Public-Key Crypto

Review: Integrity

Problem: Sending a message over an **untrusted channel** without being changed

Provably-secure solution: **Random function**

Practical solution:

e.g. "Attack at dawn", 628369867…

Pseudorandom function (**PRF**)

Input: arbitrary-length **k**
Output: fixed-length value

Secure if practically indistinguishable from a random function, unless know **k**

Real-world use:

Message authentication codes (**MACs**) built on cryptographic hash functions

Popular example: **HMAC-SHA256$_k$(m)**

[Cautions?!]

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

[What to do?]

Solution:

Public-key Crypto

So far, key to create tag == key to verify
"**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**")

Many applications

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

Best known, most common public-key algorithm: **RSA**

Rivest, Shamir, and Adleman 1978

Requirements for a public key crypto system to be secure

1. Computationally easy for Alice to generate a key pair: **Sk**, **Pk**

2. Computationally easy for sender Alice to generate the tag ("signature") for message **M**:
   Sig = Sign(**Sk**, **M**)

3. Computationally easy for receiver Bob to verify signature:
   Verify(**Pk**, **M**, **Sig**)

4. Computationally infeasible to guess **Sk** knowing **Pk**.

5. Computationally infeasible to create new valid signatures from **Pk** and **Sig**.

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 $\boldsymbol{D}(\boldsymbol{E}(\mathbf{x})) = \mathbf{x}$

Proof:

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

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

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

Best practice:
N should be at least 2048-bits

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

RSA signatures:

Sig = Sign(**Sk**, **M**)

Verify(**Pk**, **M**, **Sig**)

Use this fact:

D(E(x)) = E(D(x)) = x

D() requires **Sk**, E() requires **Pk**

D() and E() are really the same function:

D(x) = E(x, **Sk**)

E(x) = E(x, **Pk**)

RSA signatures:

Sig = Sign(**Sk**, **M**)

Verify(**Pk**, **M**, **Sig**)

Use this fact:

D(E(x)) = E(D(x)) = x

D() requires **Sk**, E() requires **Pk**

Sig = Sign(**Sk** , **M**) = D(M) = $\mathbf{M}^{\mathbf{d}}$ mod **N**

Verify…?

RSA signatures:

Sig = Sign(**Sk**, **M**)

Verify(**Pk**, **M**, **Sig**)

Use this fact:

D(E(x)) = E(D(x)) = x

D() requires **Sk**, E() requires **Pk**

Sig = Sign(**Sk** , **M**) = D(M) = $\mathbf{M}^{\mathbf{d}}$ mod **N**

Verify(**Pk**, **M**, **Sig**) = (E(**Sig**) == **M**)

= $\mathbf{Sig}^{\mathbf{e}}$ mod **N** = $\mathbf{M}^{\mathbf{de}}$ mod **N** = **M**

Attacks?

Attack: Forgery

Attacker could generate random **Sig** then use **Pk** to compute **M**

$\mathbf{M} = \mathbf{Sig}^{e} \bmod \mathbf{N}$

Attacker doesn't get to pick M, but still can generate valid signed messages without **Sk**

Solution?

Use a one-way function

Hash messages before signing

Sig = Sign(**Sk** , **M**) = D(H(M))
= H(**M**)$^{\mathbf{d}}$ mod **N**

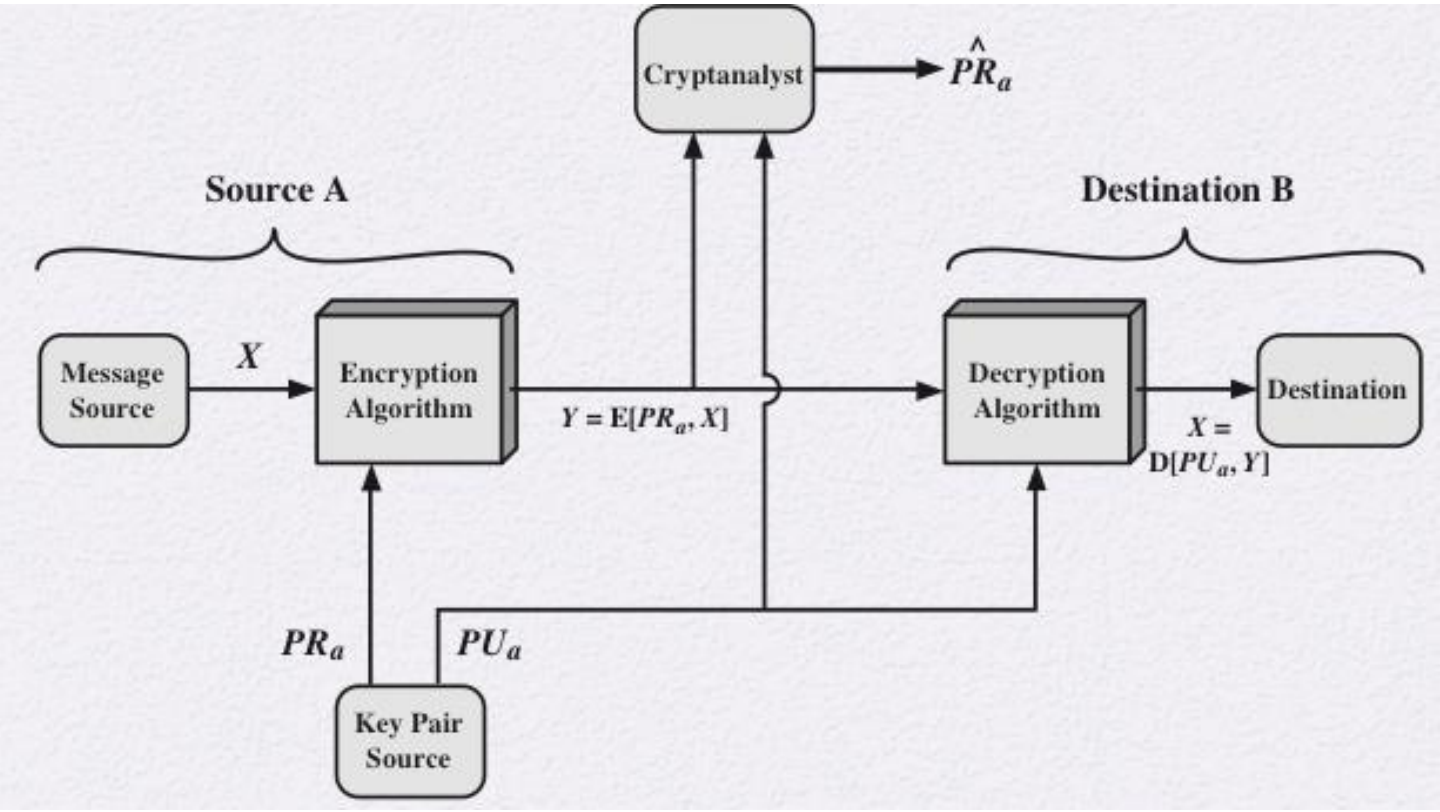
Verify(**Pk**, **M**, **Sig**) = (E(**Sig**) == H(**M**))

= **Sig**$^{\mathbf{e}}$ mod **N =** H(**M**)$^{\mathbf{de}}$ mod **N**
= H(**M**)

Attacker can still pick **Sig** and compute H(**M**),
*but can't compute **M***

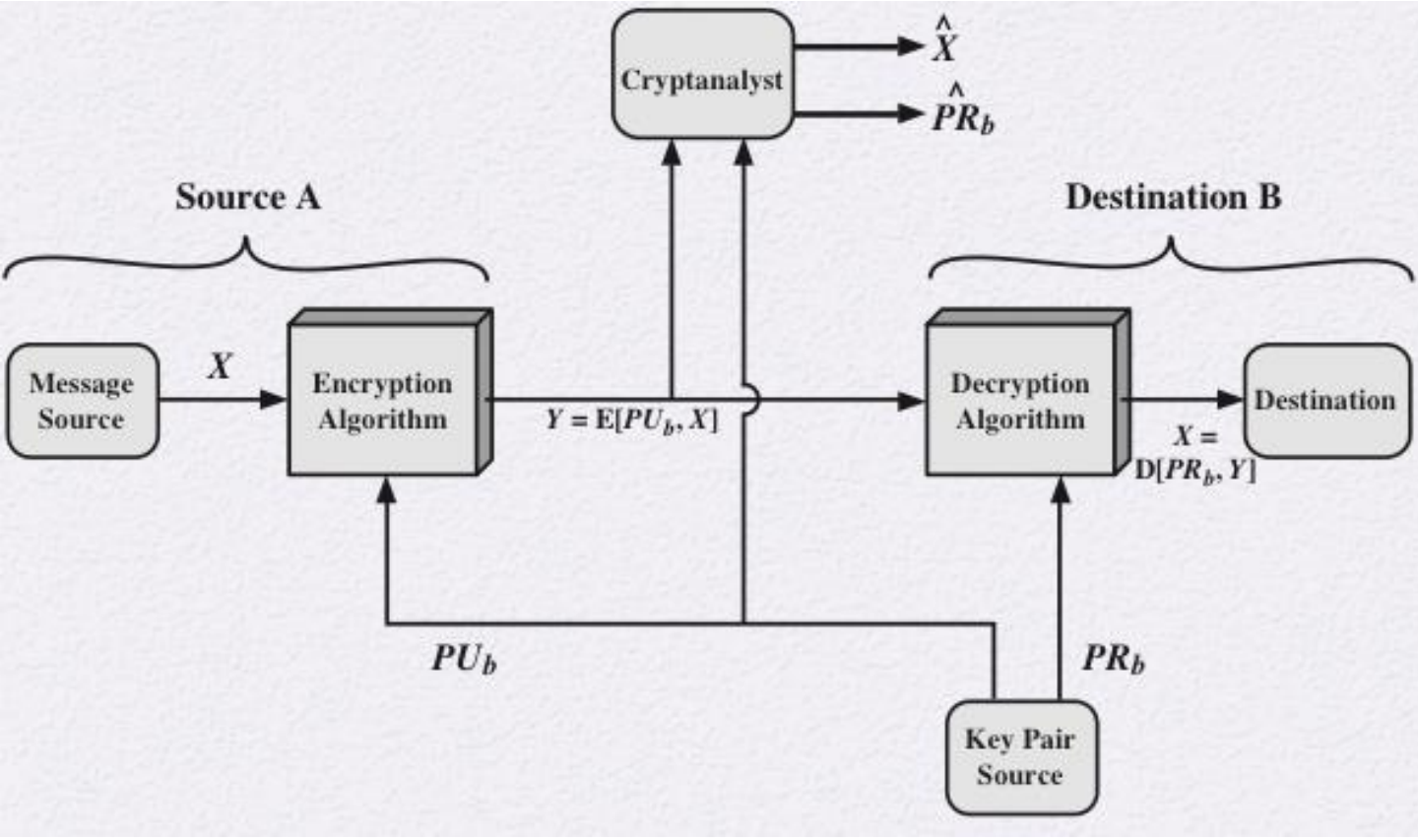
[Why not?]

Signing with private key for **integrity**/authentication.



Does this provide **confidentiality**?

Encrypting with the public key for **confidentiality** or secrecy:



Does this provide integrity?

Subtle fact: 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?]

How to have both confidentiality and integrity (using RSA)?

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

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** := H(**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/ authentication obsolete

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 ciphertext attack

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.

So Far:

The Security Mindset

Message Integrity

Public Key Crypto

Thursday:

Elliptic Curve Cryptography