RSA

By: Matthew Dixson and John Sadik

Test Questions

1. What does RSA stand for?
2. What assumption makes the RSA algorithm secure?
3. Why is hybrid encryption useful?

Matthew Dixson

Christian

Born and raised in Knoxville, TN

Got my B.S. in Computer Science with a minor in Cybersecurity last spring

Currently pursuing a Master's in CS with a Cybersecurity concentration

Interned at ORNL the last two summers

Favorite food is roast beef sandwiches

John Sadik

- ✝ Christian
- CS Undergrad at UTK
- Research with Dr. Scott Ruoti

Music Taste:

- Christian Music
- Rap
- Pop
- Musicals
- Metal
- Taylor Swift

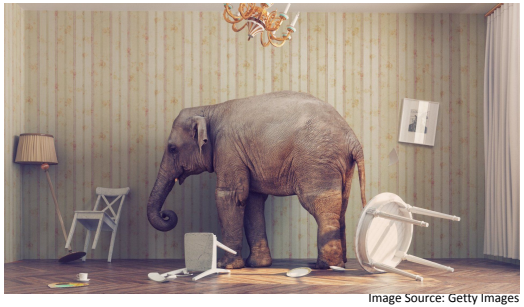


Image Source: Getty Images

Outline

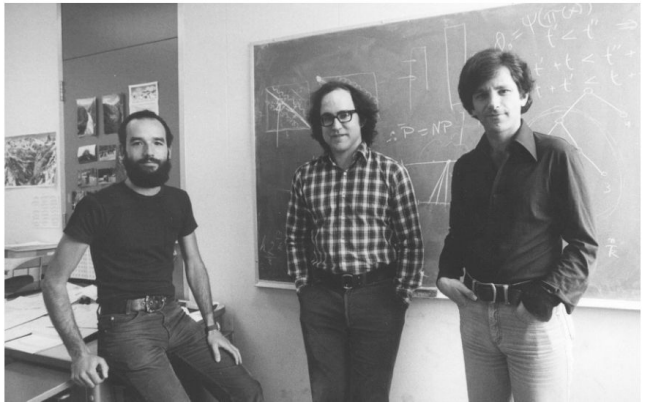
I. Overview
II. History
III. Algorithm
    A. Euclidean Algorithm
    B. Extended Euclidean Algorithm
    C. Chinese Remainder Theorem
    D. RSA
    E. Examples
IV. Applications
    A. Hybrid Encryption
    B. Message Signing
    C. Digital Certificates
V. Open Issues
    A. RSA Problem
    B. RSA in a Post-Quantum World
VI. References
VII. Questions

Overview

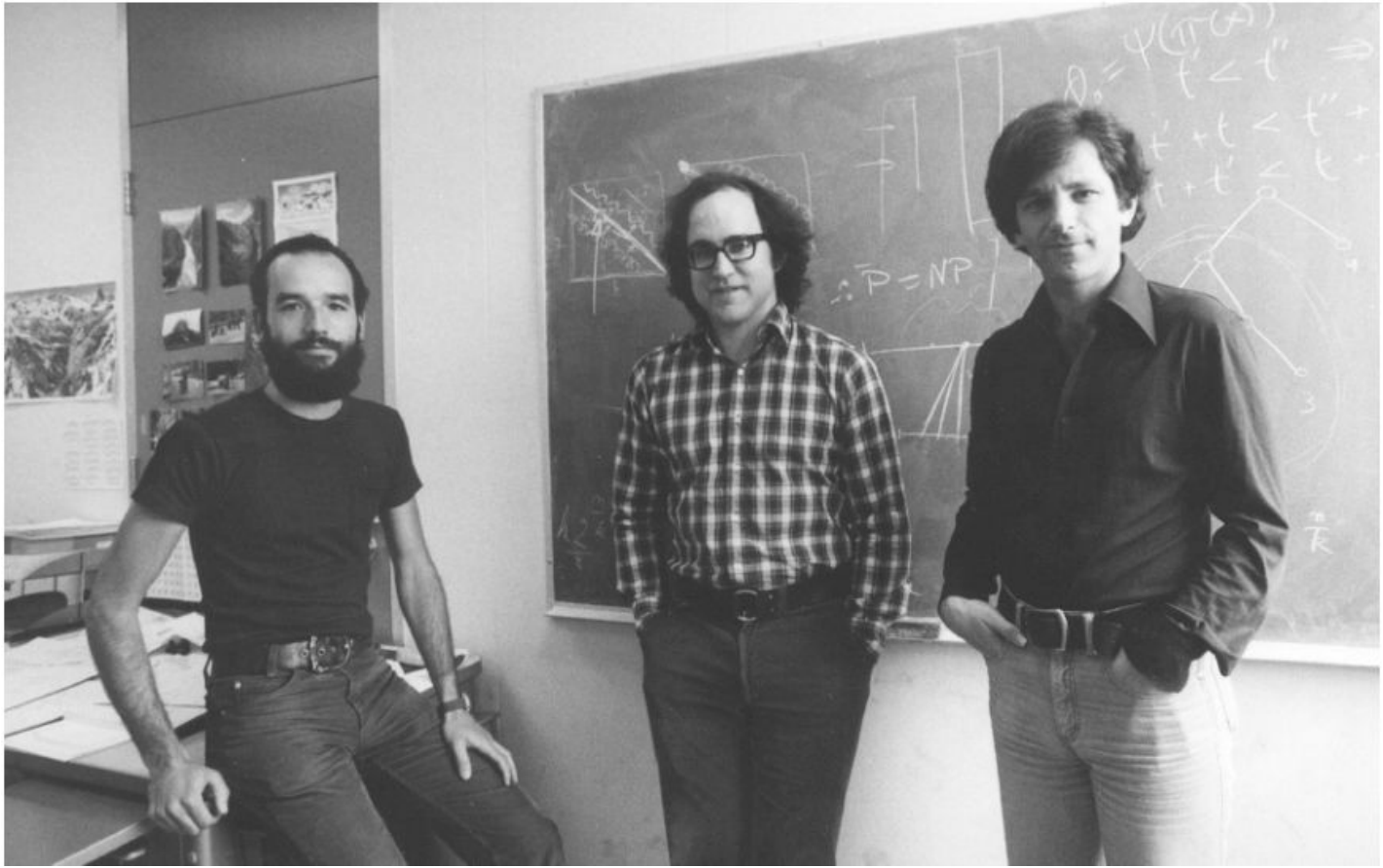
- Greatest Common Divisor (GCD)
  - d is a divisor of a,b when a,b,d $\in \mathbb{Z}$, d | a and d | b
  - d is the gcd of a,b when d is the largest possible d
- Euclidean Algorithm
  - Method for finding the greatest common divisor (GCD) of two numbers.
- Extended Euclidean Algorithm
  - Expresses the GCD as a linear combination of two numbers
- Hybrid encryption
  - Using RSA to agree on a symmetric key
- Signing messages
  - Ensure a message is untampered
- Digital Certificates
  - Verify a message sender’s identity

History

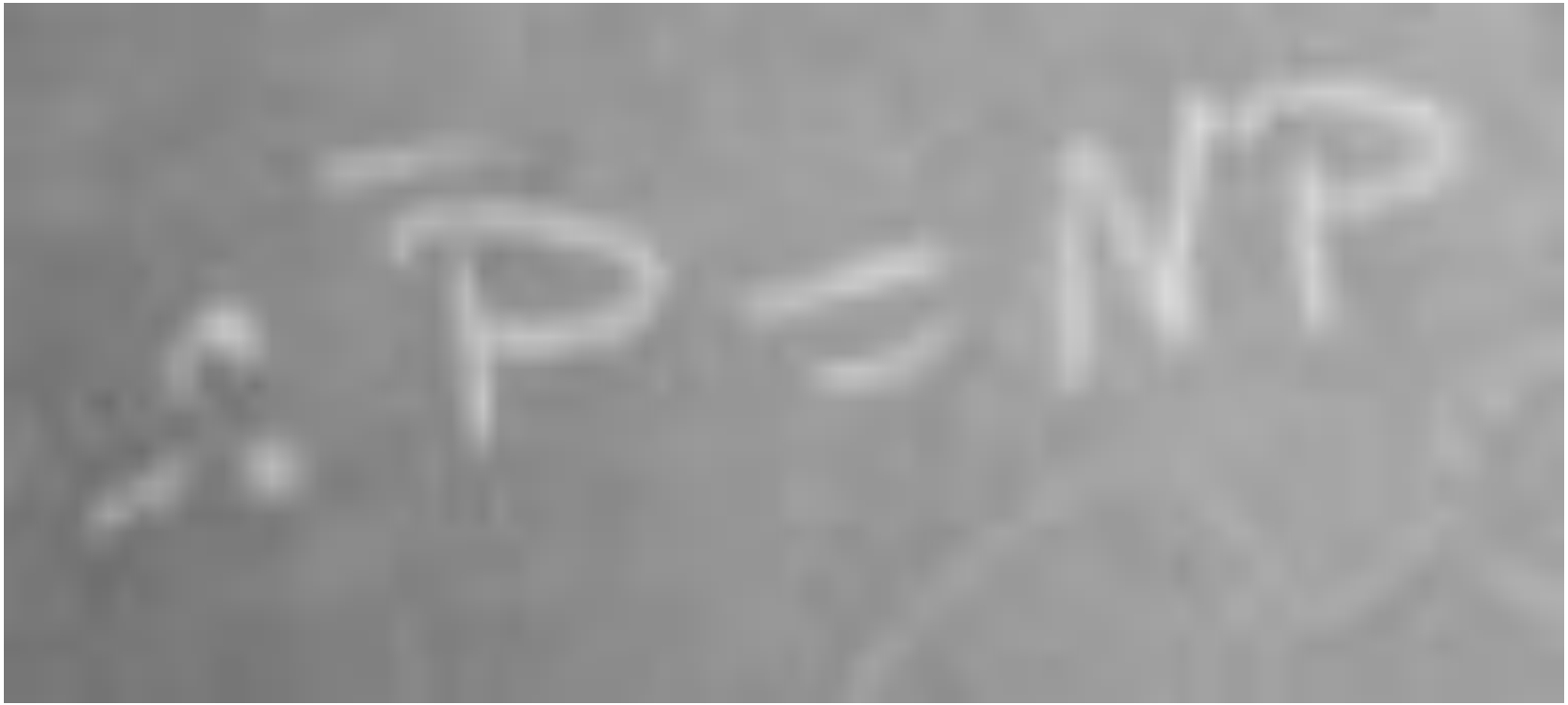
- Some of the first work on asymmetric public-private key cryptosystem was done by Whitfield Diffie and Martin Hellman in 1976
- The RSA system was developed by Ron Rivest, Adi Shamir, Leonard Adleman.
- MIT granted patent for RSA in 1983.
- Chinese Remainder Theorem- first stated by Sun-tzu in the 3rd century. Later generalized in 1247 and translated into English in the 19th century.



Source: people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf



Left to right: Shamir, Rivest, Adleman
Source: people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf



Source: people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf

Euclidean Algorithm

Two Rules:
- gcd(a,b) = gcd(b, a mod b)
- gcd(a,0) = a

Example:
- gcd(91,208)
= gcd(208, 91)
= gcd(91, 26)
= gcd(26, 13)
= gcd(13,0)
= 13

Example:
- gcd(816,22)
= gcd(22, 2)
= gcd(2, 0)
= 2

Example:
- gcd(70,266)
= gcd(266, 70)
= gcd(70, 56)
= gcd(56, 14)
= gcd(14,0)
= 14

Example:
- gcd(208,78)
= gcd(78, 52)
= gcd(52, 26)
= gcd(26, 0)
= 26

Example:
- gcd(511,61)
= gcd(61, 23)
= gcd(23, 15)
= gcd(15, 8)
= gcd(8,7)
= gcd(7,1)
= gcd(1,0)
= 1

Euclidean Algorithm Proof + Runtime

Proof:

gcd(a,b) | a and **gcd(a,b) | b**
gcd(a,b) | (b - q * a) from **b = q * a + r -> r = b - q * a**
gcd(a,b) | r

gcd(r,a) | r and **gcd(r,a) | a**
gcd(r,a) | (q * a + r) from **b = q * a + r**
gcd(r,a) | b

Therefore, **gcd(a,b) | a** and **gcd(a,b) | b** iff **gcd(r,a) | r** and **gcd(r,a) | a**

Runtime:
- O(log(min(a,b)))

Worst Case:
- Consecutive fibonacci numbers

Extended Euclidean Algorithm

Find the solution to
gcd(a,b) = s*a + t*b

Rules:
- $a = r_{-2}$ $b = r_{-1}$
- $r_{i-2} = r_{i-1} * q_i + r_i$
- $a = b * q_0 + r_0$
- $b = r_0 * q_1 + r_1$

Stop the algorithm when
$r_n | r_{n-1}$ with $gcd(a,b) = r_n$

To find s and t use the following
- **$gcd(a,b) = r_n = r_{n-2} - r_{n-1} * q_n$**

Simplify to
- **$gcd(a,b) = r_{i-1} * q_s + r_i * q_t$**

Replace r_i with
- **$r_i = r_{i-2} - r_{i-1} * q$**

Continue until you reach r_0
Now you have
- $gcd(a,b) = r_{-2} * q_s + r_{-1} * q_t$
- $a = r_{-2}$ $b = r_{-1}$
- $s = q_s$ $t = q_t$

Example:
a = 48, b = 5
48 = 5 * 9 + 3
5 = 3 * 1 + 2
3 = 2 * 1 + 1
gcd(a,b) = 1

1 = 3 - 2
1 = 3 - (5 - 3)
1 = 2 * 3 - 5
1 = 2 * (48 - 5 * 9) - 5
1 = 2 * 48 - 19 * 5
s = 2, t = -19

Example:
a = 40, b = 17
40 = 17*2 + 6
17 = 6*2 + 5
6 = 5*1 + 1
gcd(a,b) = 1

1 = 6 - 5*1
1 = 6 - (17 - 6*2)
1 = 17*-1 + 6*3
1 = -17 + (40 - 17*2) *3
1 = 40*3 - 17 * 7
s = 3, t = -7

Using the Extended Euclidean Algorithm

Start: p = 5, q = 13, e = 5

Step 1: n = 5* 13 = 65.

Step 2: $\phi(n) = 4 * 12 = 48$

Calculate d:

48 = 5 * 9 + 3

5 = 3 * 1 + 2

3 = 2 * 1 + 1

1 = 3 - 2

1 = 3 - (5 - 3)

1 = 2 * 3 - 5

1 = 2 * (48 - 5 * 9) - 5

1 = 2 * 48 - 19 * 5

Use mod to find d:

$-19 \equiv 29 \pmod{48}$

(mod m) implies

n = qm + r

-19 = -1*48 + 29

d = 29

RSA

- Asymmetric Encryption (public-key cryptographic algorithm)
  - You have a private key and everyone knows your public key
- Choose some $p, q \in \mathbb{Z}$
  - p and q need to be primes (not just coprime)
  - NIST recommends at least 2048 bits, but 4096 bits is preferable
- Calculate $n = p * q$
- Calculate $\phi(n) = (p-1)(q-1)$
- Select $e \in \mathbb{Z}^*_n$ with $\gcd(e, \phi(n)) = 1$
  - Almost always use e = 65537, but e = 3 is an easy number for toy examples
- Find $d \equiv e^{-1} \pmod{\phi(n)}$

RSA Continued

- Create the key pair
  - Public key: (e, n)
  - Private key: (d, n)
- Destroy p and q
  - Seriously
  - If an attacker gets p and q, they can calculate d on their own
- Encryption
  - Given plaintext m, $c \equiv m^e \pmod{n}$
- Decryption
  - Given ciphertext c, $m \equiv c^d \pmod{n}$

Why RSA Works

1. $c^d \ (mod\ n)$
   - $c \equiv m^e \ (mod\ n)$
2. $c^d \equiv (m^e)^d \ (mod\ n) \equiv m^{ed} \ \ (mod\ n)$
   - $d \equiv \mathrm{e}^{-1} \left(mod\ \varphi(n)\right)$
   - $1 \equiv d \cdot e \left(mod\ \varphi(n)\right)$
   - $k \cdot \varphi(n) + 1 = d \cdot e$
3. $c^d \equiv m^{k \cdot \varphi(n)+1} \equiv \left(m^{\varphi(n)}\right)^k \cdot m^1$
   - $\mathrm{a}^{\varphi(n)} \equiv 1 \ (mod\ n)$ (Euler's theorem)
   - $m^{\varphi(n)} \equiv 1 \ (mod\ n)$
4. $c^d \equiv \ 1^k \cdot m \equiv m$

Real RSA Example

p:
30535232441473242884522340247458740641767064986422534409366743629729192903225562098963871983039017055889068682679523723089254732301235498637867764892380404518085373840313218024709357156066154482089169841088920107267640303556875079619439338455037368640629538345241675978769539540096934591900466471810362098298285807185252260816219151072197990257709938953964700153678389609419998247907130934528107303279065619246462964506312455920828388957124267253025505865360097724860784451015551548819993418746159177984938423644306834221068548288443643448257810292393965093488099357847527435138263242059004940642050032691470595384711

Real RSA Example

q:
18251780939971287238364837370633353518306421374736470525315407652758072965283142279233357819301020712602069860891299542638309645806610298717604447292292337866187329993230800476462971779005674190947520141883176790020134977452398692785497197067510979421288258837101244448566920448448030890261131916829713537117802150258467514771714852722385633136135454279198375093366371318118999090236720401162626303524651736378464594088940305424020416123736283157679884701753034921278034252621733468897082567747368032092987795815312316427047412400069056705407667732175944930602829229356296976139265268184837346784386014397380349842093

Real RSA Example

n:
557322373472874249149149164496076216201070980422583979733266733260328700794038347380753998004499246932143726684125186912412053647249214038638644060612827179809011932579747818256431009139108573162540086487559489518209933952157861734177428473640754385952580263246009036195105661392579236442353941174527468969542077923606823674592434147086095095600533017479722262469174112634215378480967679538648844275910734400627919185197304233804873651780584434808794285825392064742693916171092509948055830596907608263150707944892463907456991834550314021538142681596946874011807136141007495022013344932613371692274917272544556729163777790036429088577345892280999090480015511371257201845733039404708380602879431911285863645544620909988288931382204734973702335928465686344792650542775426893940619356502776772553809271880811027882250844458606856931331998513703919882802254781124584505744965145651197709372691941915327272074163507381520291896562098952379210436771319485729274914175439967297723396478908448670033917910160272077358380437596937666784529281202209686246417307894923531955405328019418360420397250179714714278304801940552340361281677350787705418750286331051241230907406322170616771765879979257432773900598337185915875510090528426794523436440123

Real RSA Example

ϕ(n):
557322373472874249149149164496076216201070980422583979733266733260328700794038347380753998004499246932143726684125186912412053647249214038638644060612827179809011932579747818256431009139108573162540086487559489518209933952157861734177428473640754385952580263246009036195105661392579236442353941174527468969542077923606823674592434147086095095600533017479722262469174112634215378480967679538648844275910734400627919185197304233804873651780584434808794285825392064742693916171092509948055830596907608263150707944892463907456991834550314021538142681596946874011807136141007495022013344932613371692274917272544556729163729003023047644047223005103380998385855437884896042840798357253425893337010923206907666415742280872219797792838633911707974771550357840547437178330590754151556346652669232754052636942945739199209214154475634760034044223232694646110397318245602036157683047348468854788945355481926782306592001908992880216261146010994935490661183385481934691290781594574064560321231863687742494920572016420741667646830793220311159601722606956924901568502814062981544699937452305227774258431476077429260587725954058813151203751131328086268102170370362728530753740844146046861741789050670228949489320808675672033222664092379705672491213320

Real RSA Example

e = 65537

d:
27712622650920893588770424298637611934572075484094735388479326645234892810894794794458109597587044660305933694221
99916246041778576461752459092746486297940420772400454538477791375951558024515950107703486344597501322829138903660
58839943452387185939300386387622543920876322170732463514841344331144773112915738886998723995592869792914597635559
56139936478590153942184331669106800449751953514412935423558086850195534831668228954255414854544096046028291726792
78343207135791218600208059983549329906387540646974923367030556115448709652548075992696321924059368189683735412202
65852148370803834241176809385657796421528022562674330542296947932072125719674519431395978714463478527306650241331
99021944957065421771986867199001249773255978261173691067243666715417338767047443074045640624824020362111761229517
77977737265323256149941386705522919875708858615294128117800297082209920244529830018448058327685314317870818286955
48441495033042213073930852845839291799199934163720032714201199011488041080671587899238971445151153075811606801072
51573157253308720746314705727015837930763450537378216420835196465406968494760615749931814686159883686204498020537
52125503689218209652159879595744240867773996585376694298031029492731300303678633470778715816783670851 53

Real RSA Example

Message m: This is a test message.

Ciphertext c:
553501140981627037676179678758783008637639202975740281365601585142696286722526120679018671623659522153857504216941614180808945227912727934473911317761685172084658994211056646138597533592902430693230312315901834450186359358222955615153633683087315917864126006122187495840288269737643526654416186200439330863792979892365119390324147807341105780683912974427829860261135118122476382828751841507695067278984116491630539488434424340627798355733617360568708718954016102463282578481269191076340648703380150408022825012592109079320263885427858702066153135908723935668159751129710784432438157063242467165532071640085021338868022878036218522690911106712531253989526233312897561843388130124807318721361349974067016428501206511855271050474274905583986740523495435413403598887489938233784449951373981312880192912218357628445782056097533153770340972778647138129786862561191802297634240188304082220974879806668434842285591884206956010186849836995748746865414072082438655792497801508986507278536319303105034417376086215781879850146746820230490489938450849614261829954213755176777281475835091020231637411995093723314789585482214961055644450096118428667365263192319032727261452336835213100620406139908525985448842677204483604600699639063739875909566473

Why is e always 65537?

- Compromise on security and speed
  - 3 would be a nice option because it's fast, but it's not very secure
- It's a Fermat number
  - $F_n = 2^{(2\text{^}n)} + 1$
  - If $2^k + 1$ is prime and $k > 0$ then k must be a power of 2
    - Called Fermat primes.
      - As of 2023, the only known Fermat primes are F0 = **3**, F1 = **5**, F2 = **17**, F3 = **257**, and F4 = **65537**
      - Reduces $\gcd(p-1,e)=1$ to $p \not\equiv 1 \pmod e$
- 65537 in binary: 10000000000000001
  - Helps with FME

Chinese Remainder Theorem

- Let $n = n_1 n_2 \ldots n_k$ where the n_i are coprime to all other n_i.
- $a \leftrightarrow (a_1, a_2, \ldots, a_k)$ where $a \in \mathbb{Z}_n$, $a_i \in \mathbb{Z}_{ni}$ and $a_i = a \bmod n_i$
- There is a bijection between $\mathbb{Z}_n$ and the Cartesian product $\mathbb{Z}_{n1} \times \mathbb{Z}_{n2} \times \ldots \times \mathbb{Z}_{nk}$.
- If $a \leftrightarrow (a_1, a_2, \ldots, a_k)$ and $b \leftrightarrow (b_1, b_2, \ldots, b_k)$ then
  - $(ab) \bmod n \leftrightarrow ((a_1 b_1) \bmod n_1, \ldots, (a_k b_k) \bmod n_k)$
  - $(a+b) \bmod n \leftrightarrow ((a_1+b_1) \bmod n_1, \ldots, (a_k+b_k) \bmod n_k)$
  - $(a-b) \bmod n \leftrightarrow ((a_1-b_1) \bmod n_1, \ldots, (a_k-b_k) \bmod n_k)$

Chinese Remainder Theorem and RSA

- Used to speed up decryption
- You can split up the message M into two parts
  - $m_1 = (M \bmod p)^{d \bmod p-1}) \bmod p$
  - $m_2 = (M \bmod q)^{d \bmod q-1}) \bmod q$
- Then we recombine m_1 and m_2 as follows
  - $m \equiv (M^d \bmod N) \bmod p$
  - $m \equiv (M^d \bmod N) \bmod q$
- Because of CRT, we know
  - $m \equiv (M^d \bmod N) \bmod pq$

Types of RSA

- Blind RSA: Blind RSA is a technique used to prevent the private key holder from seeing the plaintext message being encrypted. This is useful in scenarios where the private key holder is untrusted, such as in electronic voting systems.
- Multi-Prime RSA: Multi-Prime RSA is a variant of RSA that uses multiple prime numbers in the key generation process. This can improve performance by reducing the number of modular multiplications required for encryption and decryption.
- Homomorphic RSA: Homomorphic RSA is a technique that allows for computations to be performed on encrypted data without decrypting it first. This is useful in scenarios where data privacy is a concern, such as in cloud computing.
- Elliptic Curve RSA: Elliptic Curve RSA is a variant of RSA that uses elliptic curve cryptography instead of the traditional modular arithmetic used in RSA. This can provide better security with shorter key sizes, making it useful in resource-constrained environments.
    - NSA backdoor???

Padding

- RSA with PKCS#1 v1.5 Padding: This is the most common implementation of RSA and is used in a wide range of applications. It uses PKCS#1 v1.5 padding to ensure that the message being encrypted is of the same length as the RSA modulus.
- RSA with OAEP Padding: RSA with Optimal Asymmetric Encryption Padding (OAEP) is an improved version of RSA with PKCS#1 v1.5 padding. OAEP provides better security and resistance against attacks like chosen ciphertext attacks.

Applications- Hybrid Encryption

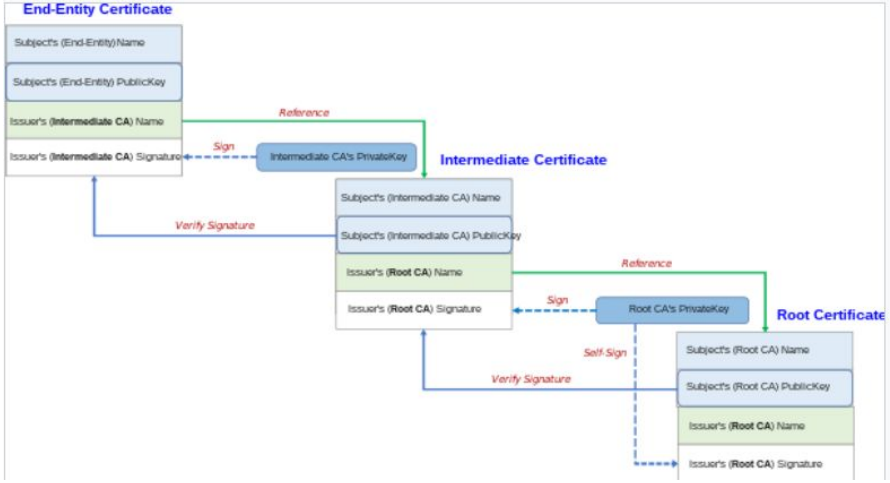
- Asymmetric key cryptosystems are typically less efficient than symmetric key systems due to their mathematical complexity
- Hybrid systems use both a asymmetric key cryptosystem (RSA) and a symmetric key cryptosystem
- Asymmetric keys are used for key encapsulation
- Symmetric keys are used for data encapsulation
- TLS (Transport Layer Security) uses a method similar to this

Applications- Message Signing

- The sender hashes the message they are wanting to send and encrypts the hash
  - Called signing
- Signed hash is sent along with the message
- The receiver can decrypt the encrypted hash using the sender’s public key
- The receiver can then hash the message and compare it with the decrypted hash
- If the hashes are different, then the message has been tampered with by someone other than the private key’s owner

Applications-Digital Certificates

- Digital certificates are a method to verify a subject's identity
- A subject has their certificate signed (encrypts) by a certificate authority (CA) using their private key
- Anyone who wants to verify the subject's identity will use the CA 's public key to verify (decrypt) the CA signature
    - If the certificate decrypts correctly, then the subject's authority is verified
    - The certificate authority's certificate is also verified by going to the certificate authority that signed it
    - Eventually, a trusted root CA is reached whose certificate is self signed
- Most common format is defined by X.509.



Source: *en.wikipedia.org/wiki/Chain_of_trust*

Open Issues

- RSA Problem
  - Find the eth roots of an arbitrary number mod n
  - Use this to obtain a value m so that $c \equiv m^e \bmod n$
  - Most efficient method is to factor n
    - Believed to be impractical with large n's.
  - This means that the RSA problem is at least as easy as the integer factorization problem
  - Correctly padding RSA messages makes this problem more difficult

RSA in a Post-Quantum World

- Shor's algorithm finds prime factors of an integer (bad for RSA)
- Quantum computers rely on qubits (quantum bits)
  - November 9, 2022: IBM revealed the largest quantum computer at the time, Osprey, with 433 qubits
- With the current recommended key size of 2048 bits it seems like we're still safe
  - Some estimates show it would take 20 million-qubit quantum systems 8 hours to crack a single encryption key
  - We can also always just increase the key size
- But it would still be better to find cryptographic algorithms that aren't vulnerable to Shor's algorithm

References

https://people.csail.mit.edu/rivest/pubs/ARS03.rivest-slides.pdf

https://www.itu.int/rec/T-REC-X.509-201910-I/en

https://sites.millersville.edu/bikenaga/number-theory/extended-euclidean-algorithm/extended-euclidean-algorithm.html

https://www.newscientist.com/article/2346074-ibm-unveils-worlds-largest-quantum-computer-at-433-qubits/#:~:text=Dubbed%20Osprey%2C%20it%20has%20433,Google's%2053%2Dqubit%20computer%20Sycamore

https://arstechnica.com/information-technology/2023/01/fear-not-rsa-encryption-wont-fall-to-quantum-computing-anytime-soon/

Questions?

Test Questions

1. What does RSA stand for?
2. What assumption makes the RSA algorithm secure?
3. Why is hybrid encryption useful?