Cryptography Basics – Public Key Cryptography

ECEN 4133

FEB 4, 2021

Shared key limitations

Suppose Alice publishes data to lots of people, and they all want to verify integrity…

- Can’t share an integrity key with *everybody*, or else *anybody* could forge messages

Suppose Bob wants to receive data from lots of people, confidentially…

- Schemes we’ve discussed would require a separate key shared with each person

[What to do?]

Public-key crypto

So far, encryption key == decryption key “**symmetric key crypto**”

New idea: Keys are distinct, and *you can’t find one from the other*

Almost always used by splitting key-pair

- Alice keeps one key private (“**private key**”)
- Publishes the other key (“**public key**”)

Invented in 1976 by Diffie and Hellman (earlier by Clifford Cocks of British intelligence, in secret)

- First popular public key algorithm: **RSA**
  - Rivest, Shamir, and Adleman 1978

Requirements for a public key crypto system to be secure

1. Computationally easy for B to generate a key pair: PU_b, PR_b
2. Computationally easy for sender A to generate the ciphertext for message M: $C=E(PU_b, M)$
3. Computationally easy for receiver B to decrypt the ciphertext: $M=D(PR_b, C)$
4. Computational infeasible to guess PR_b knowing PU_b.
5. Computational infeasible to recover M from PU_b and C.

RSA

A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

How RSA works

Key generation:

1. Pick large (say, 1024 bits) random primes **p** and **q**
2. Compute **N** := **pq**
   (RSA uses multiplication mod **N**)
3. Pick **e** to be relatively prime to (**p**-1)(**q**-1)
4. Find **d** so that **ed** mod (**p**-1)(**q**-1) = 1
5. Finally: **Public key** is **(e,N)**
   Private key is **(d,N)**

To encrypt: $E(\mathbf{x}) = \mathbf{x}^{\mathbf{e}} \bmod \mathbf{N}$
To decrypt: $D(\mathbf{x}) = \mathbf{x}^{\mathbf{d}} \bmod \mathbf{N}$

Why RSA works

“It works” theorem:

For all $0 < \mathbf{x} < \mathbf{N}$,
can show that $D(E(\mathbf{x})) = \mathbf{x}$

Proof:

$D(E(\mathbf{x})) = (\mathbf{x}^{\mathbf{e}} \bmod \mathbf{pq})^{\mathbf{d}} \bmod \mathbf{pq}$

$= \mathbf{x}^{\mathbf{ed}} \bmod \mathbf{pq}$

$= \mathbf{x}^{\mathbf{a}(\mathbf{p}-1)(\mathbf{q}-1)+1} \bmod \mathbf{pq}$ for some $\mathbf{a}$ (because $\mathbf{ed} \bmod (\mathbf{p}-1)(\mathbf{q}-1) = 1$)

$= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq}$

$= (\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq})^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq}$

$= 1^{\mathbf{a}}\mathbf{x} \bmod \mathbf{pq}$

(because of the fact that if $\mathbf{p},\mathbf{q}$ are prime, then for all $0 < \mathbf{x} < \mathbf{N}$, $\mathbf{x}^{(\mathbf{p}-1)(\mathbf{q}-1)} \bmod \mathbf{pq} = 1$)

$= \mathbf{x}$

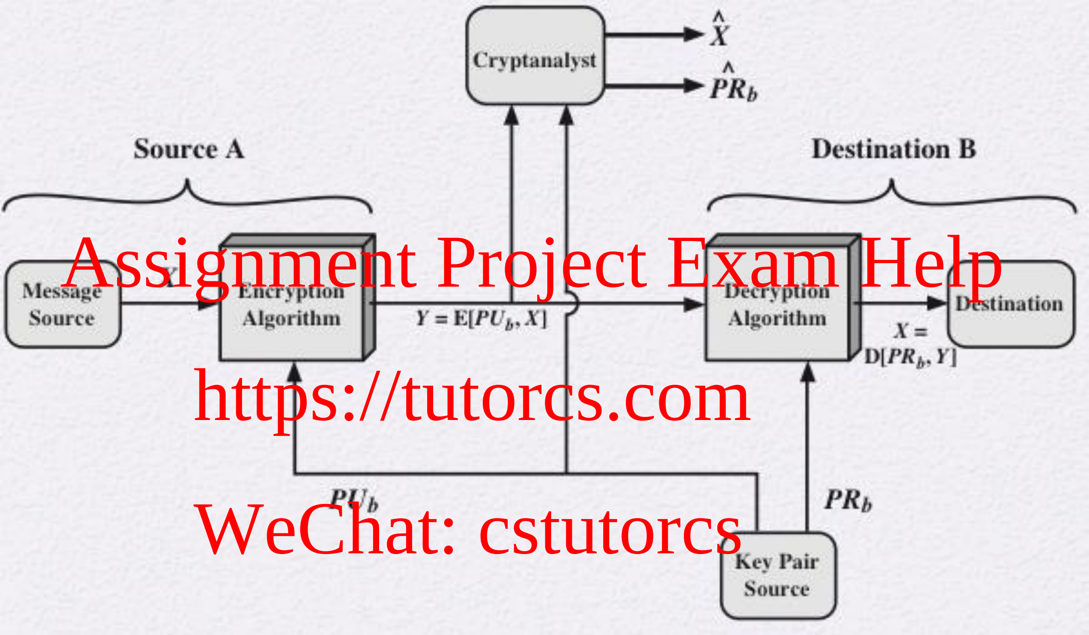
Is RSA secure?

Best known way to compute **d** from **e**
is factoring **N** into **p** and **q**.

Best known factoring algorithm:
General number field sieve
Takes more than polynomial time, but less than exponential time, to factor **n**-bit number.
(Still takes way too long if **p**,**q** are large enough and random.)

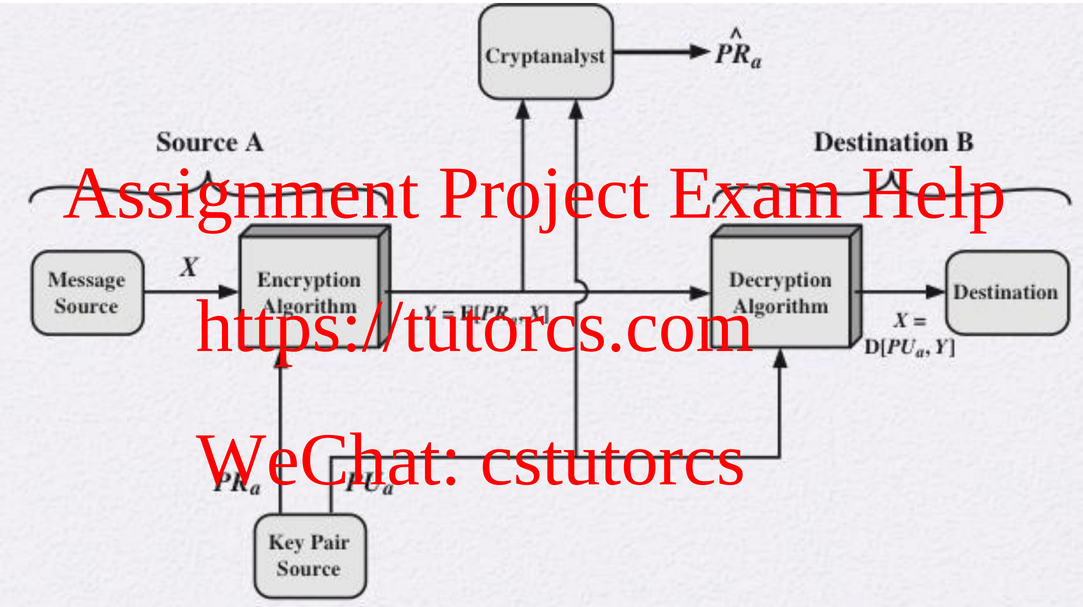
Fingers crossed…
but can't rule out a breakthrough!

Signing with the public key for confidentiality or secrecy:



Does this provide integrity?

Signing with private key for integrity/authentication.



Does this provide confidentiality?

RSA can be used for either **confidentiality** or **integrity**

RSA for confidentiality:

Encrypt with public key
Decrypt with private key
“your eyes only” message

RSA for integrity:

Encrypt (“sign”) with private key
Decrypt (“verify”) with public key
called a **digital signature**

[What if we want both confidentiality
and integrity on the same message?]

Which of these provides both confidentiality and integrity?

Alice (A) wants to send a secret message M to Bob (B) so that Bob can verify that it comes from Alice.

Which one(s) is/are secure?

1. $E(E(M, PR_A), PU_B)$
2. $E(E(M, PU_B), PR_A)$
3. $C=E(M, PR_A)$ $t=E(H(C), PU_B)$
   - Send C||t
4. $C=E(M, PU_B)$ $t=E(H(C), PR_A)$
   - Send C||t

Review: Public-key Crypto

So far, encryption key == decryption key
 "**symmetric key crypto**"

New idea: Keys are distinct.

RSA: **N** := **pq**

Public key is **(e,N)**

Private key is **(d,N)**

To encrypt: $E(\mathbf{x}) = \mathbf{x}^{e} \bmod \mathbf{N}$
To decrypt: $D(\mathbf{x}) = \mathbf{x}^{d} \bmod \mathbf{N}$

RSA for confidentiality:

Encrypt with public key
Decrypt with private key

*RSA for integrity (**digital signatures**):*

Encrypt ("sign") with private key
Decrypt ("verify") with public key

[Cautions?!]

RSA drawback: Performance

Factor of 1000 or more slower than AES.

Dominated by exponentiation – cost
goes up (roughly) as cube of key size.

Message must be shorter than **N**.

[How big should the RSA keys be?]

Use in practice:

Encryption:
Use RSA to encrypt a random **x** < **N**, compute **k** := PRF(**x**), encrypt message using a symmetric cipher and key **k**

Signing:
Compute **v** := PRF(**m**), use RSA to sign a carefully padded version of **v** (many gotchas!)

Almost always should use crypto libraries to get the details right

True or False?

Public-key encryption is a general-purpose technique that has made symmetric encryption obsolete

True or False?

Key distribution is trivial when using public-key encryption, compared to the cumbersome handshaking involved with key distribution centers for symmetric encryption.

Attacks against RSA

1. Brute force: trying all possible private keys
2. Mathematical attacks: factoring
3. Timing attacks: using the running time of decryption
4. Hardware-based fault attack: induce faults in hardware to generate digital signatures
5. Chosen plaintext attack on unpadded RSA

Exercise

Suppose Bob uses RSA crypto with a very large modulus **n** for which the factorization cannot be found in a reasonable amount of time.

Suppose Alice sends a message to Bob by representing each alphabet letter as an integer between 0 and 25 (A->0, …, Z->25) and then encrypting each number separately using RSA with large **e** and large **n**.

Is this method secure?

If yes, why?

If not, how to efficiently attack this encryption method?

Solution

For a set of message block values SM = {0, 1, 2, ..., 25}. The set of corresponding ciphertext block values SC = {0^e mod N, 1^e mod N, ..., 25^e mod N}, and can be computed by everybody with the knowledge of the public key of Bob.

The most efficient attack is to compute M^e mod N for all possible values of M, then create a look-up table with a ciphertext as an index, and the corresponding plaintext as a value of the appropriate location in the table.

Two subtle "textbook" RSA problems:

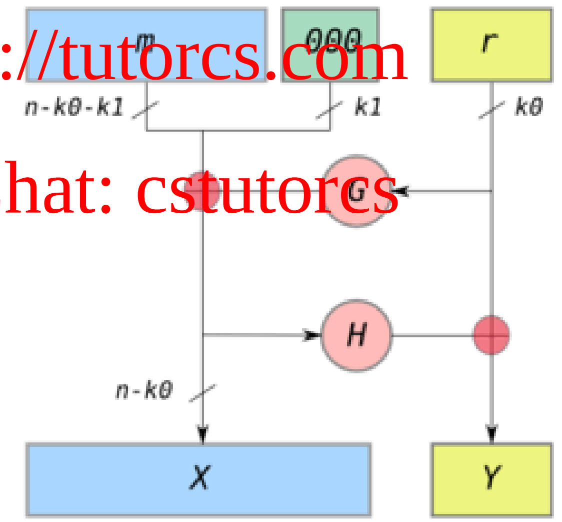
1. For small e and m:
   m^e mod N == m^e
   Trivial to decrypt!

2. If m is chosen from a small set, easy to confirm a ciphertext is a given message (anyone can encrypt!)
   Chosen plaintext attack

Solution: RSA Padding

Need to make sure m is as *large enough* to wrap around N (so can't take e-th root of ciphertext)

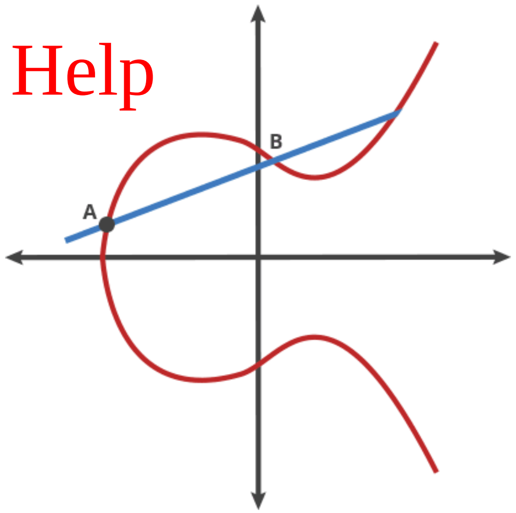
Need to *randomize* before encryption (so low-entropy plaintext can't be decrypted)



Other public key cryptography systems

RSA is popular, but not the only one:

- DSA – Digital Signature Algorithm
- ECDSA – Elliptic Curve Digital Signature Algorithm
  - Very small public keys: e.g. curve25519: 256-bits (32 bytes)
- Post-Quantum Cryptography:
  - Ring-LWE, NTRU, hash-based



So Far:

The Security Mindset

Message Integrity

Confidentiality

Key Exchange

Building a Secure Channel

Public Key Crypto

Next Week:

Begin Web Security Unit

HTTPS: Secure channels for the web