Week 4

Key Exchange

THE UNIVERSITY OF SYDNEY

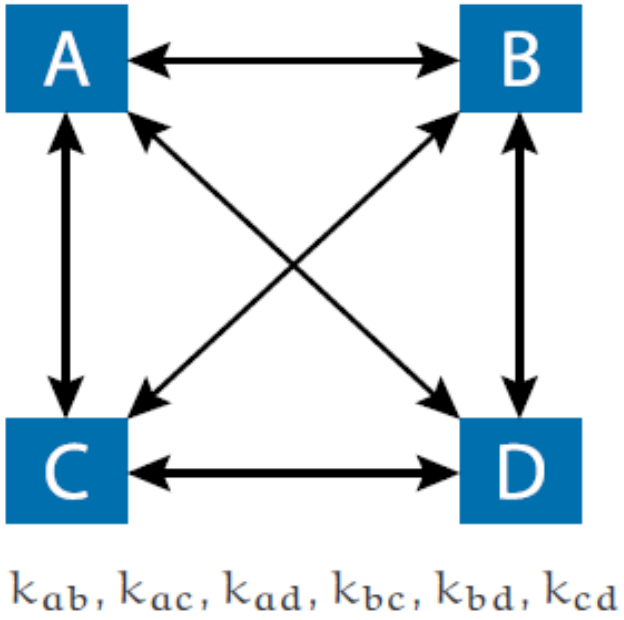
Review

1. Diffie-Hellman Key Exchange
2. Number Theory
3. Attacks on Diffie-Hellman
4. RSA (Rivest–Shamir–Adleman)
5. Symmetric and Asymmetric Encryption

Key Exchange

- Suppose we have a symmetric key network where Alice, Bob, Carol and Dave want to talk to each other.
- For secure communication with n parties, we require:

$$\binom{n}{2} = \frac{n(n-1)}{2} \text{keys}$$

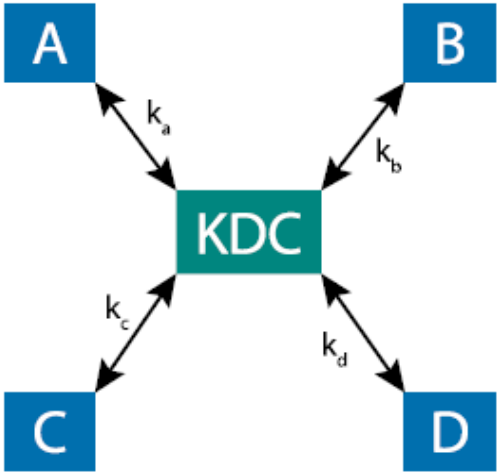


Definitions

- **Key Establishment** is the process whereby a shared key becomes available to two or more parties for subsequent cryptographic use.
- **Key Management** is the set of processes and mechanisms which support key establishment and the maintenance of on-going key relationships between parties, including replacing older keys with newer ones. This includes:
  - Key agreement
  - Key transport

Naïve Key Distribution Centre

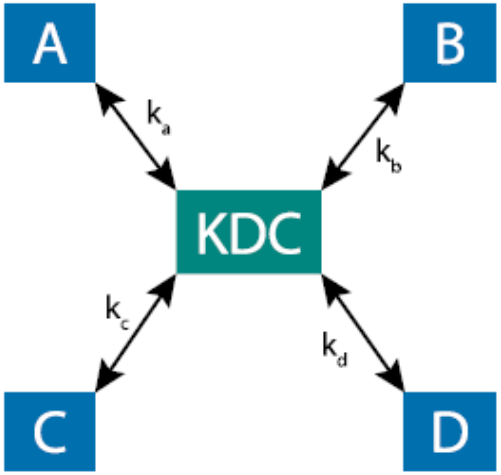
- **Protocol:**
  - 1. Alice → KDC:
    - I want to talk to Bob
  - 2. KDC →Alice:
    - KDC chooses random k_{ab}
    - Returns:
      - $E_{k_a}(k_{ab})$,
      - $E_{k_b}(k_{ab}, \text{“for talking to Alice”})$
  - 3. Alice decrypts $E_{k_a}(k_{ab})$ to get k_{ab}
  - 4. Alice →Bob:
    - $E_{k_b}(k_{ab}, \text{“for talking to Alice”})$
  - 5. Bob decrypts the message using k_b to get k_{ab}
  - 6. Alice & Bob now share k_{ab}



A Key Distribution Centre

Naïve Key Distribution Centre

- **Problems with Naïve KDC:**
  - The Key Distribution Centre is a single point of failure – likely to be attacked
  - No authentication
  - Poor scalability
  - Slow



A Key Distribution Centre

Key Exchange using Merkle's Puzzles

- Merkle's Puzzles are a way of doing key exchange between Alice and Bob without the need for a third party.
  - 1. Alice creates N puzzles $P_1, P_2, \ldots, P_N$, of the form
    $$P_i = E_{p_i}(\text{"This is puzzle \#}X_i\text{"},\ k_i)$$
    - $N \approx 2^{20}$
    - $|p_i| \approx 20$ bits (weak)
    - $|k_i| \approx 128$ bits (strong)
    - X_i, p_i, and k_i are chosen *randomly and different* for each i.
  - 2. Alice sends all puzzles to Bob: $P_1, P_2, \ldots, P_N$.
  - 3. Bob chooses a random puzzle P_j for some $j \in \{1, 2, \ldots, N\}$.
    - Finds p_j by brute force (key space search)
    - Recovers k_j and X_j
    - Bob sends X_j to Alice unencrypted
  - 4. Alice looks up the index of X_j to find the key k_j chosen by Bob.
  - 5. Alice & Bob both share key k_j

Attacking Merkle's Puzzles

- On average, Eve must break half of the puzzles to find which puzzle contains X_j (and hence obtain k_j).
- If there are 2^{20} puzzles, each encrypted with a 20-bit key p_i , Eve must search, on average, half of the key space and puzzles. $2^{19} \times 2^{19} = 2^{38}$
- If Alice and Bob can try 10,000 keys per second:
  - It will take about 1 minute for each to perform their steps: Alice to generate, and Bob to break $p_j = 2^{19}$ keys
  - Plus, another minute to communicate all the puzzles over a wireless link.
- With comparable resources, it will take Eve about a year to break the system.
- Note: Merkle's puzzles uses a lot of bandwidth – impractical!

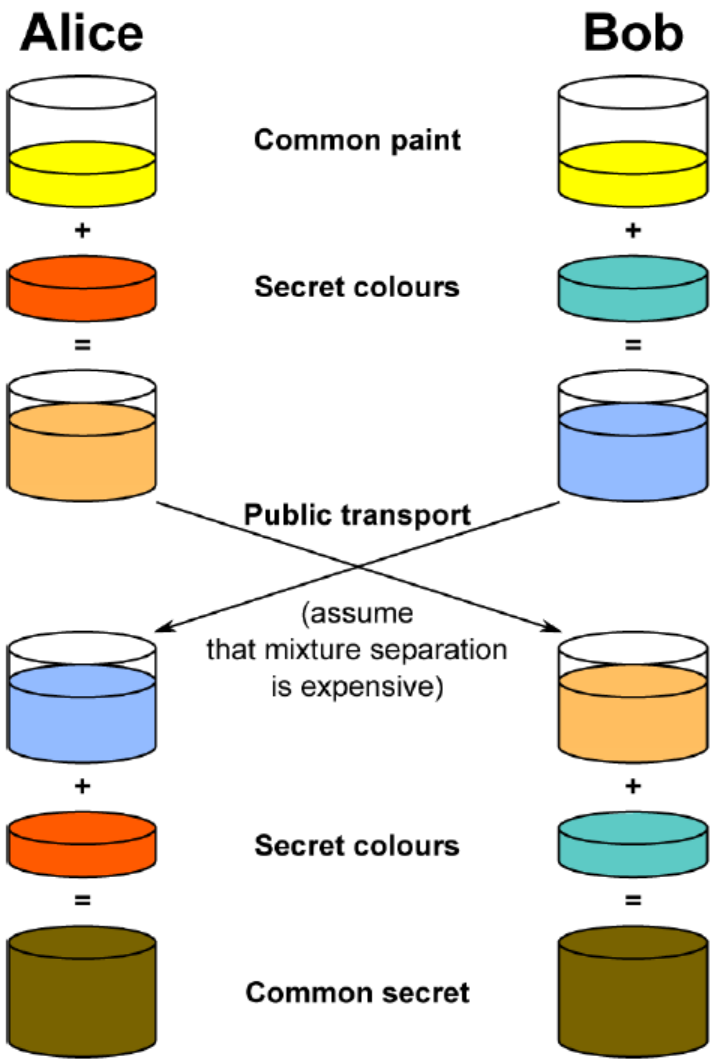
Overview of Diffie-Hellman Key Exchange

THE UNIVERSITY OF SYDNEY

Diffie-Hellman Key Exchange

- Diffie-Hellman key exchange (Stanford, 1976) is an asymmetric encryption protocol for establishing a shared cryptographic key using individual secrets. It is a worldwide standard for use in SSL, smartcards, etc.
- The rough idea is this:
  - 1. Alice and Bob agree on some number g.
  - 2. Alice generates a random number a, and sends g^a to Bob.
  - 3. Bob generates a random number b, and sends g^b to Alice.
  - 4. Alice and Bob can each compute g^{ab}, their shared secret.
- An eavesdropper only has g^a, g^b, and g. Assuming that taking logarithms is hard, the eavesdropper cannot recover either a or b.

Diffie-Hellman Key Exchange



Strength of Diffie-Hellman Key Exchange

- The strength of Diffie-Hellman key exchange is based upon two problems:
  - Discrete logarithm (DL) problem: Given g and g^a, it is difficult to calculate a
  - Diffie-Hellman (DH) problem: Given g, g^a and g^b, it is difficult to calculate g^{ab}
  - Computers perform modular arithmetic (mod p) and modulus is known (p)
  - We know that DL ⇒ DH but it is not known that DH ⇒ DL.

- The security is based on number tricks:
  - The strength is proportional to the difficulty of factoring numbers the same size as p.
  - The generator (g) can be **small.**
  - **Do not** use the secret g^{ab} as the shared key – Not all bits of g^{ab} have a flat distribution. It is better to either hash it or use it as a seed for a PRNG.

Number Theory

THE UNIVERSITY OF SYDNEY

Motivation

- Number theory is the basis of a lot of public-key cryptography.
  - 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:
    - Raising a number g to the power a is easy. Finding a given only g^a is hard if all operations are modulo p
    - Multiplying two large primes p and q is easy. Splitting a number $n = pq$ into its factors, p and q, is hard.

Integers modulo n

- Fix a number $n \in Z$, and do arithmetic modulo n (keep only the remainder after dividing by n), e.g.

  $6 + 6 \equiv 12 \equiv 0 \pmod{12}$

  $5 - 9 \equiv -4 \equiv 8 \pmod{12}$

  $5 \times 11 \equiv 55 \equiv 7 \pmod{12}$

  - This system of numbers is called Z_n. (The example above is Z_{12}).
  - Each number is represented as one of $Z_n = \{0, 1, 2, \ldots, n-1\}$
  - If $a, b \in Z_n$, we can simply write $a + b$ instead of $a + b \ (mod\ n)$.

Properties of modulo n

- Addition and subtraction in Z_n are well-behaved:
  - Addition is cyclic: $n \times (1 + 1 + 1 + 1 + \cdots + 1) = 0$
  - Every element $a \in Z_n$ has a corresponding negative $-a \in Z_n$:

$$a + (n - a) \equiv n \equiv 0 \, (mod \, n)$$

- Multiplication is badly behaved in general.
  - In 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 Z_n: this is the **multiplicative group of n ($Z_n^\times$).**

Multiplicative Group $Z_n^\times$

- The multiplicative group $Z_n^\times$ is all elements $a \in Z_n$ such that the greatest common divisor $\gcd(a, n) = 1$.

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

  - For example:

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

  - This removes 0 and any elements that share a divisor with 21.
    - Addition and subtraction can no longer be done in $Z_n^\times$.
    - Multiplication **and division** work: every number has an inverse!

Size of $Z_n^\times$

- The size of $Z_n^\times$ is $\phi(n)$ which is known as Euler's phi function or Euler's totient function.
  - $\phi(n)$ is the number of positive integers less than n and coprime to n
  - If p is prime, then $\phi(p) = p - 1$, since all of $\{1, 2, \ldots, \mathrm{p} - 1\}$ are coprime to p.
  - If n, m are coprime, then $\phi(nm) = \phi(n)\phi(m)$.
  - If p, q are distinct primes, then $\phi(pq) = \phi(p)\phi(q) = (p-1)(q-1)$

- $\phi(n)$ can be computed 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.

Cyclic Multiplication

- Multiplication is cyclic in the size of the group $|Z_n^\times| = \phi(n)$.
  - **Most important property:**
    - For any $x \in Z_n^\times$, $x^{\phi(n)} = 1$
    - This means using larger powers than $\phi(n)$ is "useless".
  - Some elements may "return to 1" sooner, e.g., in $\mathrm{Z}_{21}^\times$, $4^3 = 1$.
  - There is sometimes an element $g \in Z_n^\times$ which "hits all of $Z_n^\times$", i.e.,

    $$\{g^0, g^1, g^2, \ldots, g^{n-1}\} = \mathrm{Z}_\mathrm{n}^\times$$

  - Such a g (if it exists) is called a *generator.*
  - The length of the maximum sequence created by a generator is given by $\phi(n)$.
  - Generators *always exist* if n is prime.
  - The length of the sequence is maximal if n is prime.

Generated Sequences

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

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

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

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

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

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

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

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

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

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

Inverses

- Every element $a \in Z_n^\times$ is invertible, i.e., there exists some $b \in Z_n^\times$ such that $ab = 1$. b is called the inverse of a and it is denoted as $b = a^{-1}$.
- Since $a^{\phi(n)} = 1$, $a^{-1} = a^{\phi(n)-1} \Leftarrow a^{\phi(n)-1} \times a = 1$
- For $Z_p^\times$, where p is prime, $a^{-1} = a^{\phi(p)-1} = a^{(p-1)-1} = a^{p-2} \bmod p$
- Example: Want to invert $11 \in Z_{21}^\times$.
  - 1. Compute $\phi(21) = \phi(3 \times 7) = \phi(3)\phi(7) = (3-1)(7-1) = 12.$
  - 2. $11^{\phi(21)-1} = 11^{12-1} = 11^{11} \equiv 2 \pmod{21}.$
  - 3. Check: $2 \times 11 = 22 \equiv 1 \pmod{21}.$
  - 4. So, $11^{-1} = 2$ in $Z_{21}^\times$.

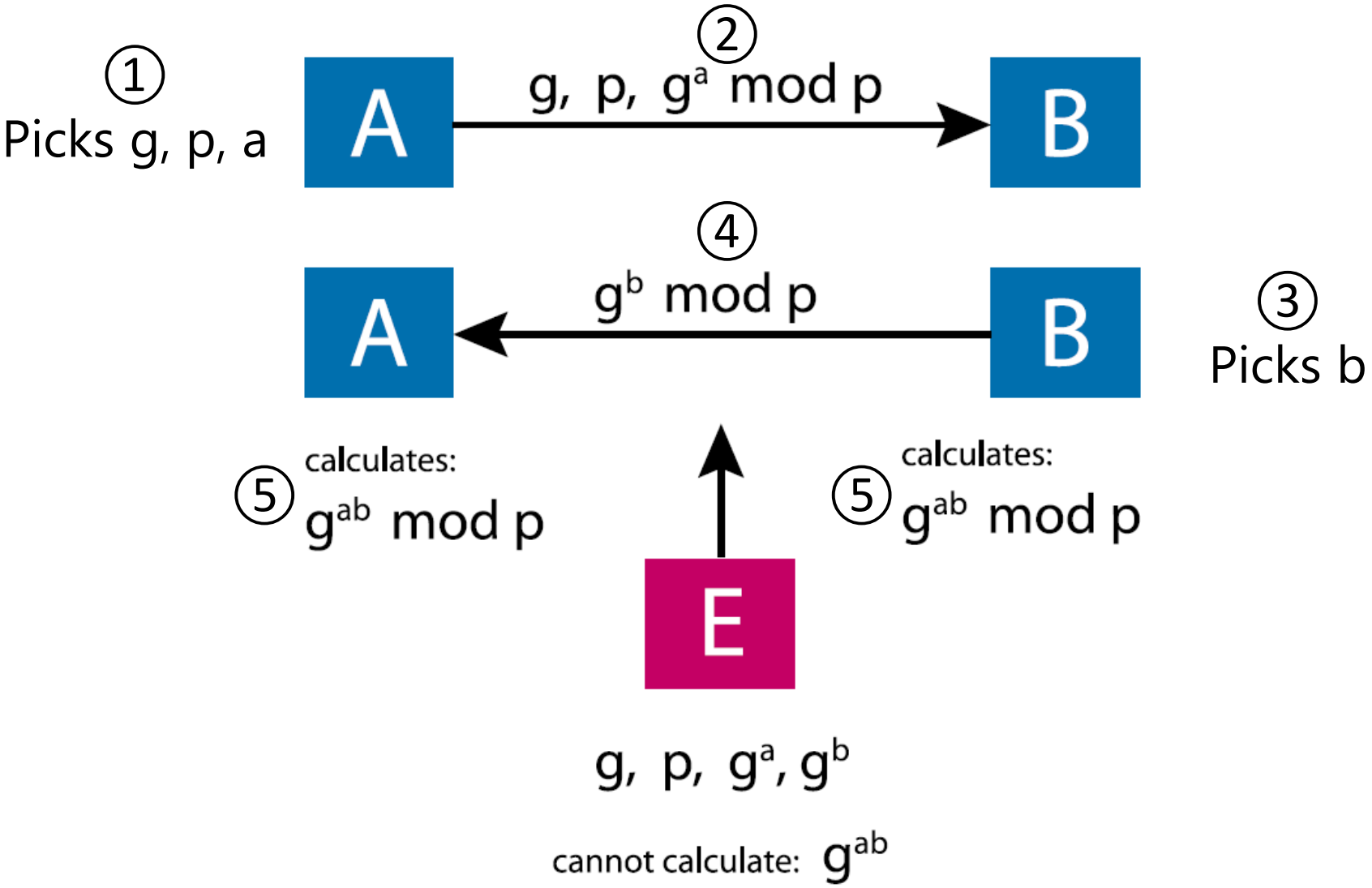
Diffie-Hellman Key Exchange

THE UNIVERSITY OF SYDNEY

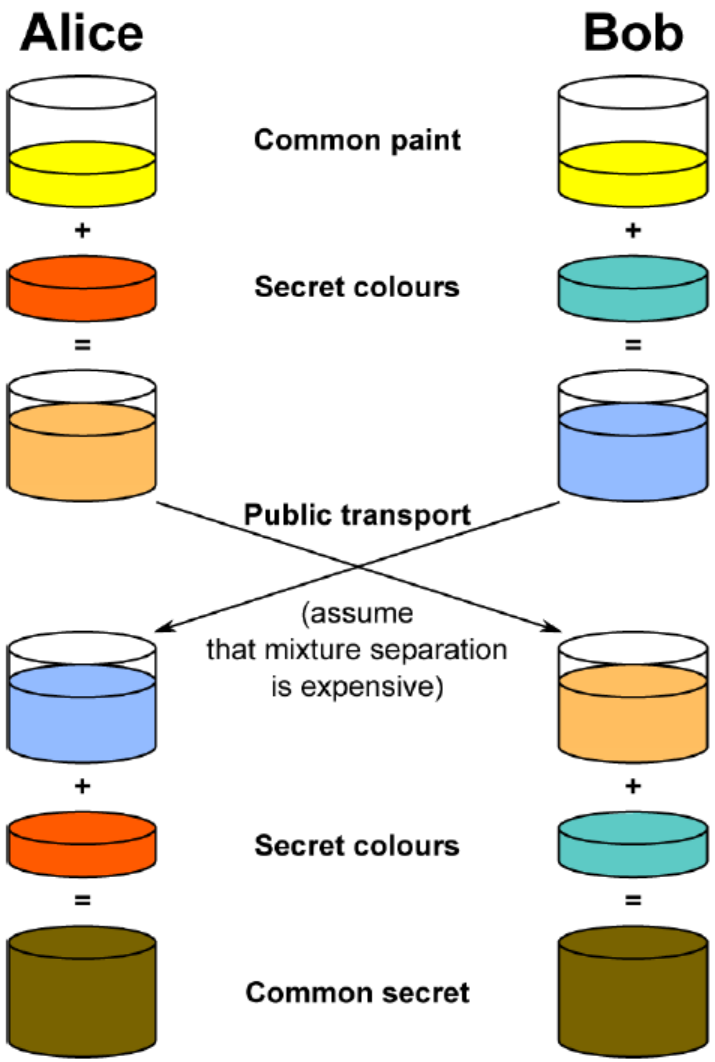
Diffie–Hellman Key Exchange

- 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?
- A solution is **Diffie–Hellman key exchange**.
  - 1. Alice and Bob agree on a large prime p, and a generator g of $Z_p^\times$.
  - 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 $g^{ab} = (ga)^b = (g^b)^a$.
  - 5. The shared secret is g^{ab}.

Diffie–Hellman Key Exchange



Diffie–Hellman Key Exchange



Computing in $Z_p^\times$

- Computing over $Z_p^\times$ (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 Diffie-Hellman

- Fix a prime p and let g be a generator of $Z_p^\times$.
- The discrete log problem (DLP) is:

  Given $x \in Z_p^\times$, find $r \in Z$ such that $x = g^r \pmod p$.
- The Diffie–Hellman problem (DHP) is:

  Given $g^a, g^b \in Z_p^\times$, 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. An "easy" solution could be uncovered in the future.

Attacks on the Discrete Log Problem (DLP)

- Obvious attack: exhaustive search
  - Linear in the scale of p, since $Z_p^\times$ 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 known as Shor's Algorithm.

Selecting g and p for Diffie-Hellman

- You can select them yourself but using RFC standards (RFC 3526) is preferred. https://tools.ietf.org/html/rfc3526
- 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 MODP Group", 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.

RSA

THE UNIVERSITY OF SYDNEY

RSA: Operation

- RSA is an asymmetric encryption algorithm with authentication. Key generation is done by a single party, Alice:
  - 1. Generate two large primes p, q, and define modulus $n = pq$.
  - 2. Select some e coprime to $\phi(n) = (p-1)(q-1)$.
  - 3. Find d such that $e \times d \equiv 1 \pmod{\phi(n)}$.
  - 4. The public key is (n, e) and the private key is (n, d).
  - 5. $p, q, \phi(n)$ are no longer needed.
- If Bob now wants to send the message $m \in Z_n^\times$ to Alice:
  - 1. Bob computes a ciphertext $C = m^e \pmod{n}$ and sends the result to Alice.
  - 2. Alice receives m^e and decrypts the ciphertext to recover the message by computing $(m^e)^d \equiv m^{ed} \equiv m \pmod{n}$. Recall that we found d such that $e \times d \equiv 1 \pmod{\phi(n)}$.

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 $ed = 1\ (mod\ \phi(n))$
    $d = 23$ 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 $65^d = 65^{23} \equiv 142\ (mod\ 187)$. She has recovered Bob's message!

Security of RSA

- The RSA problem
  - Hard to recover the value of m only from $c \equiv m^e \bmod n$ (i.e. taking the e^{th} root of c modulo n). The most efficient means so far known is to use integer factorization of 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.

- The security of RSA is not formally 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 problem of computing the RSA decryption exponent m from $m^e \, mod \, n$ with only knowledge of the public key (n, e), and the problem of factoring n are computationally equivalent.

- 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:

$$d = e^{-1}(mod \, \phi(n))$$

  - This means we now own the private key and can decrypt anything.

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

Size of Modulus in RSA

- Powerful attacks on RSA to factor the modulus $n = pq$.
  - 1977: A 129-digit (426 bit) RSA puzzle was published in Scientific American which was estimated to take "40 quadrillion years" using the fastest algorithm and computers at the time. 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 ($21039 - 1$) was factored over 8 months (a special case), using a "special number field sieve".
  - Unthinkable in 1990. We now have:
    - Developed more powerful computers
    - Developed better ways to map the algorithm onto the architecture
    - Taken better advantage of cache behaviour
  - 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 and Asymmetric Encryption

THE UNIVERSITY OF SYDNEY

Review: Symmetric Encryption

- Advantages of symmetric-key cryptography:
  - 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 cryptography:
  - 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.

Asymmetric Cryptography

- Advantages of asymmetric-key cryptography:
  - 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) trusted third party (TTP).
  - 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 cryptography:
  - 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.

Combining Cryptosystems

- Symmetric and asymmetric cryptography are complementary.
  - Asymmetric cryptography can be used to establish a key for subsequent faster symmetric cryptography (e.g. a session key)
  - Alice and Bob can take advantage of the long-term benefits of public key cryptography by publishing their public keys to a directory.
- Asymmetric cryptography is good for key management and signatures.
- Symmetric cryptography is good for encryption and some data integrity applications.

Symmetric Cryptography: 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)

Asymmetric Cryptography: Key Length

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

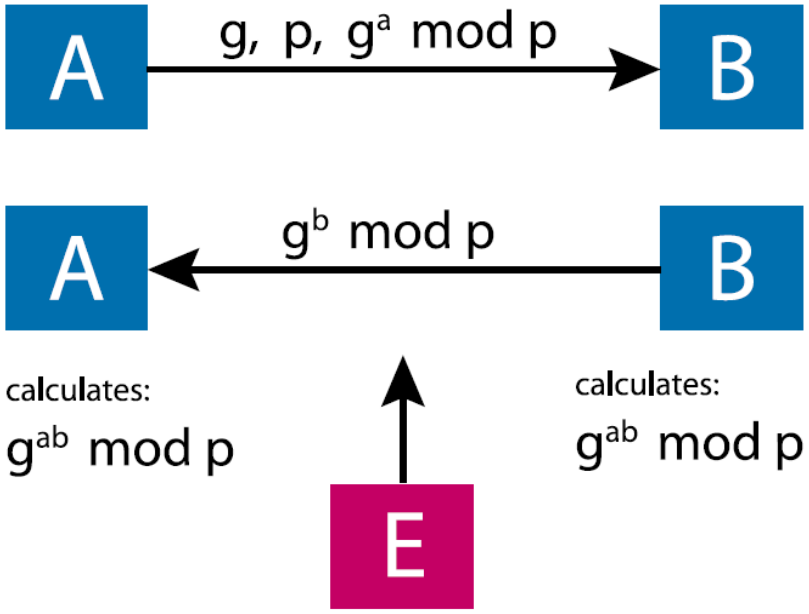
Type of Traffic	Lifetime	Min. Key Length
Tactical Military Information	Minutes / hours	64 bits
Product Announcements or M&A	Days / weeks	80 bits
Long Term Business Plans	Years	96 bits
Trade Secrets (e.g. Coca Cola)	Decades	128 bits
H-bomb Secrets	$>$ 40 years	128 – 192 bits
Identities of Spies	$>$ 50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

Questions

- How many keys are needed for secure communications between Alice, Bob, Carol and Dave in a symmetric key network?

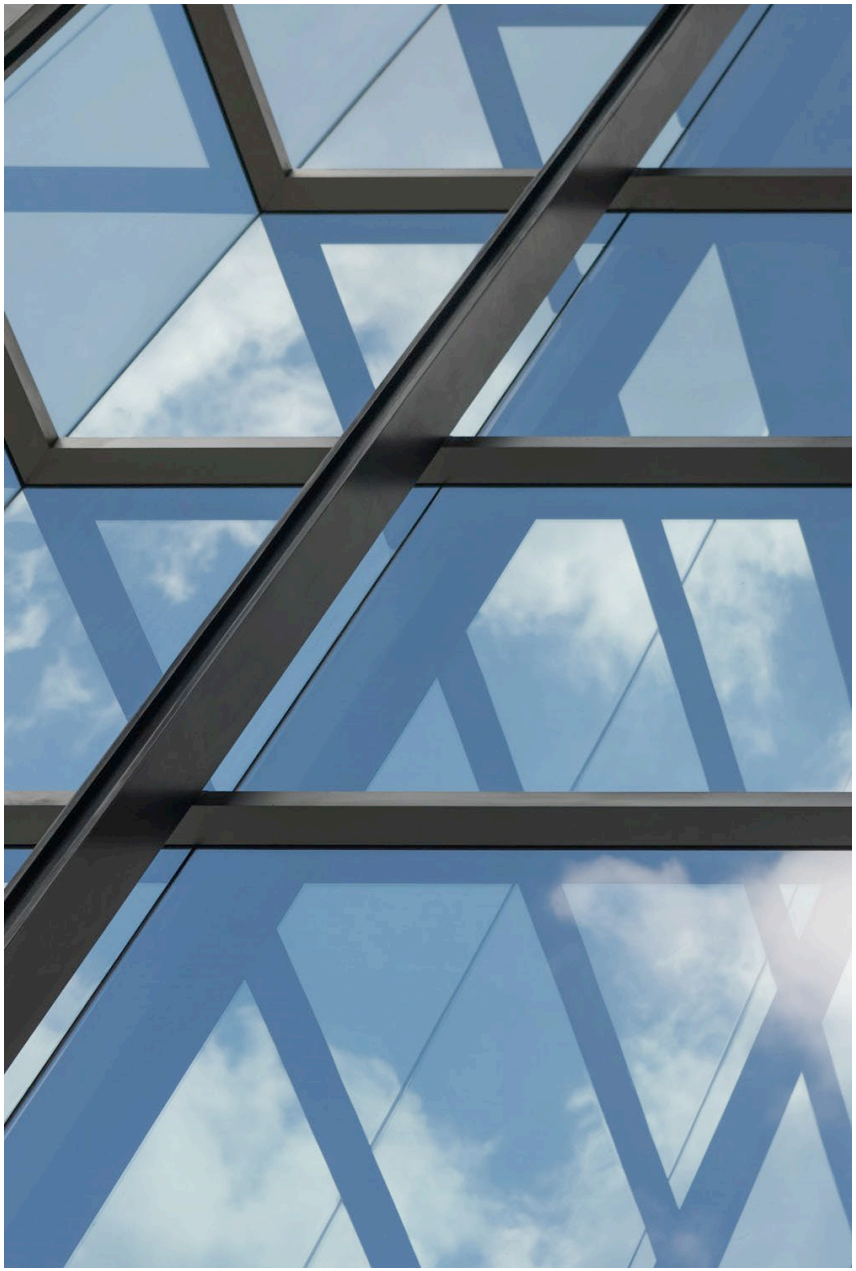
- Explain the Diffie-Hellman problem using the diagram below.



Questions

- How can we calculate a multiplicative group size given modulus $n = pq$?
- What is the strength of RSA based on? How can we be confident that it is secure?

THE UNIVERSITY OF SYDNEY



Number Theory operations in Python

THE UNIVERSITY OF SYDNEY

Arithmetic modulo n

- Arithmetic modulo n can be done easily in Python:

```
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
```

Finding $Z_n^\times$

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

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

- Outputs:

  mult_grp = [1, 2, 4, 5, 8, 10, 11, 13, 16, 17, 19, 20]

  phi_n = 12

Bézout's Identity

- Bézout's Identity is an efficient method of identifying the greatest common divisor (GCD) of two very large numbers.
- Bézout's Identity:
  - Let $a, n \in Z$. Then, there exists $\lambda, \mu \in Z$ such that

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

  - If $\gcd(a, n) = 1$, then,

$$1 \equiv \lambda a + \mu n \equiv \lambda a (\mathrm{mod}\, n) + \mu n (\mathrm{mod}\, n) \equiv \lambda a (\mathrm{mod}\, n)$$

Bézout's Identity

- The Euclidean algorithm can used to find the greatest divisor d of two numbers $x, y \in Z$, and also give the constants in Bézout's identity, i.e. $s, t \in Z$ such that $sx + ty = d = \gcd(x, y)$.

```
# 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).

Inversion in $Z_n^\times$

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

```
# 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
```

  - Alternatively, we can use $x^{-1} = x^{\phi(n)-1} = x^{(p-1)(q-1)-1}$, If p, q are distinct primes, then $\phi(n) = \phi(p)\phi(q)$

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