CSC 580
Cryptography and Computer Security

Public-Key Encryption Idea and Some Supporting Math
(Sections 9.1, 2.4-2.6)

March 13, 2018

Overview

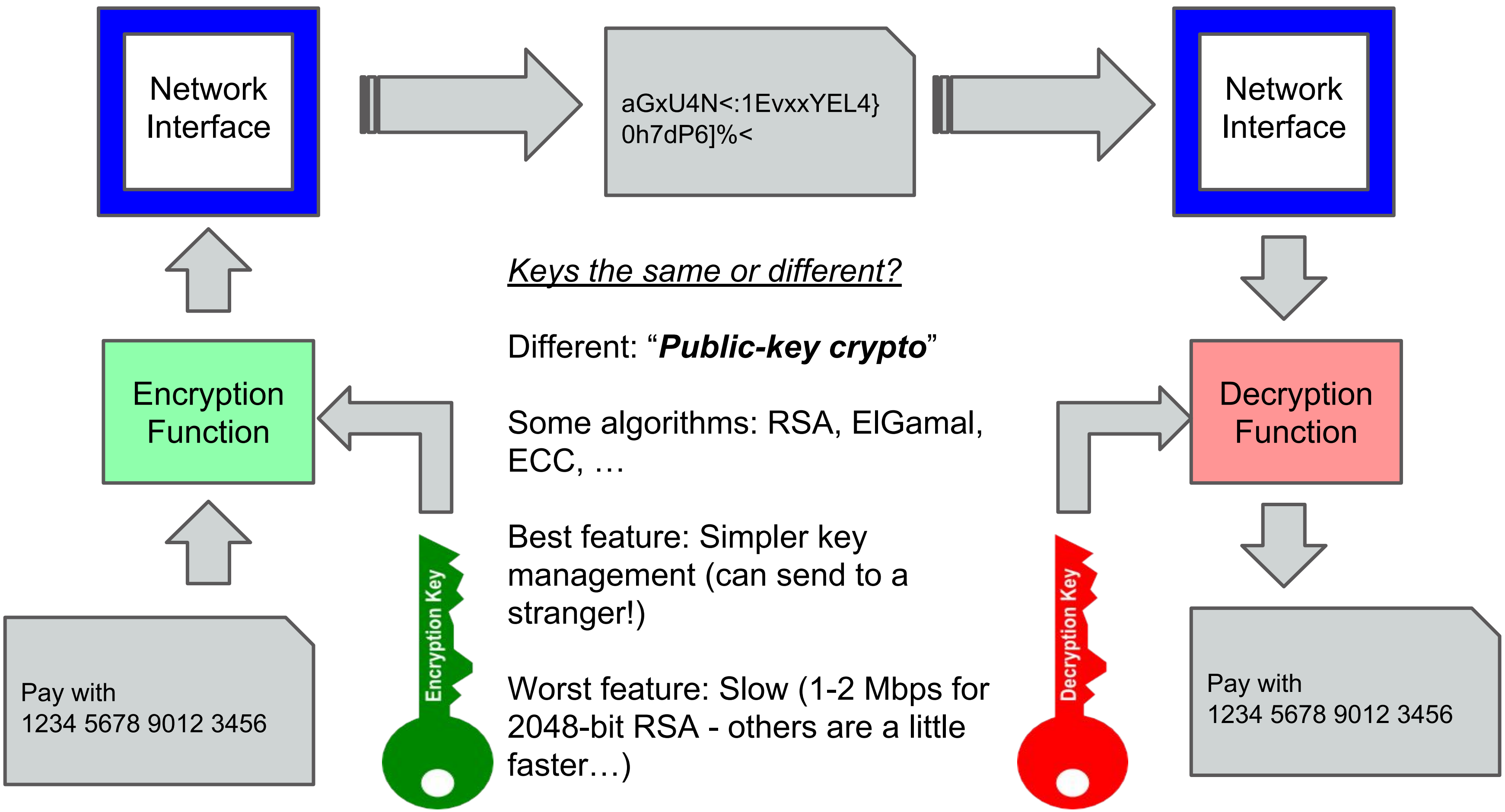
Today:

- Basic idea/motivation for public-key cryptography
- Math needed for RSA (working with prime numbers, etc.)

Next:

- Read Section 9.2 (RSA)
- Don’t forget that you have a graded homework to work on!

Recall Basic Idea



Keys the same or different?

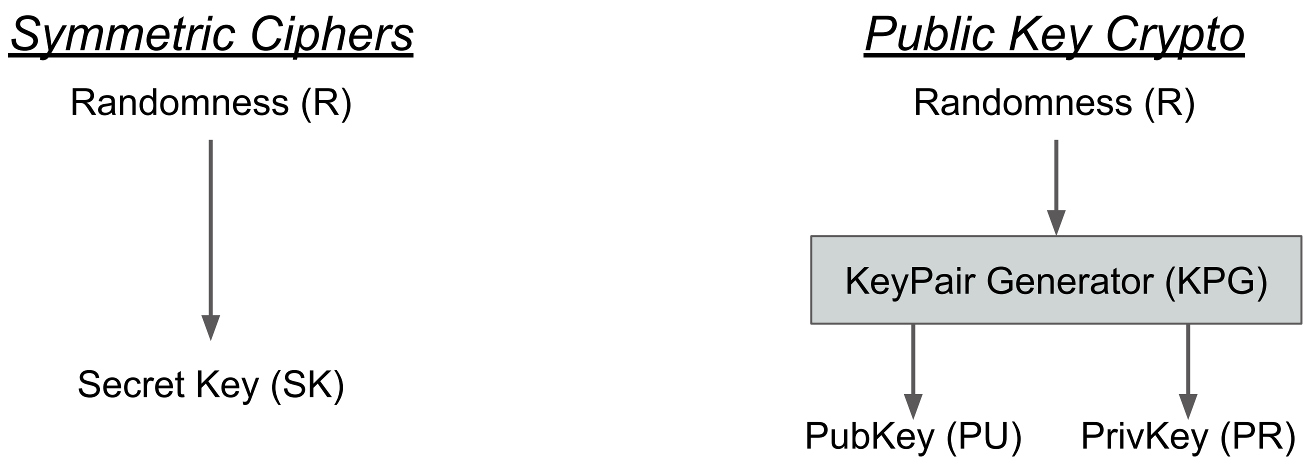
Different: “***Public-key crypto***”

Some algorithms: RSA, ElGamal, ECC, …

Best feature: Simpler key management (can send to a stranger!)

Worst feature: Slow (1-2 Mbps for 2048-bit RSA - others are a little faster…)

Public Key Crypto
Where do the keys come from?



Mathematical/Computational Properties

- KPG(R) → (PU, PR) is efficiently computable (polynomial time)
- For all messages M, D(PR, E(PU, M)) = M   (decryption works)
- Computing PR from PU is computationally infeasible   (we hope!)

Generally: PR has some “additional information” that makes some function of PU easy to compute (which is hard without that info) - this is the “trapdoor secret”

How can this be possible?

To get a sense of how trapdoor secrets help:

Problem: How many numbers $x \in \{1, N-1\}$ have $\gcd(x, N) > 1$ for $N = 32{,}501{,}477$?

(or: how many have a non-trivial common factor with N?)

How could you figure this out?
How long would it take to compute?
What if *N* were 600 digits instead of 8 digits?

How can this be possible?

To get a sense of how trapdoor secrets help:

Problem: How many numbers $x \in \{1, N-1\}$ have $\gcd(x,N)>1$ for N=32,501,477?

(or: how many have a non-trivial common factor with N?)

How could you figure this out?
How long would it take to compute?
What if *N* were 600 digits instead of 8 digits?

What if I told you the prime factorization of *N* is 5,407 * 6,011?

How can this be possible?

To get a sense of how trapdoor secrets help:

Problem: How many numbers $x \in \{1, N\text{-}1\}$ have gcd(x,N)>1 for N=32,501,477?

(or: how many have a non-trivial common factor with N?)

How could you figure this out?
How long would it take to compute?
What if *N* were 600 digits instead of 8 digits?

What if I told you the prime factorization of *N* is 5,407 * 6,011?

5,406 multiples of 6,011 share the factor 6,011 with *N*
6,010 multiples of 5,407 share the factor 5,407 with *N*
No numbers in common between these two sets (prime numbers!)
So… 5,406+6,010 = 11,416 numbers share a factor with 32,501,477

The factorization of *N* is a “trapdoor” that allows you to compute some functions of *N* faster

A Step Toward Public-Key Crypto

So, when solving the problem: *Given a number N, how many positive integers share a non-trivial factor with N*?

- If you know the prime factorization of *N*, this is easy.
- If you don't know the factorization, don't know efficient solution

How does this fit into the public key crypto model?

- Pick two large (e.g., 1024-bit) prime numbers *p* and *q*
- Compute the product $N = p * q$
- Public key is *N* (hard to find *p* and *q*!), private is the pair (p,q)

Questions:

- How do we pick (or detect) large prime numbers?
- How do we use this trapdoor knowledge to encrypt?

Prime Numbers

A prime number is a number *p* for which its only positive divisors are 1 and *p*

Question: How common are prime numbers?

- The Prime Number Theorem states that there are approximately $n / \ln n$ prime numbers less than n.
- Picking a random *b*-bit number, probability that it is prime is approximately $1/\ln(2^b) = (1/\ln 2)*(1/b) \approx 1.44 * (1/b)$
    - For 1024-bit numbers this is about 1/710
    - "Pick random 1024-bit numbers until one is prime" takes on average 710 trials ("pick random *odd* 1024-number" finds primes faster!)
    - This is efficient - if we can tell when a number is prime!

Primality Testing

Problem: Given a number *n*, is it prime?

Basic algorithm: Try dividing all numbers 2,..,sqrt(*n*) into *n*

Question: How long does this take if *n* is 1024 bits?

Fermat's Little Theorem

To do better, we need to understand some properties of prime numbers, such as…

Fermat's Little Theorem: If p is prime and a is a positive integer not divisible by p, then

$$a^{p-1} \equiv 1 \pmod{p} .$$

Proof is on page 46 of the textbook (not difficult!).

Fermat's Little Theorem - cont'd

Explore this formula for different values of *n* and random *a*'s:

a	$a^{n-1} \bmod n$ ($n = 221$)	$a^{n-1} \bmod n$ ($n = 331$)	$a^{n-1} \bmod n$ ($n = 441$)	$a^{n-1} \bmod n$ ($n = 541$)
64	1	1	379	1
189	152	1	0	1
82	191	1	46	1
147	217	1	0	1
113	217	1	232	1
198	81	1	270	1

Question 1: What conclusion can be drawn about the primality of 221?

Question 2: What conclusion can be drawn about the primality of 331?

Primality Testing - First Attempt

Tempting (but incorrect) primality testing algorithm for *n*:

```
Pick random a ∈ {2, ... , n-2}
if a^(n-1) mod n ≠ 1 then return "not prime"
else return "probably prime"
```

Why doesn't this work?

Primality Testing - First Attempt

Tempting (but incorrect) primality testing algorithm for *n*:

```
Pick random a ∈ {2, ... , n-2}
if a^(n-1) mod n ≠ 1 then return "not prime"
else return "probably prime"
```

Why doesn't this work? *Carmichael numbers*....

Example: 2465 is obviously not prime, but →

a	$a^{n-1} \bmod n$ (n = 2465)
64	1
189	1
82	1
147	1
113	1
198	1

Note: Not just for these a's, but $a^{n-1} \bmod n = 1$ for **all** a's that are relatively prime to n.

Primality Testing - Miller-Rabin

The previous idea is good, with some modifications
(Note: This corrects a couple of typos in the textbook):

```
MILLER-RABIN-TEST(n)   // Assume n is odd
    Find k>0 and q odd such that n-1 = 2^k q
    Pick random a ∈ {2, ... , n-2}
    x = a^q mod n
    if x = 1 or x = n-1 then return “possible prime”
    for j = 1 to k-1 do
        x = x^2 mod n
        if x = n-1 then return “possible prime”
    return “composite”
```

If n is prime, always returns “possible prime”
If n is composite, says “possible prime” (incorrect) with probability < ¼

Idea: Run 50 times, and accept as prime iff all say “possible prime”
Question: What is the error probability?

Euler's Totient Function and Theorem

Euler's totient function: $\phi(n)$ = number of integers from 1 .. n-1 that are relatively prime to n.

- If $s(n)$ is count of 1..n-1 that share a factor with n, $\phi(n) = n - 1 - s(n)$
  - $s(n)$ was our "trapdoor function" example
  - $\phi(n)$ easy to compute if factorization of n known
  - Don't know how to efficiently compute otherwise
- If n is product of two primes, $n=p*q$, then $s(n)=(p-1)+(q-1)=p+q-2$
  - So $\phi(p*q) = p*q - 1 - (p+q-2) = p*q - p - q + 1 = (p-1)*(q-1)$

Euler generalized Fermat's Little Theorem to composite moduli:

Euler's Theorem: For every a and n that are relatively prime (i.e., $\gcd(a,n)=1$),

$$a^{\phi(n)} \equiv 1 \pmod{n} .$$

Question: How does this simplify if n is prime?

Next Time...

In the next class we'll see the RSA Public-Key Encryption Scheme uses this!