability $\frac{1}{n}$. If heads, then broadcast. If success, then stop.

2. Else wait and try again.

3. Repeat $c n \log n$ times.
ANALYZE THE SIMPLE ALGORITHM
$S_{i,t} =$ event that node $i$ succeeds in sending its message at time $t$

$\Pr[S_{i,t} = 1] = (\frac{1}{n}) \left[ 1 - \frac{1}{n} \right]^{n-1} \prod_{j \neq i}$

heads for $i$
\[ \Pr[S_{i,t} = 1] = \frac{1}{n} \left( 1 - \frac{1}{n} \right)^{n-1} \sim \frac{1}{n \cdot \varepsilon} \]
FACT: IF \[ f(n) = \left(1 - \frac{1}{n}\right)^n \] THEN
FACT: IF \[ f(n) = \left(1 - \frac{1}{n}\right)^n \] THEN \[ \sim \frac{1}{e} \]
\[ S_{i,t} = \text{NODE } i \text{ SUCCEEDS IN SENDING AT TIME } t \]

\[ \frac{1}{en} \leq \Pr[S_{i,t} = 1] \leq \frac{1}{2n} \]
Failure

$$F_{i,t} = \text{probability that } i \text{ fails } @ \text{ times } 1, 2, 3, \ldots, t$$
Failure

\[ F_{i,t} = \text{node } i \text{ fails to send at times } 1,2,\ldots,t \]

\[ \Pr[F_{i,t}] = \bigwedge_{j=1}^{t} \Pr[S_{i,j}] \]
FAILURE

$F_{i,t} =$ \hspace{1cm} \text{NODE } i \text{ FAILS TO SEND AT TIMES } 1,2,...,t$

$\Pr[F_{i,t}] = \bigwedge_{j=1}^{t} \Pr[S_{i,j}] = \prod_{j=1}^{t} \Pr[S_{i,j}]$
\( \Pr[F_{i,t}] = \bigwedge_{j=1}^{t} \Pr[S_{i,j}] = \prod_{j=1}^{t} \Pr[S_{i,j}] \)
**FAILURE**

\[ F_{i,t} = \text{NODE } i \text{ FAILS TO SEND AT TIMES } 1,2,\ldots,t \]

\[ \Pr[F_{i,t}] = \bigwedge_{j=1}^{t} \Pr[S_{i,j}] = \prod_{j=1}^{t} \Pr[S_{i,j}] < \left(1 - \frac{1}{2n}\right)^{\frac{t}{n}} \]

**FOR**

\[ t = O(n \ln n) \]

\[ \Pr[F_{i,t}] = n^{-c} \]
ALL FAIL

\[ F_t = \]

\[ \Pr[F_t] = \]
\( F_t = \) SOME NODE \( i \) FAILS TO SEND AT TIMES 1, 2, ..., \( t \)

\[
\Pr[F_t] = \bigvee_{i=1}^{n} \Pr[F_{i,t}]
\]
\[ F_t = \text{SOME NODE } i \text{ FAILS TO SEND AT TIMES } 1, 2, \ldots, t \]

\[ \Pr[F_t] = \bigvee_{i=1}^{n} \Pr[F_{i,t}] \leq \sum_{i=1}^{n} \Pr[F_{i,t}] \leq \sum_{i=1}^{n} n^{-c} \]
SUMMARY

AT TIME T, FLIP A COIN THAT IS HEADS WITH PR \(\frac{1}{n}\).

IF HEADS, THEN BROADCAST. IF SUCCESS, THEN STOP.

ELSE WAIT AND TRY AGAIN.

REPEAT \(O(n \ln n)\) TIMES

WITH PROBABILITY

EVERY NODE SUCCEEDS IN SENDING MESSAGE.
TOOLS WE USED

\[ \left( 1 - \frac{1}{n} \right)^n \]

ANALYSIS OF

PROBABILITY THAT MANY INDEPENDENT EVENTS ALL OCCUR:

PROBABILITY THAT ONE OUT OF N EVENTS OCCURS:
SECOND EXAMPLE:

MEDIAN
SELECT \((i, A[1, \ldots, n])\)

PICK FIRST ELEMENT

PARTITION LIST ABOUT THIS ONE

IF PIVOT IS POSITION \(i\), RETURN PIVOT

ELSE IF PIVOT IS IN POSITION \(> i\)

ELSE SELECT \((i, A[1, \ldots, p - 1])\)

ELSE SELECT \(((i - p - 1), A[p + 1, \ldots, n])\)
PROBLEM: WHAT IF WE ALWAYS PICK BAD PARTITIONS?
PARTITION \((A[1, \ldots, n])\)

\[B[1, \ldots, \lceil n/5 \rceil]\]

SELECT \((\lceil n/5 \rceil / 2, B[1, \ldots, \lceil n/5 \rceil])\)
A NICE PROPERTY OF OUR PARTITION

\[ 3 \left( \left\lceil \frac{1}{2} \left\lfloor \frac{n}{5} \right\rfloor \right\rceil - 2 \right) \geq \frac{3n}{10} - 6 \]

THIS IMPLIES THERE ARE
AT MOST \( \frac{7n}{10} + 6 \) NUMBERS LARGER THAN 
/ SMALLER
**SELECT** \((i, A[1, \ldots, n])\)

**PICK FIRST ELEMENT**

**PIVOT = PARTITION** \((A[1, \ldots, n])\)

**IF** PIVOT IS POSITION \(i\), **RETURN** PIVOT

**ELSE IF** PIVOT IS IN POSITION \(i\), **SELECT** \((i, A[1, \ldots, p - 1])\)

**ELSE** **SELECT** \(((i - p - 1), A[p + 1, \ldots, n])\)

\[
S(n) = S\left(\left\lceil n/5 \right\rceil\right) + O(n) + S\left(7n/10 + 6\right)
\]

\[\Theta(n)\]
**RandomizedSelect**  
\[(i, A[1, \ldots, n])\]

- **Pick random partition element**
- **Partition list about this one**
  - If pivot is position \(i\), return pivot
  - Else if pivot is in position \(i > \) select \((i, A[1, \ldots, p - 1])\)
  - Else select \(((i - p - 1), A[p + 1, \ldots, n])\)
**RandomizedSelect**  
\( (i, A[1, \ldots, n]) \)

- Pick random partition element
- Partition list about this one
- ....

![Diagram](image-url)
RUNNING TIME ANALYSIS

RECURSIVE CALLS
PHASES

1

\[
n \cdot \left( \frac{3}{4} \right) \circ
\]

2

\[
\leq \frac{3}{7}
\]

3

\[
\leq \frac{3}{4}
\]

\[
\leq \frac{3}{4}
\]
PHASES

Algorithm is in phase $j$ if

$$ \left( \frac{3}{4} \right) ^ j n $$

Size of input list is $<$


**RandomizedSelect**

\((i, A[1, \ldots, n])\)

**Pick Random Partition Element**

**Partition List About This One**

....
$X_j = \text{NUMBER OF STEPS IN PHASE J}$

$\mathbb{E}[X_j] = $
$X_j = \text{NUMBER OF STEPS IN PHASE J}$

$$E[X_j] = \sum_{j=0}^{\infty} j \cdot \Pr[X_j = j]$$

$\Pr[X_j = 1] =$

$\Pr[X_j = 2] =$

$\Pr[X_j = j] =$
LINEARITY OF EXPECTATION

\[ \forall X, Y, \quad \mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y] \]
EXPECTED RUNNING TIME

\[ E[X] = \Theta(n) \]
PRIVATE COMMUNICATION

Alice

Eve

Bob
null
SUBSTITUTION CIPHER

\[ \mathcal{M} = \{A, B, \ldots, Z\}^* \]
\[ \mathcal{K} = \text{the set of permutations over } \{A, B, \ldots, Z\} \]
\[ \text{Gen} = k \text{ where } k \leftarrow \mathcal{K}. \]
\[ \text{Enc}_k(m_1m_2\ldots m_n) = c_1c_2\ldots c_n \text{ where } c_i = k(m_i) \]
\[ \text{Dec}_k(c_1c_2\ldots c_n) = m_1m_2\ldots m_n \text{ where } m_i = k^{-1}(c_i) \]

SIZE OF KEYSPACE IS

\[ 26! = 403291461126605635584000000 \]
FREQUENCY ANALYSIS
RSA

1. Modular exponentiation \( p\times q \)
2. Greatest common divisor \( \gcd \)
3. Picking large prime \( n \)
4. Euler's theorem

Public key encryption:

\[ \text{Enc}(\text{pk}, m) \rightarrow c \]

\[ \text{Dec}(\text{sk}, c) \rightarrow m \]
MOD-EXP

$(a, x, n) \rightarrow a^x \mod n$

$\text{1000 bits}$

$12$

$a$

$a^x \mod n$
Thus, we can write
\[ b^x + (a \mod b)^y = d \]
and by adding 0 to the right, and regrouping, we get
\[ d = b^{x + (a \mod b)^y} + (a \mod b)^y + b^{(a \mod b)^y} = b^{(x + (a \mod b)^y)} + ay \]
which shows that the return value \((y, x + (a \mod b)^y)\) is correct.

The assumption that the inputs are such that \(a > b\) is without loss of generality since otherwise the first recursive call swaps the order of the inputs.

Exponentiation modulo \(n\)

Given \(a, x, n\), we now demonstrate how to efficiently compute \(a^x \mod n\).

Recall that by efficient, we require the computation to take polynomial time in the size of the representation of \(a, x, n\).

Since inputs are given in binary notation, this requires our procedure to run in time \(O(\log(a), \log(x), \log(n))\).

The key idea is to rewrite \(x\) in binary as
\[ x = 2^x x + 2^{x-1} x \cdots + 2^1 x + x^0 \]
where \(x_i \in \{0, 1\}\) so that
\[ a^x \mod n = a^{2^x x + 2^{x-1} x + \cdots + 2^1 x + x^0} \mod n \]

Algorithm 2: ModularExponentiation

Input: \(a, x \in \{1, n\}\)

1. \(r \leftarrow 1\)
2. While \(x > 0\) do
   3. If \(x\) is odd then
      4. \(r \leftarrow r \cdot a \mod n\)
   5. \(x \leftarrow \lfloor x/2 \rfloor\)
   6. \(a \leftarrow a^2 \mod n\)
3. Return \(r\)
Thus, we can write
\[ bx + (a \mod b) y = d \]
and by adding 0 to the right, and reordering, we get
\[ d = bx b^{\lceil a/b \rceil}y + (a \mod b)y + b^{\lceil a/b \rceil}y = b^{\lceil x \rceil (a/b)} + ay \]
which shows that the return value \((y, x^{\lceil a/b \rceil}y)\) is correct.

The assumption that the inputs are such that \(a > b\) is without loss of generality since otherwise the first recursive call swaps the order of the inputs.

Exponentiation modulo \(n\)

Given \(a, x, n\), we now demonstrate how to efficiently compute \(a^x \mod n\).

Recall that by efficient, we require the computation to take polynomial time in the size of the representation of \(a, x, n\).

Since inputs are given in binary notation, this requires our procedure to run in time \(\text{poly}(\log(a), \log(x), \log(n))\).

The key idea is to rewrite \(x\) in binary as
\[ x = 2^x x + 2^1 x + \cdots + 2^1 x + x^0 \]
where \(x_i \in \{0, 1\}\) so that
\[ a^x \mod n = a^{2^x x} + 2^1 x + \cdots + 2^1 x + x^0 \mod n \]
We show this can be further simplified as
\[ a^x \mod n = \prod_{i=0}^{\log n} x_i a^{2^i} \mod n \]

Algorithm 2: ModularExponentiation\((a, x, n)\)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( r \leftarrow 1 )</td>
</tr>
<tr>
<td>2</td>
<td>( \text{while } x &gt; 0 \text{ do} )</td>
</tr>
<tr>
<td>3</td>
<td>( \text{if } x \text{ is odd then} )</td>
</tr>
<tr>
<td>4</td>
<td>( r \leftarrow r \cdot a \mod n )</td>
</tr>
<tr>
<td>5</td>
<td>( x \leftarrow \lceil x/2 \rceil )</td>
</tr>
<tr>
<td>6</td>
<td>( a \leftarrow a^2 \mod n )</td>
</tr>
<tr>
<td>7</td>
<td>( \text{Return } r )</td>
</tr>
</tbody>
</table>
**MOD-EXP**

\[(a, x, n) \rightarrow a^x \mod n \quad \ell\]

\[a^x \mod n = \prod_{i=0}^{\ell} x_i a^{2^i} \mod n\]

---

**Algorithm 2: ModularExponentiation** \( \text{ModularExponentiation}(a, x, n) \)

**Input:** \( a, x \in [1, n] \)

1. \( r \leftarrow 1 \)
2. \( \textbf{while } x > 0 \textbf{ do} \)
3.   \( \textbf{if } x \text{ is odd then} \)
4.     \( r \leftarrow r \cdot a \mod n \)
5.   \( x \leftarrow \lceil x/2 \rceil \)
6.   \( a \leftarrow a^2 \mod n \)
7. \( \text{Return } r \)
greatest common divisor of

$$\gcd(35 \text{ and } 14) = 7$$
what is the GCD?
Algorithm 1: ExtendedEuclid(a, b)

- **Input**: (a, b) s.t a > b ≥ 0
- **Output**: (x, y) s.t. ax + by = gcd(a, b)

1. if a mod b = 0 then
   2. Return (0, 1)
2. else
   4. (x, y) ← ExtendedEuclid(b, a mod b)
   5. Return (y, x − y(⌊a/b⌋))

GIVEN (A,B):

FINDS (X,Y) S.T. AX + BY = GCD(A,B)
35 and 14
\[
gcd(35, 14) = 7
\]

13 and 73
\[
gcd(13, 73) = 1
\]

(1) \[\overline{35} + (14)(-2) = 7\]

(13)(-28) + (73) \cdot 5 = 1

\[
\begin{align*}
365 \\
-364 \\
1
\end{align*}
\]
CRYPTOGRAPHY

32964031794323944819653393490459747322286350
31500646399521148595996590847768392238771217
69252874938669758963521262177684757622917354
10764395167469005450386721087598087995167019
51260209070780169584330401159403323161691626
51931932385937935848982371478700671595968131
07098610562722922433990122345442992245859824
74364293651925019779584845838833700838150940
56504167483874319231730153624474523841938831
33113697736378643670286581890300666191500953
329742364829

LARGE PRIME NUMBER
import java.io.*;
import java.math.*;
import java.util.*;

public class pr {
    public static void main(String args[]) {
        BigInteger prime = new BigInteger(1500, 80, new Random());
        System.out.println("prime is "+prime);
    }
}
RABIN-MILLER

$$L_N = \{\alpha \in \mathbb{Z}_N \mid \alpha^{N-1} = 1 \text{ and if } \alpha^{u2^j+1} = 1 \text{ then } \alpha^{u2^j} = 1\}$$
2.5. Basic Computational Number Theory

Then a prime number is usually not equal to 1. The first fact and the second base case $N = 2$ may be verified by the following procedure.

Algorithm 3: Miller-Rabin Primality Test

1. Handle base case $N = 2$
2. For $t$ times do
3. Pick a random $\alpha \in \mathbb{Z}_N$
4. If $\alpha \notin L_N$ then Output "composite"
5. Output "prime"

$L_N = \{ \alpha \in \mathbb{Z}_N | \alpha^{N-1} = 1 \text{ and if } \alpha^{u2^j+1} = 1 \text{ then } \alpha^{u2^j} = 1 \}$
RABIN-MILLER

\[ L_N = \{ \alpha \in \mathbb{Z}_N \mid \alpha^{N-1} = 1 \text{ and if } \alpha^{u2^j+1} = 1 \text{ then } \alpha^{u2^j} = 1 \} \]

**Algorithm 3:** Miller-Rabin Primality Test

1. Handle base case \( N = 2 \)
2. for \( t \) times do
   3. Pick a random \( \alpha \in \mathbb{Z}_N \)
   4. if \( \alpha \notin L_N \) then Output “composite”
5. Output “prime”

**Theorem 38.1.** If \( N \) is composite, then the Miller-Rabin test outputs “composite” with probability \( 1 - 2^{-t} \). If \( N \) is prime, then the test outputs “prime.”
EULER TOTIENT

\[ \Phi(n) : \# \text{ of integers that are smaller and relatively prime to } n \]

\[ \Phi(7) = 1 2 3 4 5 6 7 \]

\[ \gcd(7, 1) = 1 \]

\[ \phi(p) = p - 1 \]
EULER TOTIENT

\[ \Phi(n) \]

\[ \phi(p) = p - 1 \]

\[ \phi(n) = \phi(p_1) \cdot \phi(p_2) \cdot \ldots \cdot \phi(p_k) \]

\[ \phi(15) = 8 = 2 \cdot 4 = \phi(3) \cdot \phi(5) \]

15

1 2 3 4 5 6 7 8 9 10 11 12 13 14

\[ \mathbb{Z}^*_n \]
EULER TOTIENT

$|\mathbb{Z}_n^*| = \Phi(n)$

- Prime: $\Phi(p) = p - 1$
- Product of 2 primes: $\Phi(n) = (p - 1)(q - 1)$
EULER THEOREM

If $\gcd(a, n) = 1$, then $a^{\Phi(n)} = 1 \mod n$.
\[ \phi(n) \]

\[ \forall a \in \mathbb{Z}_N^*, \ a^{\Phi(N)} = 1 \mod N \]
EULER THEOREM

\[ \forall a \in \mathbb{Z}_N^*, \ a^{\Phi(N)} = 1 \mod N \]

argue: all are distinct
spse two are equal.
multiply by \( a^{-1} \)
this implies 2=6!
EULER THEOREM

\[ \forall a \in \mathbb{Z}_N^*, \ a^{\varphi(N)} = 1 \mod N \]
EULER THEOREM

\[ \forall a \in \mathbb{Z}_N^*, \ a^{\Phi(N)} \equiv 1 \mod N \]
EULER THEOREM

\[ \forall a \in \mathbb{Z}_N^*, a^{\Phi(N)} = 1 \mod N \]

\[ \prod_{x \in \mathbb{Z}_N^*} x = \prod_{x \in \mathbb{Z}_N^*} ax \]
EULER THEOREM

∀a ∈ ℤ_N^*, a^{\Phi(N)} = 1 \text{ mod } N

\prod_{x \in ℤ_N^*} x = \prod_{x \in ℤ_N^*} ax^{a^{\Phi(N)}}

\prod_{x \in ℤ_N^*} x
EULER THEOREM

$$\forall a \in \mathbb{Z}_N^*, \ a^{\Phi(N)} = 1 \mod N$$
TEXTBOOK RSA

\( \text{GEN}(1^n) \)

1. Pick 2 primes \( p, q \) \( \sim 1000 \) bit #s
2. \( N = p \cdot q \)
3. \( \phi(N) = (p-1)(q-1) \)
4. Pick \( e \) s.t. \( \gcd(e, \phi(N)) = 1 \)
5. Use Euclid to compute \( d \) s.t. \( e \cdot d = 1 \mod \phi(N) \)

\( e = 17, \phi(N) = 65537 \)

\( e \cdot d + k \cdot \phi(N) = 1 \)
TEXTBOOK RSA

\[ \text{GEN}(1^n) \]
\[ N = pq \quad \Phi(N) = (p - 1)(q - 1) \]
\[ e \text{ is a number such that } \gcd(e, \Phi(N)) = 1 \]
\[ d \text{ is such that } e \cdot d = 1 \mod \Phi(N) \]

\[ \text{ENC}_{pk}(m) : \]
\[ m^e \mod N \]

\[ \text{DEC}_{sk}(c) = c^d \mod N \]

\[ pK = (N, e) \]
\[ sK = (N, d) \]
\[ N = 949 \quad E = 11 \quad D = 707 \]
TEXTBOOK RSA

\textbf{GEN}(1^n)

\[ N \leftarrow pq, p, q \in \Pi_n, e \in \mathbb{Z}^{\ast}_{\phi(n)} \]
\[ pk \leftarrow (N, e) \quad sk \leftarrow (N, d) \]

\textbf{ENC}_{pk}(m)

\[ c \leftarrow m^e \mod N \]

\textbf{DEC}_{sk}(c)

\[ m \leftarrow c^d \mod N \]
CMDR EDWARD T LAYTON  
(FLEET INTELLIGENCE OFFICER)

LT CMDR JOSEPH ROCHEFORT  
(COMBAT INTELLIGENCE UNIT)
secure encryption schemes need to use randomness!
ENC\textsubscript{pk}(m)

**PICK** \( r \) **AS A RANDOM STRING WITH NO 0s** (TYPICALLY 8 BYTES)

\[ c \leftarrow (0||2|r||0|m)^e \mod N \]

CCA2 ATTACK AGAINST THIS SCHEME
PUBLIC FUNCTION. NOT KEYED.

ANYONE CAN EVALUATE, OUTPUT IS UNPREDICTABLE.
HEURISTIC SECURITY ONLY
CANNOT BE ALWAYS BE SECURELY INSTANTIATED
**OAEP+**

\[
\text{GEN}(1^n) \\
\begin{align*}
f, f^{-1} & \leftarrow \text{TRAPDOOR OWP()} \\
\end{align*}
\]

\[
\text{ENC}_{pk}(m) \\
\begin{align*}
r & \leftarrow U_n \\
& s \leftarrow R_1(r) \oplus m \ || \ R_2(r || m) \\
t & \leftarrow R_3(s) \oplus r \\
c & \leftarrow f(s || t)
\end{align*}
\]

\[
\text{DEC}_{sk}(c) \\
\begin{align*}
(s = (s_1, s_2), t) & \leftarrow f^{-1}(c) \\
r & \leftarrow R_3(s) \oplus t \\
m & \leftarrow R_1(r) \oplus s_1 \\
R_2(r || m) & \overset{?}{=} s_2 \quad \text{OUTPUT } m \text{ ELSE FAIL}
\end{align*}
\]
Theme

“SMALL PROBLEMS ARE EASY TO SOLVE.”

“SOLVE BIG PROBLEMS BY MAKING THEM INTO SMALLER ONES.”
first goal: create an amazing learning experience
second goal: instill my enthusiasm for this area
third goal: enjoy every second of this semester