Asymmetric key cryptography & RSA

Deepak Puthal

Email: Deepak.Puthal@uts.edu.au

41900 – Fundamentals of Security

Overview

- **Asymmetric key cryptography**
  - Working model
  - Properties
- **Diffie–Hellman**
  - Diffie–Hellman Key Exchange
  - Hardness of DH
  - Security of DH
  - Choosing DH parameters
- **RSA**
  - RSA Cryptosystem
  - Security of RSA
    - Size of Modulus in RSA
- **Symmetric vs. Asymmetric**
  - Symmetric Crypto
  - Asymmetric Crypto
  - Combining Cryptosystems
  - Symmetric Key Sizes
  - Interesting breaks on Symmetric Crypto
  - Asymmetric Key Sizes
  - Considerations for Key Sizes

Asymmetric key cryptography

- Symmetric and asymmetric-key cryptography will exist in parallel and continue to serve the community. We actually believe that they are complements of each other; the advantages of one can compensate for the disadvantages of the other.
- Asymmetric key cryptography uses two separate keys: one private and one public.

$$C = f(K_{public}, P) \qquad P = g(K_{private}, C)$$

- **Symmetric-key cryptography is based on sharing secrecy.**
- **Asymmetric-key cryptography is based on personal secrecy.**

Public Key Cryptography

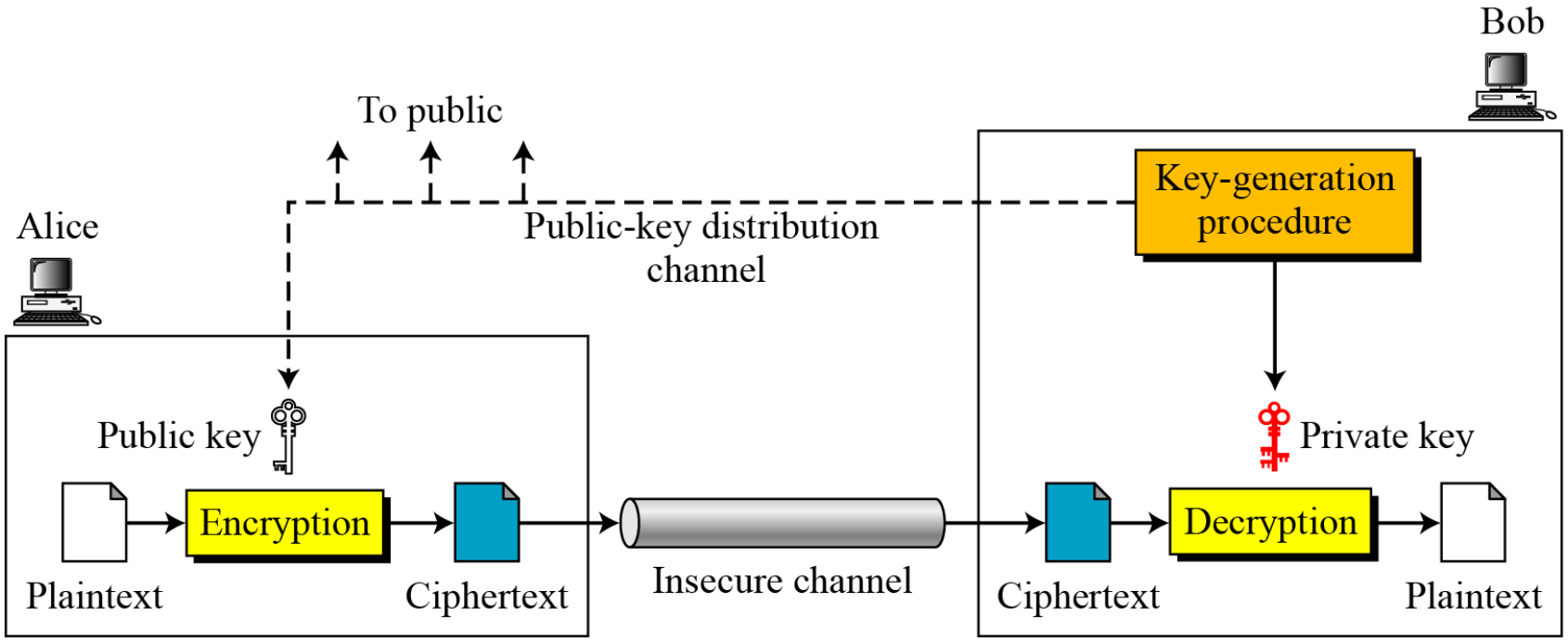
- Unlike symmetric key algorithms, it does not require a secure initial exchange of one or more secret keys to both sender and receiver.
- The asymmetric key algorithms are used to create a mathematically related key pair: a secret private key and a published public key. Use of these keys allows protection of the authenticity of a message by creating a digital signature of a message using the private key, which can be verified using the public key.
- It also allows protection of the confidentiality and integrity of a message, by public key encryption, encrypting the message using the public key, which can only be decrypted using the private key.

Public Key Cryptography

- Public key cryptography is a fundamental and widely used technology around the world. It is the approach which is employed by many cryptographic algorithms and cryptosystems.
- Each user has a pair of cryptographic keys—a public key and a private key. The private key is kept secret, whilst the public key may be widely distributed.
- Messages are encrypted with the recipient's public key and can *only* be decrypted with the corresponding private key. The keys are related mathematically, but the private key cannot be feasibly derived from the public key.

Asymmetric cipher

- Also known as public key
- More computationally intensive



Asymmetric Encryption

Two most popular algorithms are RSA & ElGamal

- RSA
  - Developed by Ron Rivest, Adi Shamir, Len Adelman
  - Both public and private key are interchangeable
  - Variable Key Size (512, 1024, or 2048 buts)
  - Most popular public key algorithm
- ElGamal
  - Developed by Taher ElGamal
  - Variable key size (512 or 1024 bits)
  - Less common than RSA, used in protocols like PGP

Why Asymmetric key?

- The reason public keys are used is to establish secure communication when there is no way to exchange a key beforehand.
  - Confidential/authenticated channels for free?
- Must ensure that the public key belongs to the correct party (binding of identity to key). The public key directory may be corrupted:
  - Solution: Use a Public Key Infrastructure to certify your keys (PKI)
- Anybody may encrypt messages that only destination device may read, since he knows the private key ($K_{private}$)

Summary

- 2 different keys are used
- Users get the Key from an Certificate Authority

- Advantages
  - More Secured
  - Authentication

- Disadvantages
  - Relatively Complex

Diffie–Hellman Key Exchange

Diffie–Hellman Key Exchange

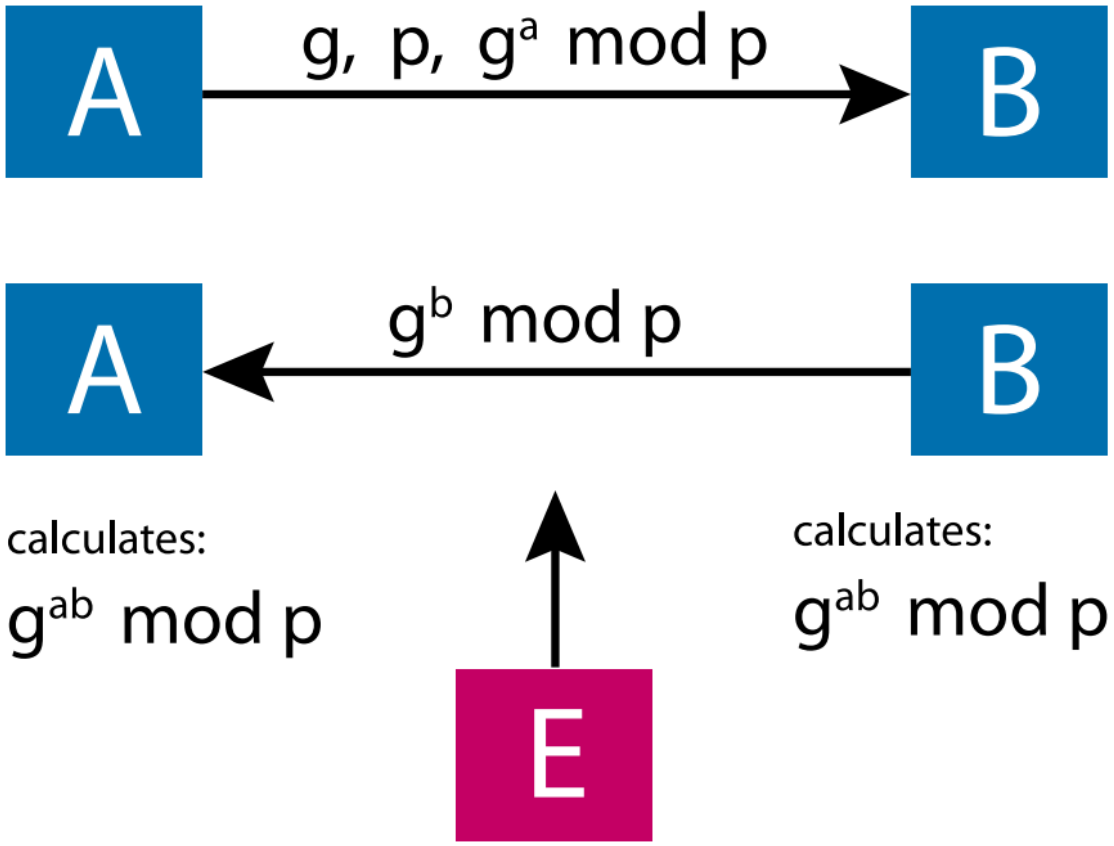
Problem: Key exchange over an insecure channel

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?

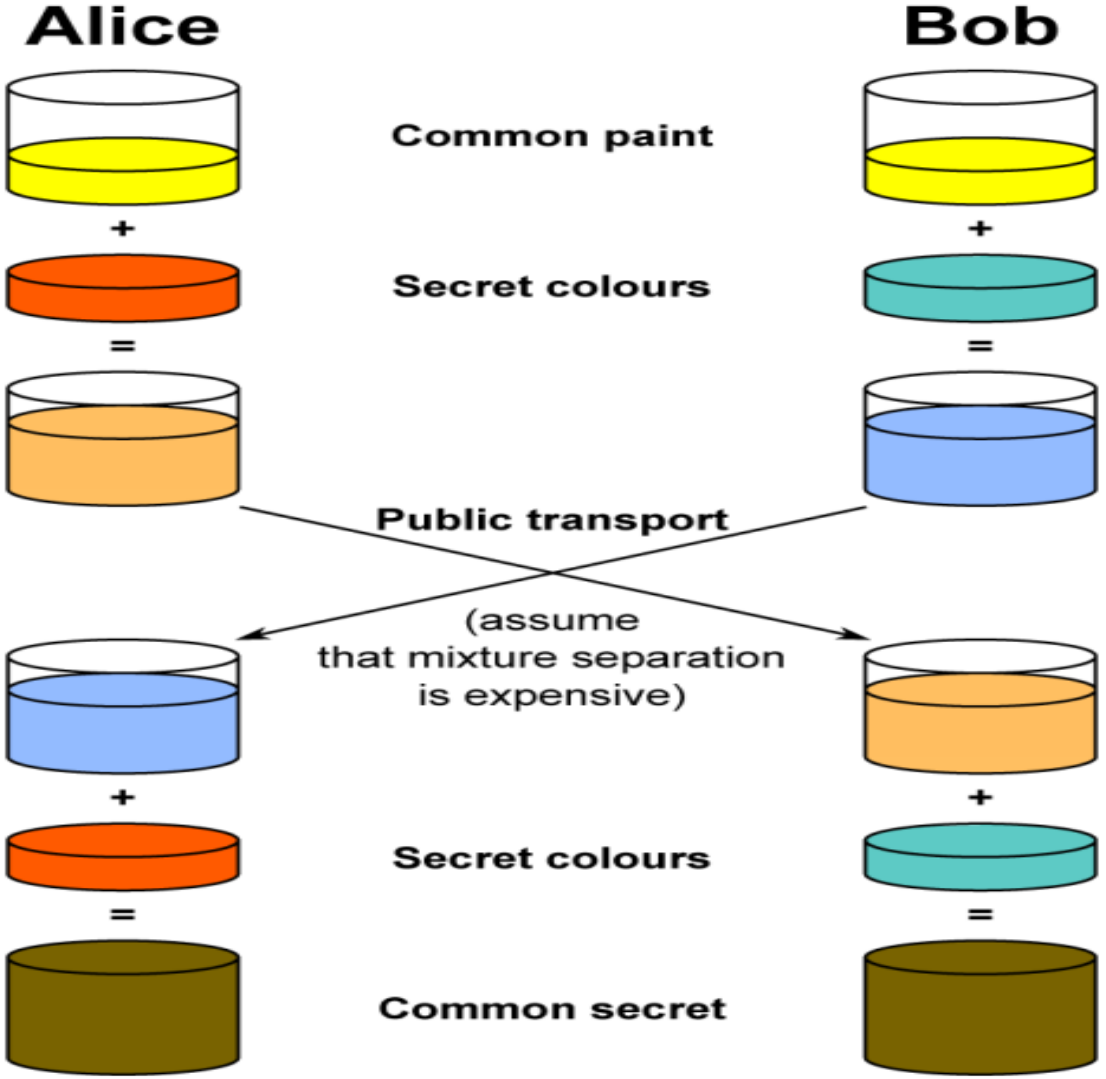
This problem was solved!3 . A solution is Diffie–Hellman key exchange.

1. Alice and Bob agree on a large prime p, and a generator g of $Z_p^{\times}$.
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}.

Diffie–Hellman Key Exchange



Diffie–Hellman Key Exchange: paint analogy



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 mod 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 $Z_p^{\times}$.

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. 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 $Z_p^{\times}$ 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**(`RFC 3526`) **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.

The Strength of Diffie-Hellman

The strength of Diffie-Hellman is based upon two issues:

- Given p, g, g^a - It is difficult to calculate a
  (*the discrete logarithm problem*)
- Given p, g, g^a, g^b - it is difficult to calculate g^{ab}
  (*the **Diffie-Hellman problem***)
- We know that DL → DH but it is not known that DH → DL.

The Strength of Diffie-Hellman

Essentially 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 a key!**

Protip: It is better to either hash it or use it as a seed for a PRNG - not all bits of the secret have a flat distribution!

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.

RSA: Operation

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

Bob computes $m^e \pmod n$ and sends the result to Alice.

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 $k \in Z$ 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 $Z_n^\times$

Computing over $Z_n^\times$, where n = pq for primes p and q.

Things that are **easy**:

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

Things that are **hard**:

- Finding the prime factors of n
- Computing the e^{th} root (as hard as factoring n)
- Solving polynomial equation of degree d

Core security of RSA

The RSA problem

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

(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

$p_1, p_2, \ldots P_n$

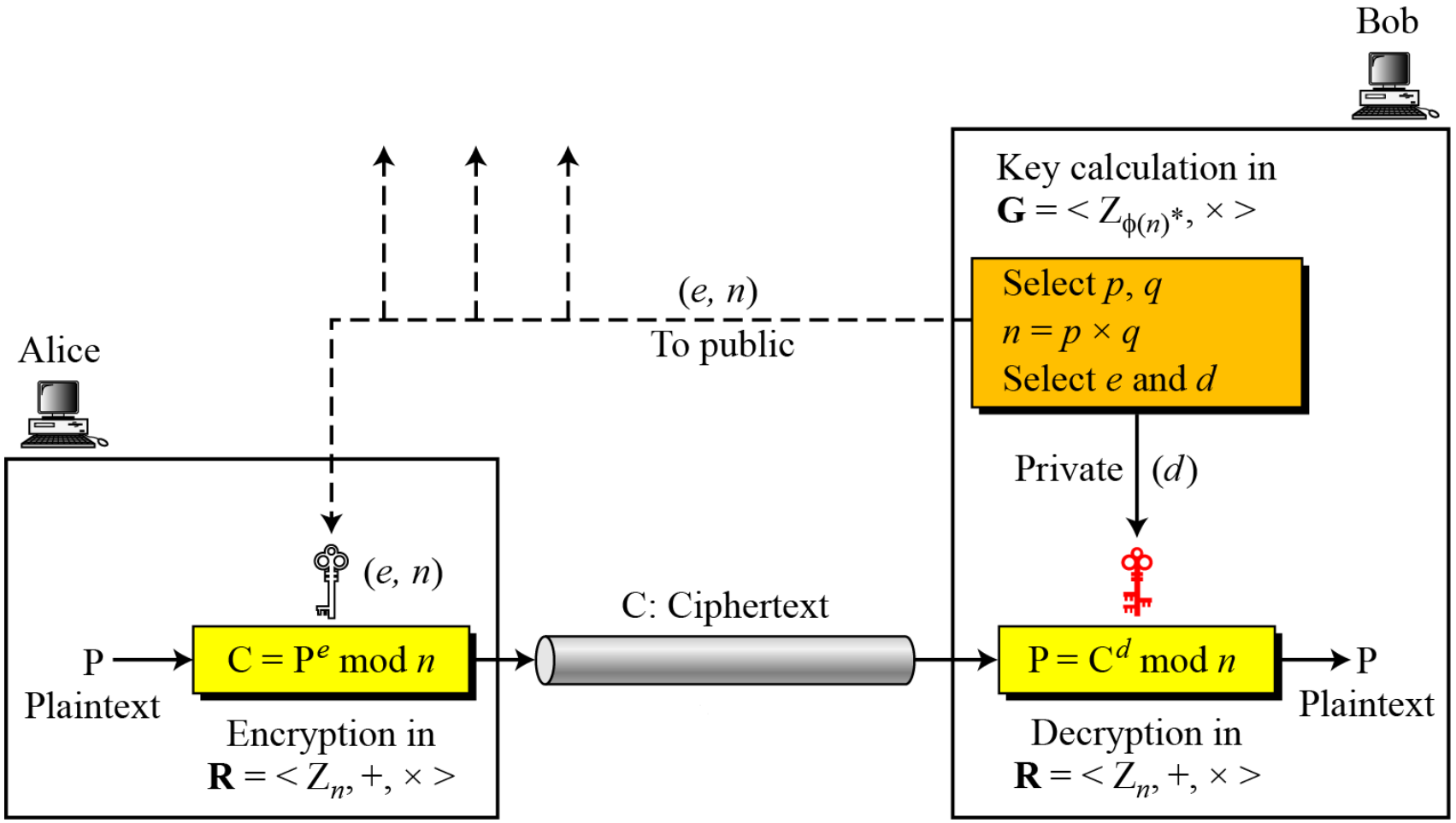
No polynomial time algorithm is yet known.

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.

RSA Process


Bob
Key calculation in
$\mathbf{G} = < Z_{\phi(n)^*}, \times >$
Select p, q
$n = p \times q$
Select e and d
(e, n)
To public
Alice
Private (d)
(e, n)
C: Ciphertext
P
Plaintext
$C = P^e \bmod n$
$P = C^d \bmod n$
P
Plaintext
Encryption in
$\mathbf{R} = < Z_n, +, \times >$
Decryption in
$\mathbf{R} = < Z_n, +, \times >$

Factoring Attack on RSA

The RSA problem: recover m from m^e mod 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

Size of Modulus in RSA

- 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.

Summary: Symmetric Crypto

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.
- 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 $\binom{n}{2}$

Summary: Asymmetric Crypto

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.

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 ≈ 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

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 × 1016 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.