Number Theory for Asymmetric Crypto

Luke Anderson

luke@lukeanderson.com.au

22nd April 2019

University Of Sydney

Overview

Number Theory

Motivation

Number theory is the basis of a lot of public-key crypto.

- Diffie-Hellman is "secure" because the discrete log problem is hard.
- RSA is "secure" because factoring large numbers is hard.

Core Concept

Find a number theoretic problem that's incredibly difficult to solve if you don't have a key piece of information.

For example:

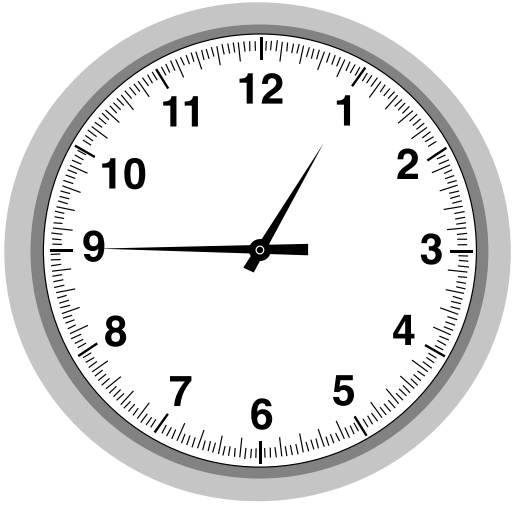
- Multiplying two large primes p, q is easy. Splitting a number $n = pq$ into its factors is hard.
- Raising a number g to the power a is easy. Finding a given only g^a is hard[1].

[1] if all the operations are modulo p

Integers modulo n: $\mathbb{Z}_n$

Fix a number $n \in \mathbb{Z}$, and do arithmetic *modulo* n: keep only the remainder after dividing by n.

$$\begin{aligned} 6 + 6 &\equiv 12 \equiv 0 \pmod{12} \\ 5 - 9 &\equiv -4 \equiv 8 \pmod{12} \\ 5 \times 11 &\equiv 60 \equiv 7 \pmod{12} \end{aligned}$$



- This system of numbers is called $\mathbb{Z}_n$. (The example above is $\mathbb{Z}_{12}$).
- It is finite: each number is uniquely represented as one of

$$\mathbb{Z}_n = \{0, 1, 2, \ldots, n-1\}$$

- If $a, b \in \mathbb{Z}_n$, write simply $a + b$ instead of $a + b \pmod{n}$.

Properties of $\mathbb{Z}_n$

Addition and subtraction in $\mathbb{Z}_n$ are interesting, but well-behaved:

- Addition is cyclic: $\overbrace{1+1+\cdots+1+1}^{n \text{ times}} = 0$. (This turns out to not cause problems)
- Every element $a \in \mathbb{Z}_n$ has a corresponding negative $-a \in \mathbb{Z}_n$:

$$a + (n - a) \equiv n \equiv 0 \pmod{n}$$

Multiplication is badly behaved in general.

- In $\mathbb{Z}_{12}$, $3 \neq 0$ and $4 \neq 0$, but $3 \times 4 = 0$.
- These *zero-divisors* can never have inverses: division not possible.
- However, if we stay away from divisors of n, this problem never occurs.

Extract all of the elements that "multiply nicely" in $\mathbb{Z}_n$: this is the multiplicative group $\mathbb{Z}_n^\times$.

The multiplicative group of $\mathbb{Z}_n$

The multiplicative group $\mathbb{Z}_n^\times$

$$\mathbb{Z}_n^\times = \{a \in \mathbb{Z}_n \mid \gcd(a, n) = 1\}$$

$\mathbb{Z}_n^\times$ is all elements $a \in \mathbb{Z}_n$ such that the $\gcd(a, b) = 1$.

For example:

$$\mathbb{Z}_{21} = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]$$
$$\implies \mathbb{Z}_{21}^\times = \{1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20\}$$

This removes 0 and any elements that share a divisor with 21.

- Addition and subtraction can no longer be done in $\mathbb{Z}_n^\times$.
- Multiplication **and division** work: every number has an inverse!.
- The size of $\mathbb{Z}_n^\times$ is $\phi(n)$, the number of positive integers less than n coprime to n.

Properties of $\mathbb{Z}_n^\times$: Size of the group

The size of the group $\mathbb{Z}_n^\times$ is denoted $\phi(n)$, called *Euler's phi function* or *Euler's totient function*.

- $\phi(n)$ counts the positive integers which are smaller than n and coprime to n.
- If p is prime, then $\phi(p) = p - 1$, since all of $\{1, 2, \ldots, p-1\}$ are coprime to p.
- If n, m are coprime, then $\phi(nm) = \phi(n)\phi(m)$.
- If p, q are distinct primes, then

$$\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$$

It is easy to compute $\phi(n)$ using the prime factorisation of n. *This is essentially the only way to compute it*, and factorisation is very hard in general. The security of RSA *relies* on this fact.

Properties of $\mathbb{Z}_n^\times$: Cyclic multiplication

Multiplication is *cyclic* in the size of the group $|\mathbb{Z}_n^\times| = \phi(n)$.

- For any $x \in \mathbb{Z}_n^\times$, $x^{\phi(n)} = 1$.
  - This means using larger powers than $\phi(n)$ is "useless".
- Some elements may "return to 1" sooner: in $\mathbb{Z}_{21}^\times$, $4^3 = 1$.
- There is sometimes an element $g \in \mathbb{Z}_n^\times$ which "hits all of $\mathbb{Z}_n^\times$", ie $\{g^0, g^1, g^2, \ldots, g^{n-1}\} = \mathbb{Z}_n^\times$.
- Such a g (if it exists) is called a *generator*.
- Generators *always* exist if n is prime.

Most Important Property

For any $x \in \mathbb{Z}_n^\times$, $x^{\phi(n)} = 1$.

This appears in other specialisations/generalisations: Fermat's Little Theorem, Euler's Theorem, Lagrange's Theorem...

Properties of $\mathbb{Z}_n^\times$: Inverses

Every element $a \in \mathbb{Z}_n^\times$ has an inverse: some $b \in \mathbb{Z}_n^\times$ such that $ab = 1$.
Since $a^{\phi(n)} = 1$, this makes $a^{\phi(n)-1}$ the inverse of a: $a^{\phi(n)-1}a = 1$.

Example: Want to invert $11 \in \mathbb{Z}_{21}^\times$.

1. Compute $\phi(21) = \phi(3 \times 7) = \phi(3)\phi(7) = (3-1)(7-1) = 12$.
2. $11^{\phi(21)-1} = 11^{11} \equiv 2 \pmod{21}$.
3. Check: $2 \times 11 = 22 \equiv 1 \pmod{21}$.
4. So $11^{-1} = 2$ in $\mathbb{Z}_{21}^\times$.

Alternatively, inverses can be found *without* needing to know $\phi(n)$.

- By Bézout's identity, there exist $\lambda, \mu \in \mathbb{Z}$ such that

$$\lambda a + \mu n = \gcd(a, n) = 1$$

- Hence $1 \equiv \lambda a + \mu n \equiv \lambda a \pmod{n}$, making λ the inverse of a.

Inverses in $\mathbb{Z}_n^\times$

Each element $a \in \mathbb{Z}_n^\times$ has an *inverse* a^{-1} such that $a \times a^{-1} = 1 \bmod n$.

Each element $a \in \mathbb{Z}_n^\times$, except for 0, is *invertable*.

Simple inversion algorithm[2]

For $\mathbb{Z}_p^*$, where p is prime:

$$x^{-1} = x^{\phi(n)-1} = x^{(p-1)-1} = x^{p-2} \bmod p$$

For $\mathbb{Z}_n^\times$, where $n = pq$:

$$x^{-1} = x^{\phi(p)\phi(q)-1} = x^{(p-1)(q-1)-1} \bmod n$$

[2] from Fermat's little theorem

Example inversion in $\mathbb{Z}_n^\times$

Example:

Given $p = 7$, $q = 3$, and $n = pq = 7 \times 3 = 21$
We select $x = 11$ out of $\mathbb{Z}_{21}^*$ and want to invert it.

$$\begin{aligned} x^{-1} &= x^{(p-1)(q-1)-1} \bmod n \\ &= 11^{(6\times 2)-1} \bmod 21 \\ &= 11^{11} \bmod 21 \\ &= 2 \end{aligned}$$

In Python:

```
p,q,n = 7,3,p*q
xinv = pow(x, (p-1)*(q-1)-1, n)
```

Check that $x \cdot x^{-1} \bmod n = 1$
$11 \times 2 \bmod 21 = 22 \bmod 21 = 1$

GCDs and Bézout's Identity

Bézout's identity is an efficient method of identifying the GCD of two very large numbers.

A few definitions:

- $\mathbb{Z} = \{\ldots, -2, -1, 0, 1, 2, \ldots\}$ the set of integers.
- a *divides* b if there is some $n \in \mathbb{Z}$ such that $an = b$.
- If a divides b, call a a *divisor* of b.
- $p \in \mathbb{Z}$ is *prime* if its only positive divisors are 1 and p.
- The *greatest common divisor* (GCD) of a and b is the largest $d \in \mathbb{Z}$ such that d divides a and d divides b. Call this number $d = \gcd(a, b)$.

Theorem: Bézout's Identity

Let $n, m \in \mathbb{Z}$. Then there exist $\lambda, \mu \in \mathbb{Z}$ such that:

$$\lambda n + \mu m = \gcd(n, m)$$

The numbers λ, μ can be found using the extended euclidean algorithm, which runs in $O(\log n + \log m)$ time and is the oldest recorded algorithm.

Generated Sequences in $\mathbb{Z}_n^\times$

If all elements in $\mathbb{Z}_n^\times$ can be obtained via g using:

$$g^x \bmod n$$

Where $x \in \mathbb{Z}$ (i.e. *any* integer)

Then we state that:

$$g \text{ is a } \textbf{generator} \text{ for } \mathbb{Z}_n^\times$$

$$\mathbb{Z}_n^\times = [1, g, g^2, g^3, \cdots, g^{\phi(n)-1}]$$

The length of the maximum sequence for $\mathbb{Z}_n^\times$ is given by $\phi(n)$.

- If $\mathbb{Z}_p^*$, where p is prime, then $\phi(p) = p - 1$
- If $\mathbb{Z}_n^\times$, where $n = pq$ (a composite prime), then:

$$\phi(n) = \phi(p)\phi(q) = (p-1)(q-1)$$

Note: *the length of the sequence is maximal for* $\mathbb{Z}_p^*$

Generated Sequences in $\mathbb{Z}_n^\times$

Let $g \in \mathbb{Z}_n^\times$. If the order of g is $\phi(n)$, then g is a *generator* of $\mathbb{Z}_n^*$.
i.e. g can produce all the elements in $\mathbb{Z}_n^\times$ by $g^x \bmod n$

Is $g = 2$ a generator for $\mathbb{Z}_7^*$?

$2^1 = 2 \bmod 7 \quad = 2$
$2^2 = 4 \bmod 7 \quad = 4$
$2^3 = 8 \bmod 7 \quad = 1$
$2^4 = 16 \bmod 7 \quad = 2$
$2^5 = 32 \bmod 7 \quad = 4$
$2^6 = 64 \bmod 7 \quad = 1$
$2^7 = 256 \bmod 7 \quad = 2$

$\mathbb{Z}_7^* \neq [1, 2, 4]$
Nope

Is $g = 2$ a generator for $\mathbb{Z}_5^*$?

$2^1 = 2 \bmod 5 \quad = 2$
$2^2 = 4 \bmod 5 \quad = 4$
$2^3 = 8 \bmod 5 \quad = 3$
$2^4 = 16 \bmod 5 \quad = 1$
$2^5 = 32 \bmod 5 \quad = 2$
$2^6 = 64 \bmod 5 \quad = 4$
$2^7 = 256 \bmod 5 \quad = 3$

$\mathbb{Z}_5^* = [1, 2, 3, 4]$
Yes!

Is $g = 4$ a generator for $\mathbb{Z}_5^*$?

$4^1 = 4 \bmod 5 \quad = 4$
$4^2 = 16 \bmod 5 \quad = 1$
$4^3 = 64 \bmod 5 \quad = 4$
$4^4 = 256 \bmod 5 \quad = 1$
$4^5 = 1024 \bmod 5 \quad = 4$
$4^6 = 4096 \bmod 5 \quad = 1$
$4^7 = 16384 \bmod 5 \quad = 4$

$\mathbb{Z}_5^* \neq [1, 4]$
Nope

Python: Arithmetic in $\mathbb{Z}_n$ and $\mathbb{Z}_n^\times$

Arithmetic modulo n can be done easily in Python:

Basic Arithmetic

```
a, b, n = 3, 4, 12

a_plus_b = (a + b) % n
a_minus_b = (a - b) % n
a_times_b = (a * b) % n

a_power_7 = (a ** 7) % n
a_power_7 = pow(a, 7, n) # Much more efficient
```

Python: Finding $\mathbb{Z}_n^\times$

Recall, that $\mathbb{Z}_n^\times$ was just the subset of $\{0, 1, \ldots, n-1\}$ where each number is coprime (has no common factors) with n.

Finding $\mathbb{Z}_n^\times$

```
from fractions import gcd
n = 21
mult_grp = [x for x in range(n) if gcd(n,x) ==  1]
phi_n = len(mult_grp)
```

Produces:

- `z = [1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20]`
- `phi = 12`

Python: Bézout's Identity

The Euclidean algorithm can be extended to not only find the greatest divisor d of two numbers $x, y \in \mathbb{Z}$, but also give the constants in Bézout's identity, i.e. $s, t \in \mathbb{Z}$ such that $sx + ty = d = \gcd(x, y)$.

The extended Euclidean algorithm

```
# Return (d, s, t) so sx + ty = d = gcd(x, y)
# Always returns d >= 0
def bezout(x, y):
    if x == 0:
        return (y, 0, 1) if y >= 0 else (-y, 0, -1)
    d, s, t = bezout(y % x, x)
    return (d, t - (y // x) * s, s)
```

This algorithm runs in $O(\log x + \log y)$ time, making it efficient even for very large numbers ($\geqslant$ 2048 bits).

Python: Inversion in $\mathbb{Z}_n^\times$

Suppose we wish to find the inverse of $x \in \mathbb{Z}_n^\times$.

- $\gcd(x, n) = 1$ and so by Bézout, there are $s, t \in \mathbb{Z}$ with $sx + tn = 1$.
- Modulo n, that becomes $sx \equiv 1 \pmod{n}$, so s is the inverse of x.

Finding an inverse in $\mathbb{Z}_n^\times$

```
# Invert an element x of Zn*
def modinv(x, n):
    d, s, t = bezout(x, n)
    if d != 1:
        raise ValueError("%s is not coprime to %s" % (x, n))
    return s
```

Diffie–Hellman

Diffie–Hellman Key Exchange

Problem: Key exchange over an insecure channel

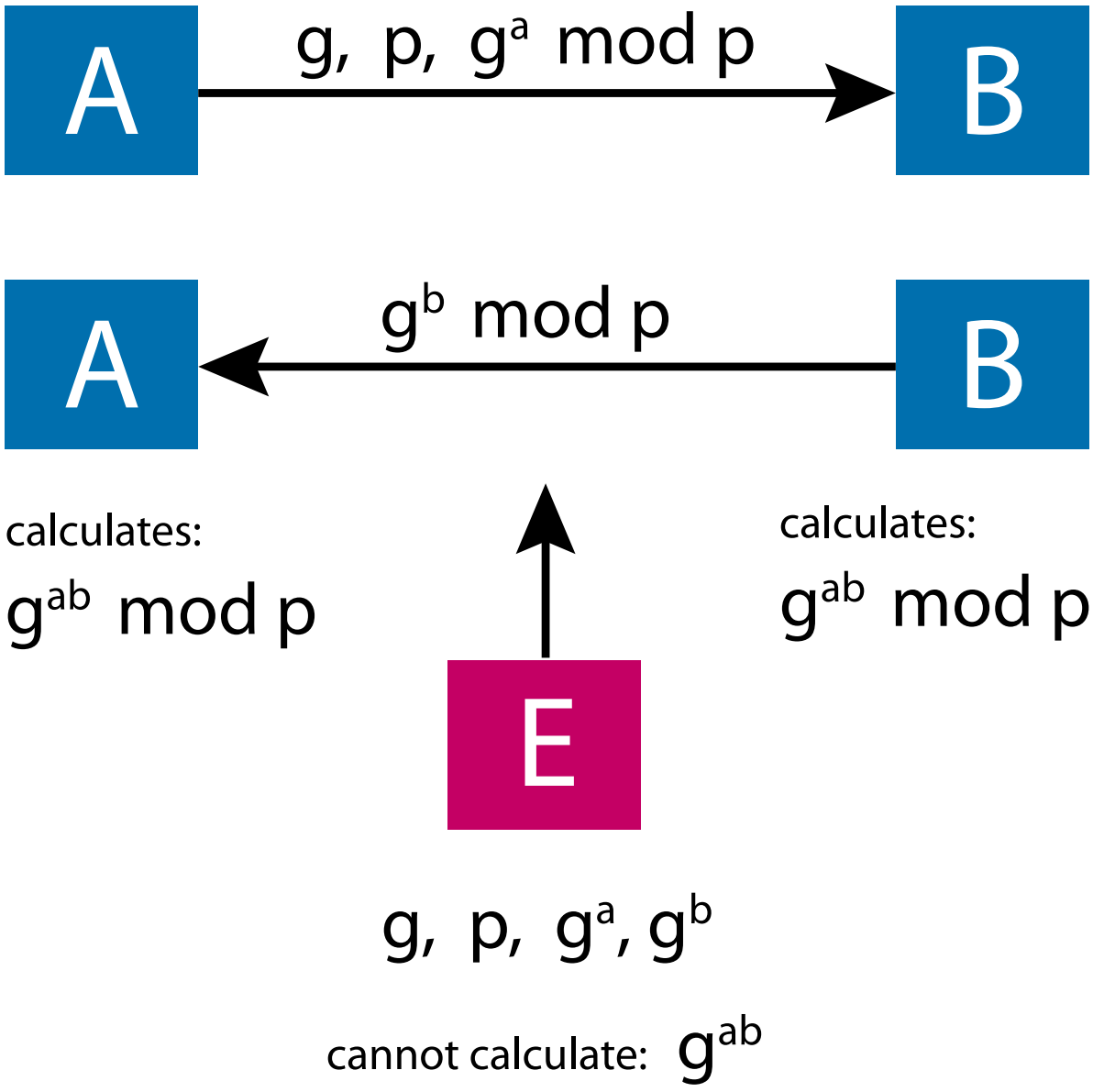
Alice and Bob, talking on an insecure channel, wish to establish a shared key. Eavesdroppers are present and can read all communication between Alice and Bob (but cannot interfere otherwise). Is it possible for Alice and Bob to establish a shared secret?

This problem was solved![3]. A solution is Diffie–Hellman key exchange.

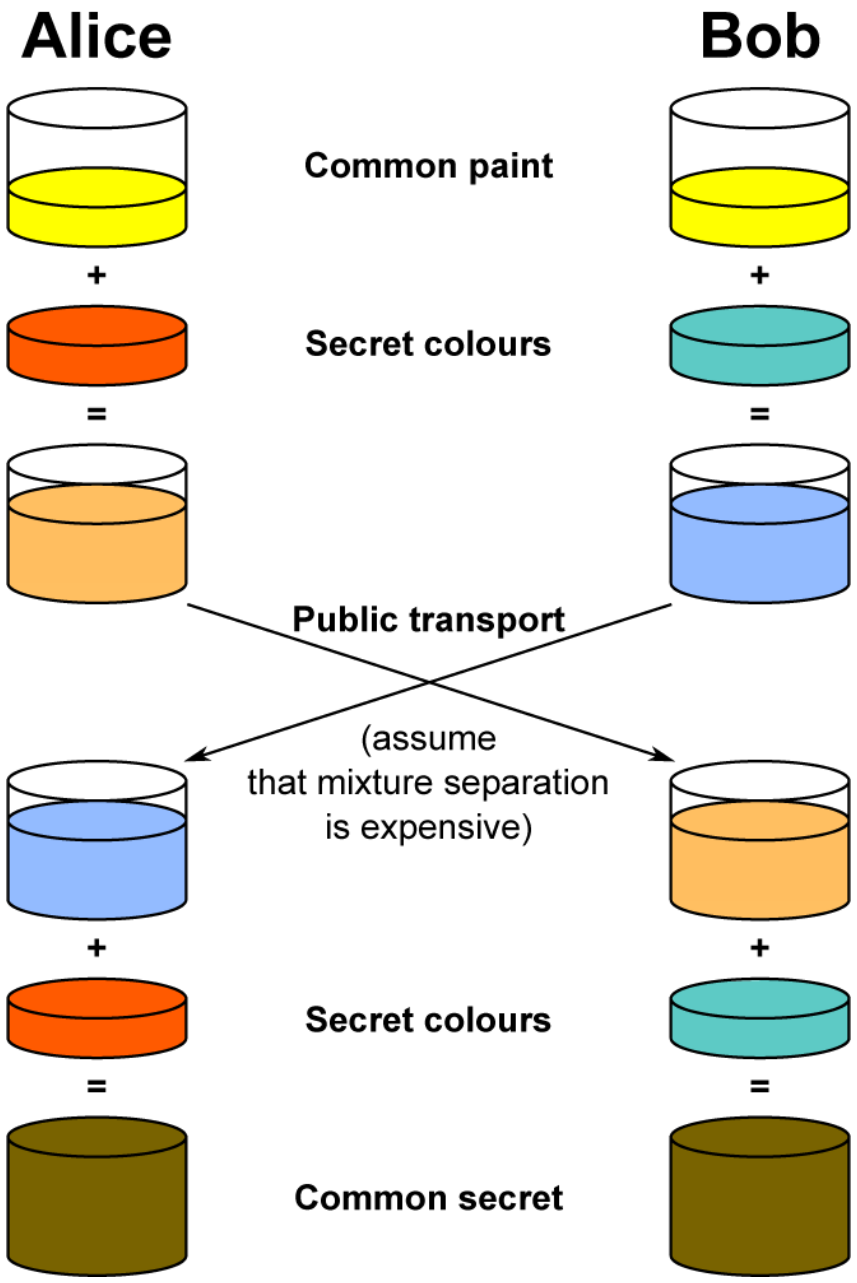
1. Alice and Bob agree on a large prime p, and a generator g of $\mathbb{Z}_p^\times$.
2. Alice picks a random number a $(1 < a < p)$ and sends g^a.
3. Bob picks a random number b $(1 < b < p)$ and sends g^b.
4. Alice and Bob both compute $g^{ab} = (g^a)^b = (g^b)^a$.
5. The shared secret is g^{ab}.

[3] Diffie & Hellman, New directions in cryptography, 1976

Diffie–Hellman Key Exchange



Diffie–Hellman Key Exchange: paint analogy



Computing in $\mathbb{Z}_p^\times$

Computing over $\mathbb{Z}_p^\times$ (where p is prime)

Things that are **easy**:

- Generate a random number
- Multiply two numbers
- Take powers: $g^r \bmod p$
- Find the inverse of an element
- Solve a linear system
- Solve polynomial equation of degree d

Things that are **hard**:

- The Discrete Log Problem (DLP)
- The Diffie–Hellman Problem (DHP)

Hardness of DH

Fix a prime p, and let g be a generator of $\mathbb{Z}_p^\times$.

The *discrete log problem* (DLP) is:

$$\text{Given } x \in \mathbb{Z}_p^\times, \text{ find } r \in \mathbb{Z} \text{ such that } x = g^r \pmod{p}.$$

The *Diffie–Hellman problem* (DHP) is:

$$\text{Given } g^a, g^b \in \mathbb{Z}_p^\times, \text{ find } g^{ab}.$$

An algorithm solving DLP would solve DHP as well.

We assume DLP and DHP are computationally hard, but we have no proofs. Tomorrow an "easy" solution could be uncovered.

Attacks on the Discrete Log Problem (DLP)

Obvious attack: **exhaustive search**

- Linear in the scale of p, since $\mathbb{Z}_p^\times$ has $p - 1$ elements.
- If p has b bits, $p \approx 2^b$. So $O(2^b)$, where b is the key length.

Various methods are more efficient and can calculate in $O(2^{b/2})$:

- Baby-step giant-step (square root) algorithm
  A time-memory trade-off of the exhaustive search method
- Pollard's rho model
- Pohlig-Hellman algorithm

For large values of n, these methods are **not currently practical**.
There does exist an efficient *quantum* algorithm for solving DLP however.

Selecting g and p for Diffie-Hellman

You can select them yourself, but **using RFC standards[4] is preferred**.

A set of Modular Exponential (MODP) groups are defined in RFCs. This means that instead of exchanging g and p each time, you simply state: "`RFC3526 1536-bit`", and both parties know the parameters to use.

- These avoid known attacks against certain classes of primes.

If g and p are chosen well, then the security is in the random choice of a and b.

[4] RFC 3526

RSA

RSA: Operation

Key generation is done by a single party, Alice:

1. Generate two large primes p, q, and let $n = pq$.
2. Select some e coprime to $\phi(n) = (p-1)(q-1)$.
3. Find d such that $ed \equiv 1 \pmod{\phi(n)}$.
4. The *public key* is (n, e) and the *private key* is (n, d).
5. $p, q, \phi(n)$ can now be thrown away.

If Bob now wants to send the message $m \in \mathbb{Z}_n^\times$ to Alice:

1. Bob computes $m^e \pmod{n}$ and sends the result to Alice.
2. Alice receives m^e and computes $(m^e)^d \equiv m^{ed} \equiv m \pmod{n}$.

Proof: Note that since $ed \equiv 1 \pmod{\phi(n)}$, there is some $k \in \mathbb{Z}$ such that $ed = 1 + k\phi(n)$. Now

$$m^{ed} \equiv m^{1+k\phi(n)} \equiv m \cdot (m^{\phi(n)})^k \equiv m \cdot 1^k \equiv m \pmod{n}$$

RSA: Example

Alice generates a new key:

- Choose primes $p = 11$, $q = 17$, and let $n = pq = 187$.
- Select $e = 7$, which is coprime to $\phi(n) = (p-1)(q-1) = 160$.
- Find d such that $ed = 1 \pmod{\phi(n)}$ using the Euclidean algorithm. $d = 23$ works: $ed = 7 \times 23 = 161 \equiv 1 \pmod{160}$.
- Public key: $(n, e) = (187, 7)$, Private key: $(n, d) = (187, 23)$.

Suppose Bob wants to send $m = 142$ to Alice.

- Compute $m^e \pmod{n}$: $142^7 \equiv 65 \pmod{187}$. Send 65 to Alice.
- Alice receives 65, and computes $65^d = 65^{23} \equiv 142 \pmod{187}$. She has recovered Bob's message!

Security of RSA: Calculating in $\mathbb{Z}_n^\times$

Computing over $\mathbb{Z}_n^\times$, where $n = pq$ for primes p and q.

Things that are **easy**:

- Generate a random number
- Multiply numbers
- Take powers: $g^r \bmod n$
- Find the inverse of an element
- Solve a linear system

Things that are **hard**:

- Finding the prime factors of n
- Computing the eth root (as hard as factoring n)
- Solving polynomial equation of degree d

Core security of RSA

The RSA problem

Recovering the value m from $c \equiv m^e \mod n$

(i.e. Taking the e^{th} root of m modulo n)

This is hard: the most efficient means so far known is to factor n into p and q

Integer Factorisation

Given a composite number n, decompose it into its prime factors $p_1, p_2, \ldots p_n$

No polynomial time algorithm is yet known.

Not proven to be "hard"

Like DLP and DHP from DH key exchange, we assume the RSA and integer factorisation problems are computationally "hard" but we have no proofs.

Quantum algorithms already exist that would easily break RSA.

Factoring Attack on RSA

The RSA problem: recover m from $m^e \bmod n$ knowing only n and e.

Suppose n can be factored into p and q:

- Then $\phi(n) = (p-1)(q-1)$ can be found.
- Knowing $\phi(n)$, compute the decryption exponent: $d = e^{-1} \pmod{\phi(n)}$.
- This means we now own the private key, and can decrypt *anything*.

Fact

The problem of computing the RSA decryption exponent from the public key (n, e), and the problem of factoring n are computationally equivalent.

When performing key generation, it is imperative that the primes p and q are selected to make factoring $n = pq$ very difficult
e.g. by picking a p and q that are roughly equal size

Size of Modulus in RSA

Powerful attacks on RSA include using a quadratic sieve and number field sieve factoring algorithms to factor the modulus $n = pq$.

- **1977:** a 129 digit (426 bit) RSA puzzle was published by Martin Gardner's Mathematical Games in Scientific American. Ron Rivest said RSA-125 would take "40 quadrillion years".
  In 1993 it was cracked by a team using 1600 computers over 6 months: "*the magic words are squeamish ossifrage*".
- **1999:** a team led by de Riele factored a 512 bit number
- **2001:** Dan Bernstein wrote a paper proposing a circuit-based machine with active processing units (the same density as RAM) that could factor keys roughly 3 times as long with the same computational cost – so is 1536 bits insecure??
  - The premise is that algorithms exist where if you increase the number of processors by n, you decrease the running time by a factor greater than n.
  - Exploits massive parallelism of small circuit level processing units
- **2009:** RSA-768, 232 digits (768 bits), is factored over a span of 2 years.

Size of Modulus in RSA

In **2012**, a 1061 bit (320 digit) special number ($2^{1039} - 1$) was factored over 8 months (a special case), using a "special number field sieve".

- Unthinkable in 1990
- We have:
  - Developed more powerful computers
  - Come up with better ways to map the algorithm onto the architecture
  - Taken better advantage of cache behaviour
- "Is the writing on the wall for 1024-bit encryption? The answer to that is an unqualified yes" – Lenstra

It is recommended today that 4096 bit keys be used (or at least 2048 bit) and p and q should be about the same bit length (but not *too* close to each other).

Advances in factoring are leaps and bounds over advances in brute force of classical ciphers:

http://en.wikipedia.org/wiki/RSA_Factoring_Challenge

Symmetric vs. Asymmetric

Summary: Symmetric Crypto

Advantages of symmetric-key crypto:

- Can be designed to have high throughput rates
- Keys are relatively short
  (128, ..., 256 bits)
- Can be used as primitives to create other constructs such as pseudo random number generators (PRNGs).
- Can be used to construct stronger ciphers
  e.g. simple substitutions and permutations can be used to create stronger ciphers
- All known attacks involve "exhaustive" key search.

Disadvantages of symmetric-key crypto:

- In a two party network, the key must remain secret at both ends
- Sound practice dictates the key needs to be changed frequently
  (e.g. each session)
- In a large network, $\binom{n}{2}$ keys are required, which creates a massive problem for key management.

Summary: Asymmetric Crypto

Advantages of asymmetric-key crypto:

- Only the private key needs to remain secret
- The administration of keys on a network requires the presence of only one functionally trusted (honest and fair) TTP[5].
- Depending on the mode of usage, the public and private key pairs may be used for long periods of time (upper bound: Moore's Law)
- In large networks, n keys are required instead of $\binom{n}{2}$

Disadvantages of asymmetric-key crypto:

- Throughput rates are typically very slow (for all known algorithms)
- Key sizes are typically much larger (1024, ..., 4096 bits)
- Security is based upon the *presumed* difficulty of a small set of number-theoretic problems; all known are subject to short-cut attacks (e.g. knowing the prime factorisation of n)
- Public key crypto does not have as much of an extensive history in the public world, compared to symmetric.

[5] Trusted Third Party

Combining Cryptosystems

Symmetric and asymmetric crypto are complementary.

Asymmetric crypto can be used to establish a key for subsequent faster symmetric crypto
(e.g. a session key)

Alice and Bob can take advantage of the long term benefits of public key crypto by publishing their public keys to a directory.

Asymmetric crypto is good for key management and signatures.

Symmetric crypto is good for encryption and some data integrity applications.

Symmetric Crypto: Key Length

Security of a symmetric cipher is based on:

- strength of the algorithm
- length of the key

Assuming the strength of the algorithm is perfect (impossible in practice) then brute force is the best attack.

Cost (USD)	40 bits	56 bits	64 bits	128 bits
$100k	0.06 sec	1.1 hrs	11.5 days	10^{18} years
$1M	6.25 ms	6.5 mins	1.2 days	10^{17} years
$100M	0.06 ms	3.75 mins	17 mins	10^{15} years
$1B	6.25 µs	0.4 sec	1.9 mins	10^{14} years

Table: Hardware Attack Estimates (2005)

Interesting ways to break symmetric ciphers

Virus / Worm:

- What if a bot net forced a cipher?
- Melissa infected $\approx$ 800k computers
- If we can cracking DES @ 280k keys/s (P4 @ 2.8 Ghz)
- Melissa-DES could brute force the key in 30 hours
- In 2000, a worm did just that.
- Newer botnets have many more computers.

Chinese Lottery

- Say a 1M key/s chip was built into every radio and TV in China
- Each chip is designed to brute force a key sent over the air
- If 10% of the people in China have a radio or TV:
  the 56 bit DES key space can be exhausted in 12 minutes
- arstechnica.com/security/2017/03/smart-tv-hack-embeds-attack-code-into-broadcast-signal-no-access-required

In future lectures, we'll see how Bitcoin extends on these concepts.

Asymmetric Crypto Key Length

Security of all known public key algorithms is based on the presumed difficulty of a small set of number-theoretic problems.

All are subject to short cut attacks. Tomorrow we might find a way to factor easily.

e.g. Quantum? DNA?

1977: Ron Rivest (RSA) said factoring 125 digit number would take 4×10^{16} years

2003: 576 bit (177 digit) number factored

2012: 1061 bit (320 digit) number factored

Designs are being drawn out for optical / quantum sieving machines that could lead to massive optimisations on these numbers in the near future.

How long should an asymmetric key be?

Protection	**Symmetric**	**Asymmetric**	**Hash**
Attacks in real time by individuals	32	-	-
Short term protection against small organisations	64	816	128
Short term protection against medium organisations	72	1008	144
Very short term protection against agencies, long term protection against small organisations (2016 – 2017)	80	1,248	160
Legacy standard level (2016 – 2020)	96	1,776	192
Medium-term protection (2016 – 2030)	112	2,432	224
Long-term protection (2016 – 2040)	128	3,248	256
"Foreseeable future" good protection against quantum computers unless Shor's algorithm applies	256	15,424	512

Minimal key sizes for different types of crypto (all sizes are in bits)

Considerations for Key Sizes

Type of Traffic	**Lifetime**	**Min. Key Length**
Tactical Military Information	Minutes / hours	64 bits
Product Announcements or M&A	Days / weeks	80 bits
Long Term Business Plans	Years	96 bits
Trade Secrets (e.g. Coca Cola)	Decades	128 bits
H-bomb Secrets	$>$ 40 years	128 – 192 bits
Identities of Spies	$>$ 50 years	128 – 192 bits
Personal Affairs	$>$ 50 years	128 – 192 bits
Diplomatic Embarrassments	$>$ 65 years	192 bits
U.S. Census Data	100 years	256 bits

All key sizes are optimistic. Five or ten years ago people would've suggested smaller key sizes for these tasks.