CS306: Introduction to IT Security

Assignment Project Exam Help

Fall 2020

https://eduassistpro.github.io/

ght

Add WeChat edu_assist_pro

Instructor: **Nikos**

November 3, 2020

STEVENS INSTITUTE of TECHNOLOGY

THE INNOVATION UNIVERSITY®

Assignment Project Exam Help

https://eduassistpro.github.io/

ounce

Add WeChat edu_assist_pro

CS306: Announcements

- HW2 to come out this week
  - this time, for real!
  - covers MACs, hashing, PK-crypto, honeywords…
  - due in two weeks
- HW1 grades
  - talk to TAs
- Midterm grades
  - as soon as possible, given that they are graded by your instructor – end of week
- Labs resume this Thursday

CS306: Tentative Syllabus

Week	Date	Topics	Reading	Assignment
1	Sep 1	Introduction	Lecture 1	-
2	Sep 8	Symmetric-key encryption	Lecture 2	Lab 1
3	Sep 15		Lecture 3	Lab 2, HW 1
4	Sep 22		Lecture 4	Lab 3, HW 1
5	Sep 29	Ciphers in pra[illegible]	[illegible]ecture 5	Lab 4
6	Oct 6	MACs & hashing	Lecture 6	Lab 5
-	Oct 13	No class (Monday schedule)		Lab 6
7	Oct 20	Public-key cryptography	Lecture 7	

CS306: Tentative Syllabus (continued)

Week	Date	Topics	Reading	Assignment
8	**Oct 27**	**Midterm**	**All materials covered**	
9	Nov 3		Lecture 8	Lab 7, HW 2
10	Nov 10	[illegible]		
11	Nov 17	Web & Network		
12	Nov 24	Cloud Securit		
13	Dec 1	Software/Database Security		
14	Dec 8	Economics, Legal & Ethical Issues		
15	Dec 15 (or later)	**Final** (closed “books”)	All materials covered*	

* w/ focus on what covered after midterm

Two weeks ago

- Public-key (PK) cryptography
  - Motivation, PK Infrastructure, PK encryption, digital signatures
  - Discrete log problem & ElGamal encryption, hybrid encryption
- Demo
  - The length-extension attack against naïve

Today

- Crypto Light
  - Special topics on message authentication, cryptographic hashing, RSA
  - Final remarks on crypto tools w/ a focus on practical applications
  - Topics
    - authenticated en
    - cryptographic hashing in practice & Ho
    - RSA problem & RSA crypto system

Assignment Project Exam Help

https://eduassistpro.github.io/

henticated

n

Add WeChat edu_assist_pro

Recall: Two distinct properties

Secrecy

- **sensitive** information has value
  - if **leaked**, it can be **risky**
- **specific** scope / **gener**
- **prevention**
- does **not** imply integrity
  - e.g., bit-flipping “attack”

Integrity

- **correct** information has value
  - if **manipulated**, it can **harmful**
  - . **adversarial** manipulation
- / **context-specific** semantics
  - ntent authentication
- **detection**
- does **not** imply secrecy
  - e.g., user knows cookies’ “contents”

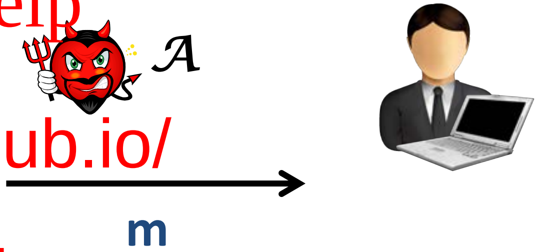
Recall: Yet, they are quite close…

Common setting

- communication (storage) over an "**open**," i.e., **unprotected**, channel (medium)

Fundamental security problems

- while in transit (at rest
  - no message (file) sh
  - no message (file) sh



Core cryptographic protections

- **encryption schemes** provide **secrecy / confidentiality**
- **MAC schemes** provide **integrity / unforgeability**

Can we achieve both at once in the symmetric-key setting? **Yes!**

Authenticated Encryption (AE): Catch 2 birds w/ 1 stone

Cryptographic primitive that realizes an "**ideally secure**" communication channel

- motivation
  - important in practice as real apps often **need both**
  - **good security hygie**
    - even if a given a integrity than the other, it's always better **to achieve both**!

Three generic AE constructions

Constructions of a **secure authenticated encryption** scheme Π_{AE}

- they all make use of
  - a **CPA-secure** encryption scheme Π_E = (Enc, Dec); and
  - a **secure MAC** Π_M = (Mac, Vrf)
  - which are instantia ke, km
  - …but the **order** with which these are use

Generic AE constructions (1)

1. **encrypt-and-authenticate**

- $\mathrm{Enc}_{ke}(m) \rightarrow c$; $\mathrm{Mac}_{km}(m) \rightarrow t$; send ciphertext (c, t)
- if $\mathrm{Dec}_{ke}(c) = m \neq$ `fail` and $\mathrm{Vrf}_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, genera**
  - e.g., MAC tag t may leak
  - e.g., if MAC is deterministic (e.g., CBC-MAC) the PA-secure
  - used in SSH

Generic AE constructions (2)

2. **authenticate-then-encrypt**

- $\text{Mac}_{km}(m) \longrightarrow t$; $\text{Enc}_{ke}(m||t) \longrightarrow c$; send ciphertext c
- if $\text{Dec}_{ke}(c) = m||t \neq$ `fail` and $\text{Vrf}_{km}(m,t)$ accepts, output m; else output `fail`
- **insecure scheme, genera**
  - used in TLS, IPsec

Generic AE constructions (3)

3. **encrypt-then-authenticate** (cf. "authenticated encryption")

- $\mathrm{Enc}_{ke}(m) \longrightarrow c$; $\mathrm{Mac}_{km}(c) \longrightarrow t$; send ciphertext (c, t)
- if $\mathrm{Vrf}_{km}(c,t)$ accepts then output $\mathrm{Dec}_{ke}(c) = m$, else output `fail`
- **secure scheme, generall**
  - used in TLS, SSHv2, IPsec

Application: Secure communication sessions

An AE scheme Π_{AE} = (Enc, Dec) enables two parties to **communicate securely**

- session: period of time during which sender and receiver maintain state
- idea: send any message m as c = $Enc_k(m)$ & ignore received c that don't verify
- security: **secrecy & int**
- remaining possible attac
  - **re-ordering** attack — counters can be use dering/replays
  - **reflection** attack — directional bit can be used to eliminate reflections
  - **replay** attack — c = $Enc_k(b_{A\rightarrow B} \| ctr_{A,B} \| m)$; $ctr_{A,B}$++

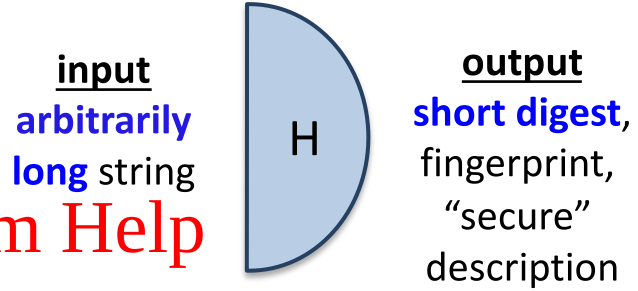
Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

lications of phic hashing

Recall: Cryptographic hash functions



Basic cryptographic primitive

- maps **objects** to a **fixed-length** binary strings
- for all practical purposes, mapping **avoids collisions** (distinct objects x ≠ y map
  - **collision resistance**: no nd that collide in the hash domain

“any object can be securely mapped t ique short digest”

Collision resistance implies two weaker security properties

- finding a collusion w.r.t. a given **random object x** is also infeasible
- finding any preimage of a **random hash value h** is also infeasible

Recall: Weaker security notions

Given a hash function H: X → Y, then we say that H is

- **preimage resistant** (or **one-way**)
  - given a **uniform** y ∈ Y, finding a value x ∈ X s.t. H(x) = y happens negligibly often
- **2-nd preimage resistan**
  - given a **uniform** x ∈ and H(x') = H(x) happens negligibly often

- cf. **collision resistant** (or **strong collision resistant**)
  - finding two distinct values x', x ∈ X, s.t. H(x') = H(x) happens negligibly often

Recall: Davies-Meyer & Merkle-Damgård transforms

$h(x||k) = F_k(x)$ XOR x

- h is a CR compression function,
  if F is an **ideal cipher (a more secure PRF)**

- H is a CR hash function, h is CR

Recall: Current hash standards & SHA2-512

Recall: Birthday attacks against collision resistance

Assume a CR function h producing hash values of size n

- **brute-force** attack
  - evaluate h on **$2^n + 1$** distinct inputs
  - by the "pigeon hole" … e found
- **birthday** attack
  - evaluate h on (much) **fewer** distinct input … **ndom** values
  - by "balls-into-bins" **probabilistic analysis**, at least 1 collision will **likely** be found
  - when hashing **only half** distinct inputs, it's **more likely** to find a collision!
  - thus, in order to get **n-bit security**, we (**at least**) need **hash values of length 2n**

Recall: Security strength due to birthday attacks

- # hash evaluations for finding collisions on n-bit digests with probability p

n m

- for large $m = 2^n$, average # hash evaluations before finding the first collision is

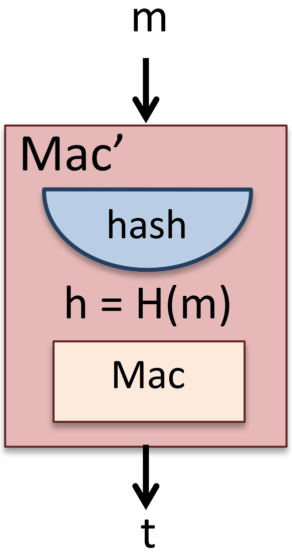
$1.25(m)^{1/2}$

plications of
raphic
phy

[1] “Hash-and-MAC” & “hash-and-sign”

Hash-and-MAC construction based on

- a CR hash function H; and
- a secure fixed-message MAC



Similarly, digital signature

- the hash of a message is signed, instead of the message itself

Intuition

- since **H is CR**:
  authenticating **digest H(m)** is **a good as** authenticating **m itself**!

[2] Hash-based MAC (HMAC): A naïve, insecure, approach

Set tag t as:

$$\mathrm{Mac}_k(m) = \mathbf{H}(k\,||\,\mathbf{m})$$

- intuition: given $H(k||m)$ it should be infeasible to compute $H(k||m')$, $m' \neq m$

Insecure construction

- practical CR hash functions
  employ the Merkle-Damgård design

- **length-extension attack**
  - knowledge of $H(m_1)$ makes it feasible to compute $H(m_1||m_2)$
  - by knowing the length of m_1, one can learn internal state z_B even without knowing m_1!

[2] HMAC: Secure design

Set tag t as:

$$\mathrm{HMAC}_k[m] \;=\; \mathbf{H}\,[\;(k \oplus opad)\;||\;\mathbf{H}[\;(k \oplus ipad)\;||\;\mathbf{m}\;]\;]$$

- intuition: **instantiation of hash & sign paradigm**
- two layers of hashing
  - **upper layer**
    - y = H((k ⊕ ipad) || m)
    - y = H'(m), i.e., "hash"
  - **lower layer**
    - t = H ((k ⊕ opad) || y')
    - t = Mac'(k_{out}, y'), i.e., "sign"

plications of
raphic

[1] Digital envelops

Commitment schemes

- two operations
- commit(x, r) = C

  - i.e., put message x s r)
  - e.g., commit(x, r) =

  - **hiding property**: you cannot see throug nvelop

- open(C, m, r) = ACCEPT or REJECT
  - i.e., open envelop (using r) to check that it has not been tampered with
  - e.g., open(C, x, r): check if h(x || r) =? C
  - **binding property**: you cannot change the contents of a sealed envelop

[1] Security properties

Hiding: perfect opaqueness

- similar to indistinguishability; commitment reveals nothing about message
  - adversary selects two messages x_1, x_2 which he gives to challenger
  - challenger randoml ness and) commitment C_i of x_i
  - challenger gives C_b ind bit b (better than guessing)

Binding: perfect sealing

- similar to unforgeability; cannot find a commitment “collision”
  - adversary selects two distinct messages x_1, x_2 and two corresponding values r_1, r_2
  - adversary wins if $\text{commit}(x_1, r_1) = \text{commit}(x_2, r_2)$

[1] Example 1: Online auctions

Suppose Alice, Bob, Charlie are bidders in an online auction

- Alice plans to bid A, Bob B and Charlie C
  - they do not trust that bids will be secret
  - nobody is willing to
- solution
  - Alice, Bob, Charlie submit **hashes** h(A), h   bids
  - all received hashes are posted online
  - then parties' bids A, B and C revealed
- analysis
  - "hiding:" hashes do not reveal bids   (which property?)
  - "binding:" cannot change bid after hash sent   (which property?)

[1] Example 1: Online auctions (II)

A general issue with concealing private data via hashing

- due to the small search space, this protocol is not secure!
- a forward search attack is possible
  - e.g., Bob computes
- how to prevent this?
  - increase search space
  - e.g., Alice computes h(A||R), where R is randomly chosen
    - at the end, Alice must reveal A and R
    - but before he chooses B, Bob cannot try all A and R combination

[1] Example 2: Fair decision via coin flipping

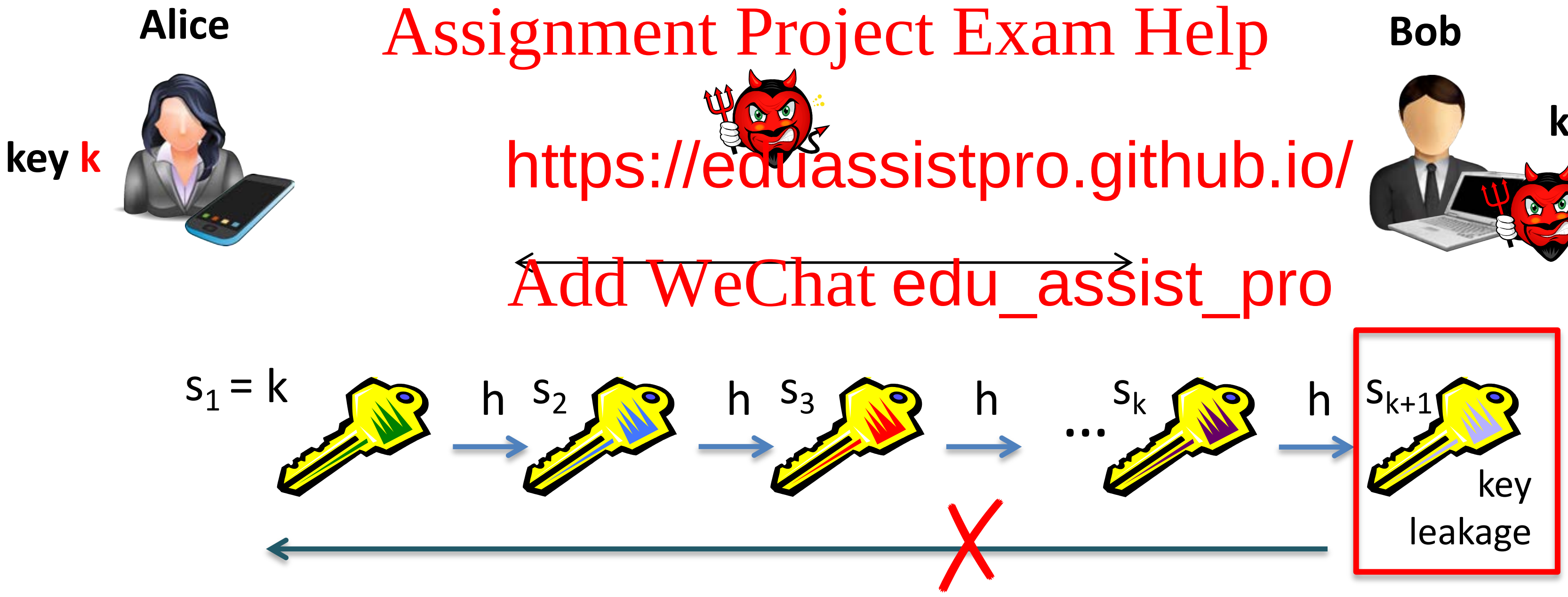
Alice is to “call” the coin flip and Bob is to flip the coin

- to decide who will do the dishes…
- problem: Alice may change her mind, Bob may skew the result
- protocol
  - Alice "calls" the coin f her call
  - Bob flips the coin and
  - Alice reveals what she committed to
  - Bob verifies that Alice's call matches her com
  - If Alice's revelation matches the coin result Bob reported, Alice wins
- hiding: Bob does not get any advantage by seeing Alice commitment
- binding: Alice cannot change her mind after the coin is flipped

[2] Forward-secure key rotation

Alice and Bob secretly communicate using symmetric encryption

- Eve intercepts their messages and later breaks into Bob's machine to steal the shared key



[3] Hash values as file identifiers

Consider a cryptographic hash function H applied on a file F

- the hash (or digest) H(M) of F serves as a **unique** identifier for F
  - “uniqueness”
    - if another file F’ adicts the security of H
  - thus
    - the hash H(F) of F is like a fingerprint
    - one can check whether two files are equal by comparing their digests

Many real-life applications employ this simple idea!

Examples

3.1 Virus fingerprinting

- When you perform a virus scan over your computer, the virus scanner application tries to identify and block or quarantine programs or files that contain viruses
- This search is primarily based [on] the digest of your files agains[t] the digests of already known viruses
- The same technique is used for confirming that is safe to download an application or open an email attachment

3.2 Peer-to-peer file sharing

- In distributed file-sharing applications (e.g., systems allowing users to contribute contents that are shared amongst each other), both shared files and ... nodes (e.g., their IP addresses) are into identifiers in a hash range
- Wh... dded in the system it is con... peer nodes that are resp... es those digests fall in a certain sub-range
- When a user looks up a file, routing tables (storing values in the hash range) are used to eventually locate one of the machines storing the searched file

Example 3.3: Data deduplication

Goal: Elimination of duplicate data

- Consider a cloud provider, e.g., Gmail or Dropbox, storing data from numerous users.
- A vast majority of stored d e.g., think of how many us email attachments, or a popular video…
- Huge cost savings result from deduplication:
  - a provider stores identical contents possessed by different users once!
  - this is completely transparent to end users!

Idea: Check redundancy via hashing

- Files can be reliably checked whether they are duplicates by comparing their digests.
- s ready to upload a new file to the 's digest is first uploaded.
- cks to find a possible duplicate, ointer to this file is added.
- Otherwise, the file is being uploaded literally
- This approach saves both storage and bandwidth!

[4] Password hashing

Goal: User authentication

- Today, passwords are the dominant means for user authentication, i.e., the process of verifying the identity of a u access to some computing
- This is a "something you know" type of user authentication, assuming that only the legitimate user knows the correct password.
- When you provide your password to a computer system (e.g., to a server through a web interface), the system checks if your submitted password matches the password that was initially stored in the system at setup.

Problem: How to protect password files

- If passwords are stored at the server in the clear, an attacker can steal the password file after the authentication server – this type pens routinely nowadays…
- involved having the server s of the users passwords.
- Thus, even if a password file leaks to an attacker, the onewayness of the used hash function can guarantee some protections against user-impersonation simply by providing the stolen password for a victim user.

[4] Password storage

Assignment Project Exam Help

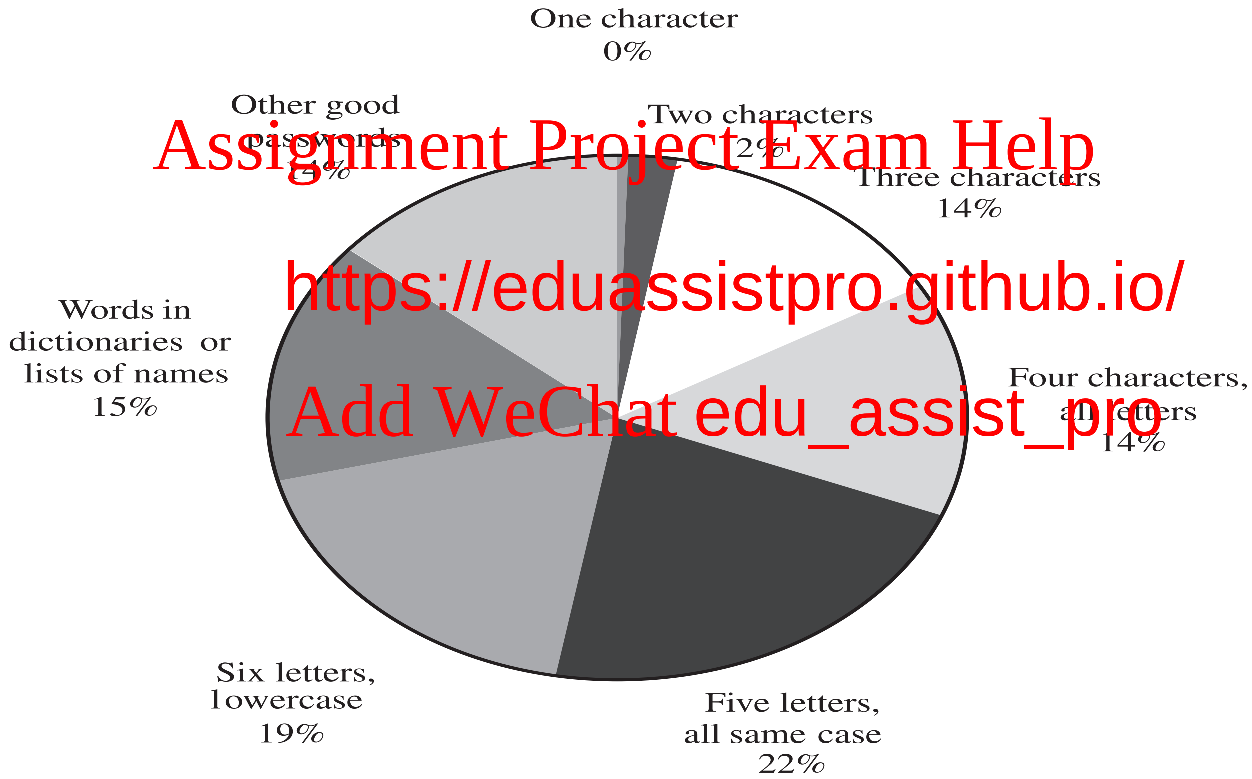
https://eduassistpro.github.io/

Add WeChat edu_assist_pro

Plaintext

Concealed via hashing

[4] Distribution of password types



[4] Dictionary attacks

- "online" brute-force or dictionary attack against passwords
  - employs only the authentication system
  - the attacker tries to impersonate a victim by trying
    - all possible (sh
    - passwords co
- “offline” brute-force or dictionary attac
  - employs a leaked file of hashed pass

[4] Countermeasures

Password salting

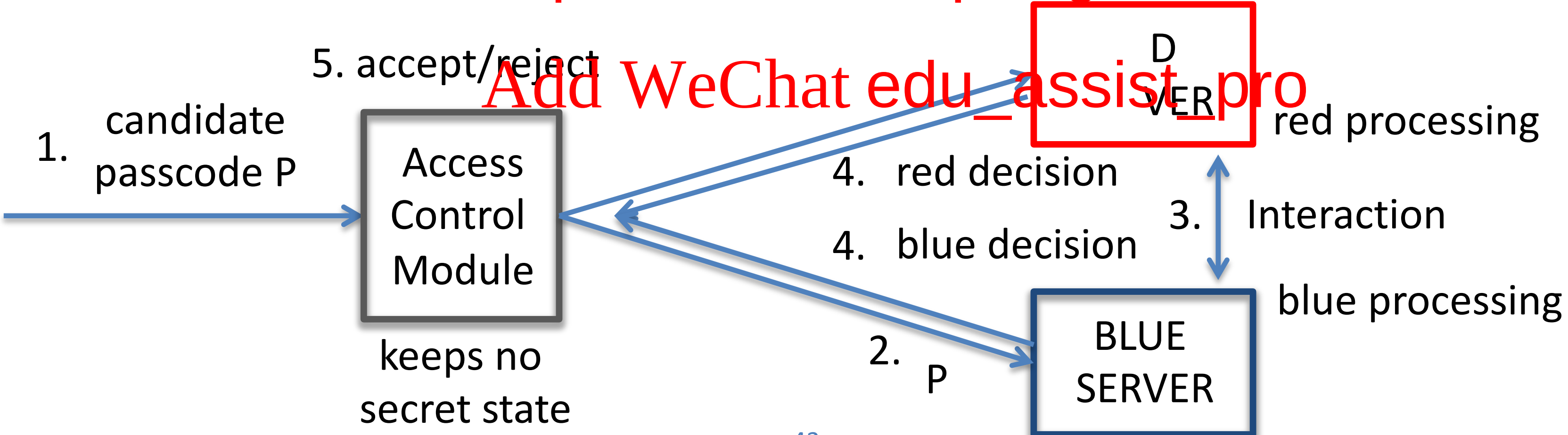
- to slow down dictionary attacks
  - a user-specific **salt** is appended to a user's password before it is being hashed
  - each salt value is store ding hashed password
  - if two users have the s t hashed passwords
  - example: Unix uses a 12 bit salt

Hash strengthening

- to slow down dictionary attacks
  - a password is hashed k times before being stored

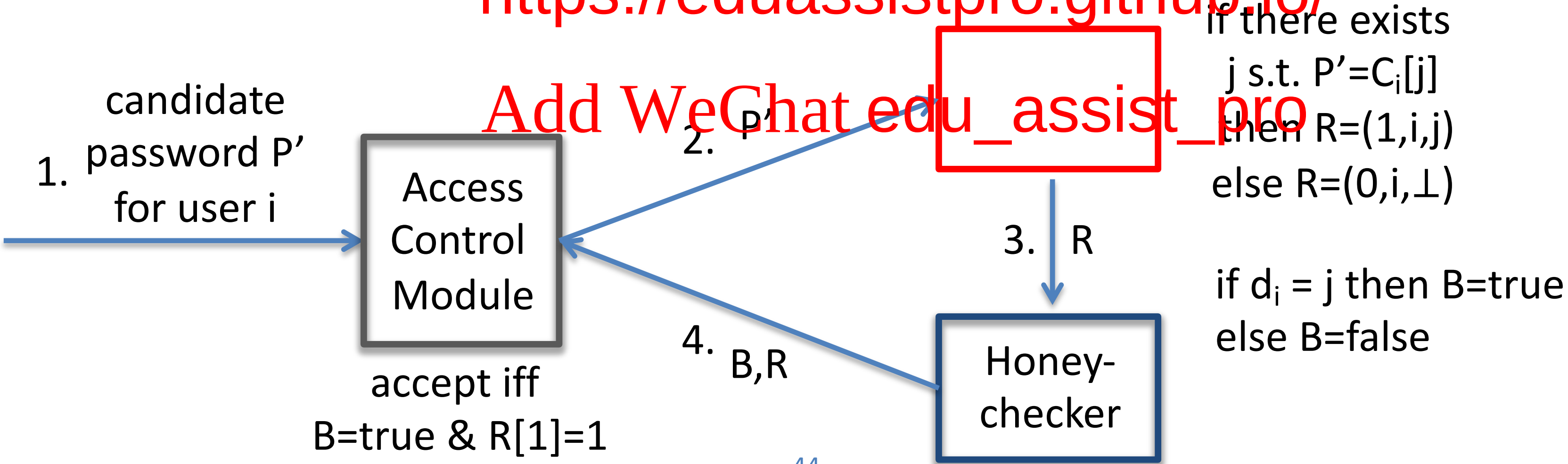
[4] A promising approach: Split verification into two servers

- Key idea: Distribute password verification across two servers
  - Red server verifies "half" the credentials; blue server verifies other "half"
  - Authentication decision relies on both outputs
    - Compromise of o to attacker



[4] Honeywords

- Based on decoy passwords, aka honeywords
  - Red stores user's i real password P_i and k-1 fake ones in unlabeled set C_i
  - Blue server stores the index d_i of P_i in set C_i
  - Password verification



[4] Example

- User nikos: hel00w0rld, **heloow0rld,** hel00w1rld, hel11w0rld
- Red server: nikos, hel00w0rld, heloow0rld, hel00w1rld, hel11w0rld
- Blue server: nikos, 2

- User provides hel11w0rld
- Red: if password is contained in list at output (nikos, 4)
  - Else reject
- Blue: if match, then output OK, else ALERT

Assignment Project Exam Help

https://eduassistpro.github.io/

ber theory

nd

Add WeChat edu_assist_pro

Multiplicative inverses

The residues modulo a positive integer n comprise set $Z_n = \{0,1,2,\ldots,n - 1\}$

- let x and y be two elements in Z_n such that $x\ y \bmod n = 1$
  - we say: y is the multiplicative inverse of x in Z_n
  - we write: $y = x^{-1}$

- example:
  - multiplicative inverses of the residues **m**

x	0	1	2	3	4	5	6	7	8	9	10
x^{-1}		1	6	4	3	9	2	8	7	5	10

Multiplicative inverses (cont'ed)

Theorem

An element x in Z_n has a multiplicative inverse iff x, n are relatively prime

- e.g., the only elements of Z_{10} having a multiplicative inverse are 1, 3, 7, 9

x	0							7	8	9
x^{-1}								3		9

Corollary

If p is prime, every non-zero residue in Z_p h e inverse

Theorem

A variation of Euclid's GCD algorithm computes the multiplicative inverse of an element x in Z_n or determines that it does not exist

Computing multiplicative inverses

Fact

- given two numbers **a** and **b**, there exist integers x, y s.t.

$$\mathbf{x\ a + y\ b = gcd(a, b)}$$

which can be computed e … n algorithm.

Thus

- the multiplicative inverse of a in Z_b exists iff
- i.e., iff the extended Euclidean algorithm computes x and y s.t. $\mathbf{x\ a + y\ b = 1}$
- in this case, the multiplicative inverse of a in Z_b is $\mathbf{x}$

Euclid's GCD algorithm

Computes the greater common divisor by repeatedly applying the formula

gcd(a, b) = gcd(b, a mod b)

Algorithm EuclidGCD(a, b)
Input integers **a** and **b**
Output gcd(a, b)

n a

uclidGCD(**b**, **a** mod **b**)

- example
  - gcd(412, 260) = 4

a	412	260	152	108	44	20	4
b	260	152	108	44	20	4	0

Extended Euclidean algorithm

Theorem

If, given positive integers **a** and **b**,
d is the smallest positive integer
s.t. **d** = **ia** + **jb**, for some integers
i and **j**, then **d** = gcd(**a**,

- example
  - **a** = 21, **b** = 15
  - **d** = 3, **i** = 3, **j** = -4
  - 3 = 3·21 + (-4)·15 = 63 - 60 = 3

Algorithm **Extended-Euclid**(**a**, **b**)
Input integers **a** and **b**
a, **b**), i and j
+ib = gcd(a, b)
ended-Euclid(**b**, **a** mod **b**)
(d, x, y) = (d′, y′, x′ - [a/b]y′)
return (d, x, y)

Multiplicative group

A set of elements where multiplication • is defined

- closure, associativity, identity & inverses
- multiplicative groups Z^*_n, defined w.r.t. Z_n (residues modulo n)

  - subsets of Z_n containi n

  - **CASE 1**: if n is a prime Z_n have an inverse
    - Z^*_7 = {1,2,3,4,5,6}, n = 7

    - 2 • 4 = 1 (mod 7), 3 • 5 = 1 (mod 7), 6 • 6 = 1 (mod 7)
  - **CASE 2**: if n is not prime, then not all integers in Z_n have an inverse
    - Z^*_{10} = {1,3,7,9}, n = 10
    - 3 • 7 = 1 (mod 10), 9 • 9 = 1 (mod 10), 1 • 1 = 1 (mod 10)

Order of a multiplicative group

Order of a group = cardinality of the group

- multiplicative groups for Z^*_n
- the totient function $\phi(n)$ denotes the order of Z^*_n, i.e., $\phi(n) = |Z^*_n|$
  - if **n = p is prime**, then , $\phi(n) = p-1$
    - e.g., $Z^*_7 = \{1,2,3,4,$
  - if **n is not prime**, $\phi(n) = n(1-1/p_1)(1-1/p_2)\ldots(1-$ $^{e1}{}_1 p^{e2}{}_2 \ldots p^{ek}{}_k$
    - e.g., $Z^*_{10} = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$

- if $n = p\,q$, where p and q are distinct primes, then $\phi(n) = (p-1)(q-1)$ **Factoring problem**
  - difficult problem: given $n = pq$, where p, q are primes, find p and q or $\phi(n)$

Fermat’s Little Theorem

Theorem

If **p is a prime**, then for each nonzero residue x in Z_p, we have $x^{p-1} \bmod p = 1$

- example (p = 5):

$1^4 \bmod 5 = 1$ 1

$3^4 \bmod 5 = 81 \bmod 5 = 1$ = 1

Corollary

If **p is a prime**, then the multiplicative inverse of each x in Z^*_p is $x^{p-2} \bmod p$

- proof: $x(x^{p-2} \bmod p) \bmod p = xx^{p-2} \bmod p = x^{p-1} \bmod p = 1$

Euler's Theorem

Theorem

For each element x in Z^*_n, we have $x^{\phi(n)}$ mod n = 1

- example (**n = 10**)
  - $Z^*_{10} = \{1,3,7,9\}$, n = 10
  - $3^{\phi(10)}$ mod 10 = 3^4 mod 10 = 81 mod 10 = 1
  - $7^{\phi(10)}$ mod 10 = 7^4 mod 10 = 2401 mod 10 = 1
  - $9^{\phi(10)}$ mod 10 = 9^4 mod 10 = 6561 mod 10 = 1

Computing in the exponent

For the multiplicative group Z^*_n, we can reduce the exponent modulo $\phi(n)$

- $x^y \bmod n = x^{k\,\phi(n) + r} \bmod n = (x^{\phi(n)})^k\, x^r \bmod n = x^r \bmod n = x^{\,y \bmod \phi(n)} \bmod n$

Corollary: For Z^*_p, we can

- example
  - $Z^*_{10} = \{1,3,7,9\}$, $n = 10$, $\phi(10) = 4$
  - $3^{1590} \bmod 10 = 3^{1590 \bmod 4} \bmod 10 = 3^2 \bmod 10 = 9$

- example
  - $Z^*_p = \{1,2,\ldots,p - 1\}$, $p = 19$, $\phi(19) = 18$
  - $15^{39} \bmod 19 = 15^{39 \bmod 18} \bmod 19 = 15^3 \bmod 19 = 12$

Powers

Let p be a prime

- the sequences of successive powers of the elements in Z^*_p exhibit repeating subsequences

- the sizes of the repe mber of their repetitions are the d

- example, p = 7

x	x^2	3	4	x^5	x^6
1	1	[illegible]	[illegible]	1	1
2	4	[illegible]	[illegible]	4	1
3	2	6	4	5	1
4	2	1	4	2	1
5	4	6	2	3	1
6	1	6	1	6	1

Assignment Project Exam Help

https://eduassistpro.github.io/

RSA a

Add WeChat edu_assist_pro

The RSA algorithm (for encryption)

General case

Setup (run by a given user)

- **n** = **p** · **q**, with **p** and **q** primes
- **e** relatively prime to $\phi(n) = (p - 1)(q - 1)$
- **d** inverse of **e** in $Z_{\phi(n)}$

Keys

- public key is $K_{PK} = (n, e)$
- private key is $K_{SK} = d$

Encryption

- $C = M^e \bmod n$ for plaintext **M** in Z_n

Decryption

- $M = C^d \bmod n$

Example

Setup

- **p** = 7, **q** = 17, **n** = 7 · 17 = 119
- **e** = 5, $\phi(n) = 6 \cdot 16 = 96$

- (119, 5)
- is 77

Encryption

- $C = 19^5 \bmod 119 = 66$ for $M = 19$ in Z_{119}

Decryption

- $M = 66^{77} \bmod 119 = 19$

Another complete example

- Setup
  - **p** = 5, **q** = 11, **n** = 5 · 11 = 55
  - **ϕ**(**n**) = 4 · 10 = 40
  - **e** = 3, **d** = 27 (3·27
- Encryption
  - **C** = **M**^{3} mod 55 for **M** in $\mathbf{Z}_{55}$
- Decryption
  - od 55

M	1	2	3	4	5	6	7	8	9	[illegible]	[illegible]	[illegible]	13	14	15	16	17	18
C	1	8	27	9	15	51	13	17	14	[illegible]	[illegible]	[illegible]	52	49	20	26	18	2
M	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	7	24	50	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

*Correctness of RSA

Given

Setup

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with $\mathbf{p}$ and $\mathbf{q}$ primes
- $\mathbf{e}$ relatively prime to $\phi(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- $\mathbf{d}$ inverse of $\mathbf{e}$ in $\mathbf{Z}_{\phi(n)}$

Encryption

- $\mathbf{C} = \mathbf{M}^e \bmod \mathbf{n}$ for plaintext $\mathbf{M}$ in $\mathbf{Z}_n$

Decryption

- $\mathbf{M} = \mathbf{C}^d \bmod \mathbf{n}$

Fermat's Little Theorem **(2)**

- for prime p, non-zero x: $x^{p-1} \bmod p = 1$

Analysis

Need to show

- $M^{ed} = M \bmod p \cdot q$

Use **(1)** and apply **(2)** for prime p

d - 1 M = $(M^{p-1})^{h(q-1)}$ M

- -1) M mod p = **M mod p**

rime q)

- **d q**

Thus, since p, q are co-primes

- $\mathbf{M^{ed} = M \bmod p \cdot q}$

A useful symmetry

[1] RSA setting

- modulo **n** = **p** · **q**, p & q are **primes**, public & private keys (e,d): **d · e = 1 mod (p-1)(q-1)**

[2] RSA operations involve **exponentiations**, thus they are **interchangeable**

- **C** = $\mathbf{M^e}$ mod **n** … t **M in $\mathbf{Z_n}$**)
- **M** = $\mathbf{C^d}$ mod **n** … xt **C in $\mathbf{Z_n}$**)

Indeed, their order of execution does not matter … **d) e mod n**

[3] RSA operations involve exponents that **"cancel** … are **complementary**

- $\mathbf{x^{(p-1)(q-1)}}$ mod **n** = **1** (Euler's Theorem)

Indeed, they invert each other: $\mathbf{(M^e)^d = (M^d)^e = M^{ed} = M^{k(p-1)(q-1)+1} \bmod n}$

$\mathbf{= (M^{(p-1)(q-1)})^k \cdot M = 1^k \cdot M = M \bmod n}$

Signing with RSA

RSA functions are complementary & interchangeable w.r.t. order of execution

- <u>core property</u>: **$M^{ed} = M \bmod p \cdot q$** for any message **M in Z_n**

RSA cryptosystem lend

- 'reverse' use of keys

- signing algorithm **Sign(M,d,n)**: **$\sigma = M^d$** age **M** in **Z_n**
- verifying algorithm **Vrfy(σ,M,e,n)**: return **M** == **σ^e** mod **n**

The RSA algorithm (for signing)

General case

Setup (run by a given user)

- $\mathbf{n} = \mathbf{p} \cdot \mathbf{q}$, with **p** and **q** primes
- **e** relatively prime to $\boldsymbol{\phi}(n) = (\mathbf{p} - 1)(\mathbf{q} - 1)$
- **d** inverse of **e** in $\mathbf{Z}_{\phi(n}$

Keys (same as in encryption)

- public key is $\mathbf{K}_{PK} = (\mathbf{n}, \mathbf{e})$
- private key is $\mathbf{K}_{SK} = \mathbf{d}$

Sign

- $\boldsymbol{\sigma} = \mathbf{M}^{\mathbf{d}} \bmod \mathbf{n}$ for message **M** in $\mathbf{Z}_{\mathbf{n}}$

Verify

- Check if $\mathbf{M} = \boldsymbol{\sigma}^{\mathbf{e}} \bmod \mathbf{n}$

Example

Setup

- $\mathbf{p} = 7$, $\mathbf{q} = 17$, $\mathbf{n} = 7 \cdot 17 = 119$
- $\mathbf{e} = 5$, $\boldsymbol{\phi}(\mathbf{n}) = 6 \cdot 16 = 96$

(119, 5)

- is 77

Signing

- $\boldsymbol{\sigma} = 66^{77} \bmod 119 = 19$ for $\mathbf{M} = 66$ in $\mathbf{Z}_{119}$

Verification

- Check if $\mathbf{M} = 19^{5} \bmod 119 = 66$

Digital signatures & hashing

Very often digital signatures are used with hash functions

- the hash of a message is signed, instead of the message itself

Signing message M

- let h be a cryptogra … setting (n, d, e)
- compute signature … od n
- send σ, M

Verifying signature σ

- use public key (e, n) to compute (candidate) hash value $H = \sigma^e \bmod n$
- if $H = h(M)$ output ACCEPT, else output REJECT

Security of RSA

Based on difficulty of **factoring** large numbers (into large primes), i.e., n = p · q into p, q

- note that for RSA to be secure, both p and q must be large primes
- widely believed to hold true
  - since 1978, subject … t any serious flaws found
  - best known algorit … urity parameter (key length |n|)
- how can you break RS

Current practice is using 2,048-bit long RSA ke … digits)

- estimated computing/memory resources needed to factor an RSA number within one year

Length (bits)	PCs	Memory
430	1	128MB
760	215,000	4GB
1,020	342×10^{6}	170GB
1,620	1.6×10^{15}	120TB

RSA challenges

Challenges for breaking the RSA cryptosystem of various key lengths (i.e., |n|)

- known in the form RSA-`key bit length’ expressed in bits or decimal digits
- provide empirical evidence/confidence on strength of specific RSA instantiations

Known attacks

- RSA-155 (**512-bit**) factored in ears (**1999**) and 292 machines
  - 160 175-400MHz SGI/Sun, 8 250MHz SGI/Origin, 12 II, 4 500MHz Digital/Compaq
- RSA-**640** factored in **5 mo**. using 30 2.2GHz CPU-years (**2**
- RSA-220 (**729-bit**) factored in **5 mo**. using 30 2.2GHz CPU-years (**2005**)
- RSA-232 (**768-bit**) factored in **2 years** using **parallel** computers 2K CPU-years (1-core 2.2GHz AMD Opteron) (**2009**)

Most interesting challenges

- prizes for factoring RSA-**1024**, RSA-**2048** is $100K, $200K – estimated at 800K, 20B Mips-centuries

Deriving an RSA key pair

- public key is pair of integers (e,n), secret key is (d, n) or d
- the value of n should be quite large, a product of two large primes, p and q
- often p, q are nearly 100 digits each, so n ~= 200 decimal digits (~512 bits)

  - but 2048-bit keys are becoming a standard requirement nowadays
- the larger the value o p and q

  - but also the slower t
- a relatively large integer e is chosen

  - e.g., by choosing e as a prime that is larger) and (q − 1)
  - why?
- d is chosen s.t. $e \cdot d = 1 \bmod (p-1)(q-1)$
  - how?

Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ(n)** = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is $\mathbf{C} = \mathbf{M}^3 \bmod 55$ for **M** in $\mathbf{Z}_{55}$

M	1	2	3	4	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	13	14	15	16	17	18
C	1	8	27	9	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	52	49	20	26	18	2
M	19	20	21	22	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	31	32	33	34	35	36
C	39	25	21	33	12	19	5	31	48	[illegible]	[illegible]	[illegible]	36	43	22	34	30	16
M	37	38	39	40	41	42	43	44	45	[illegible]	[illegible]	[illegible]	49	50	51	52	53	54
C	53	37	29	35	6	3	32	44	45	41	38	42	4	40	46	28	47	54

Discussion on RSA

- Assume **p** = 5, **q** = 11, **n** = 5 · 11 = 55, **ϕ**(**n**) = 40, **e** = 3, **d** = 27
  - why encrypting small messages, e.g., **M** = 2, 3, 4 is tricky?
  - recall that the ciphertext is **C** = **M**^{3} mod 55 for **M** in $\mathbf{Z}_{55}$

- Assume n = 2043439438435553434354542894348343435 6091 = p · q
  - can e be the number

- Are there problems
  - what other algorithms are required to be user?

- Are there problem with respect to RSA
  - does it satisfy CPA security?

Algorithmic issues

The implementation of the RSA cryptosystem requires various algorithms

- Main issues
  - representation of integers of arbitrarily large size; and
  - arithmetic operatio dular powers
- Required algorithms
  - generation of **random numbers** of a given o compute candidates **p**, **q**)
  - **primality testing** (to check that candidates **p**, **q** are prime)
  - computation of the **GCD** (to verify that **e** and **ϕ**(**n**) are relatively prime)
  - computation of the **multiplicative inverse** (to compute **d** from **e**)

Modular powers

Repeated squaring algorithm

- speeds up computation of $\mathbf{a}^{\mathbf{p}}$ mod $\mathbf{n}$
- write the exponent $\mathbf{p}$ in binary
  - $\mathbf{p} = \mathbf{p}_{\mathbf{b}-1} \mathbf{p}_{\mathbf{b}-2} \dots \mathbf{p}_1 \mathbf{p}$
- start with $\mathbf{Q}_1 = \mathbf{a}^{\mathbf{p}_{\mathbf{b}-1}}$ m
- repeatedly compute $\mathbf{Q}_\mathbf{i} = ((\mathbf{Q}_{\mathbf{i}-1})^2 \text{ mod } \mathbf{n})\mathbf{a}^{\mathbf{p}_{\mathbf{b}-\mathbf{i}}} \text{ mod } \mathbf{n}$
- obtain $\mathbf{Q}_\mathbf{b} = \mathbf{a}^\mathbf{p}$ mod $\mathbf{n}$

In total **O** (log **p**) arithmetic operations

Example

- 3^{18} mod 19 (18 = 10010)
- $_1$ 19 = 3
- d 19)3^0 mod 19 = 9
- 9)3^0 mod 19 = 81 mod 19 = 5
- $_4$ 9)3^1 mod 19 = (25 mod 19)3 mod 19 = 18 mod 19 = 18
- $\mathbf{Q}_5 = (18^2 \text{ mod } 19)3^0 \text{ mod } 19 = (324 \text{ mod } 19) \text{ mod } 19 = 17 \cdot 19 + 1 \text{ mod } 19 = 1$

Pseudo-primality testing

Testing whether a number is prime (**primality testing**) is a difficult problem

An integer $\mathbf{n} \geq 2$ is said to be a base-$\mathbf{x}$ **pseudo-prime** if

- $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$ (Fermat's little theorem)

- Composite base-$\mathbf{x}$ pse

  - a random 100-bit i o -prime
    with probability less than 10^{-13}

  - the smallest composite base-2 pseudo-p
- Base-$\mathbf{x}$ pseudo-primality testing for an integer $\mathbf{n}$
  - check whether $\mathbf{x}^{\mathbf{n}-1} \bmod \mathbf{n} = 1$
  - can be performed efficiently with the repeated squaring algorithm

Security properties

- Plain RSA is deterministic
  - why is this a problem?
- Plain RSA is also homomorphic
  - what does this mea
  - multiply ciphertext
  - $[(m_1)^e \bmod N][(m_2)^e \bmod N] = (m_1 m_2)^e$ m
  - however, not additively homomorphic

Real-world usage of RSA

- Randomized RSA
  - to encrypt message M under an RSA public key (e,n), generate a new random session AES key K, compute the ciphertext as [K^e mod n, $AES_K(M)$]
  - prevents an adve tions of the same M since K is chosen at ran es place
- Optimal Asymmetric Encryption Pa
  - roughly, to encrypt M, choose rand as $M' = [X = M \oplus H_1(r)\ ,\ Y = r \oplus H_2(X)\ ]$ where H_1 and H_2 are cryptographic hash functions, then encrypt it as $(M')^e$ mod n