1495 UNIVERSITY OF ABERDEEN

CELEBRATING 525 YEARS 1495 – 2020

ABERDEEN 2040

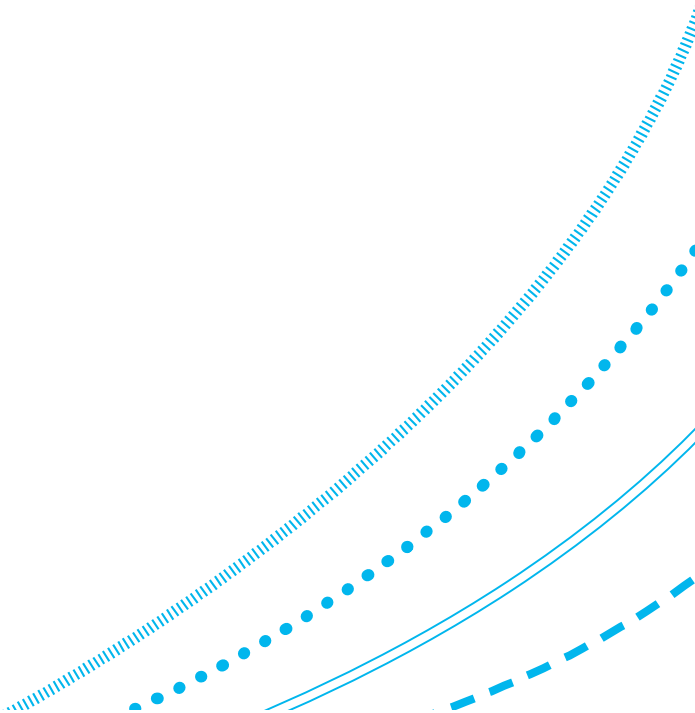
Network Security Technology

Asymmetric and public-key cryptography

September 2025

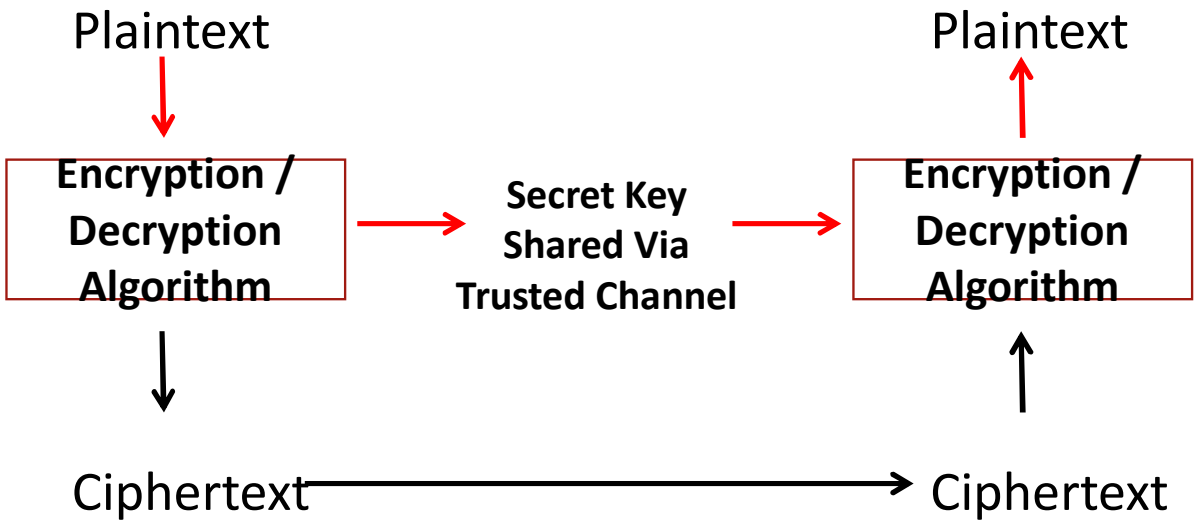
Outline of lecture

1. Introduction.
2. Mathematical and basic ideas.
3. Diffie-Hellman.
4. Elgamal.
5. RSA (Rivest–Shamir–Adleman).



Review: Symmetric-key encryption

1. Same key is used by sender and receiver, has to be shared via some trusted channel.
2. What do you do when there is no secure (trusted) channel?



Key distribution

1. With any symmetric algorithm, the key must be agreed upon by sender and receiver in a secure way
2. Before 1976, key exchange was one of the biggest problems in secure communications.
3. Various people and groups arrived at ``asymmetric'' algorithm solutions around the same period. Some public, some not.

Key distribution (cont'd)

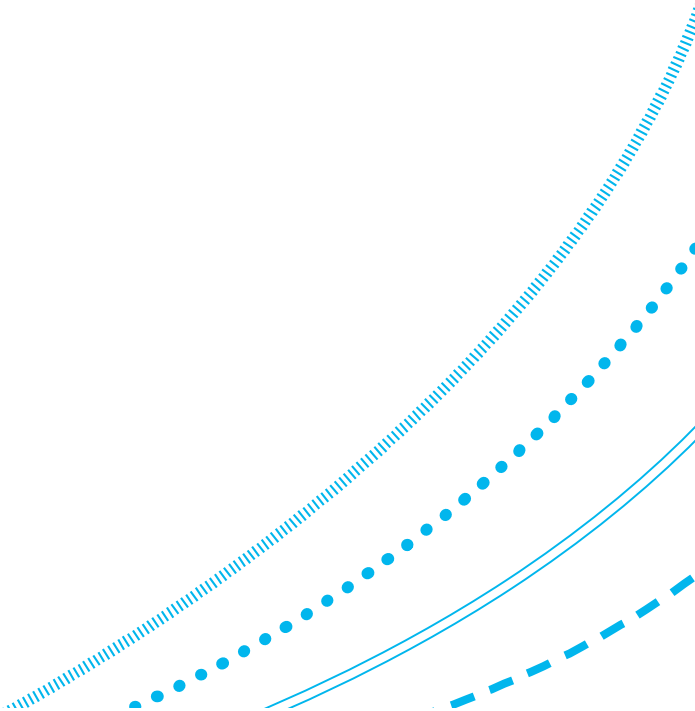
1. Possible Strategies:
    1. A key could be selected by A and physically delivered to B.
    2. A third party could select the key and physically deliver it to A and B.
    3. If A and B have previously used a key, one party could transmit the new key by encrypting it with the old key.
    4. If both A and B have an encrypted connection with a third party C, C could deliver a key on the encrypted links to A and B.

Key establishment / agreement

1. Modern internet imposes new requirements.
2. Need something that scales up to deal with huge numbers of communicating peers.
3. Pairs of peers may be new to each other.
4. May need to minimize dependence on trusted-third parties to deliver keys to peers.
    1. Some intelligence agencies would like to roll this back a bit.

Asymmetric cryptography

1. In symmetric cryptography, pairs of communicating peers share the same key to secure the communication (in some way).
2. In asymmetric cryptography, they use different-but-related keys.



Key establishment / agreement

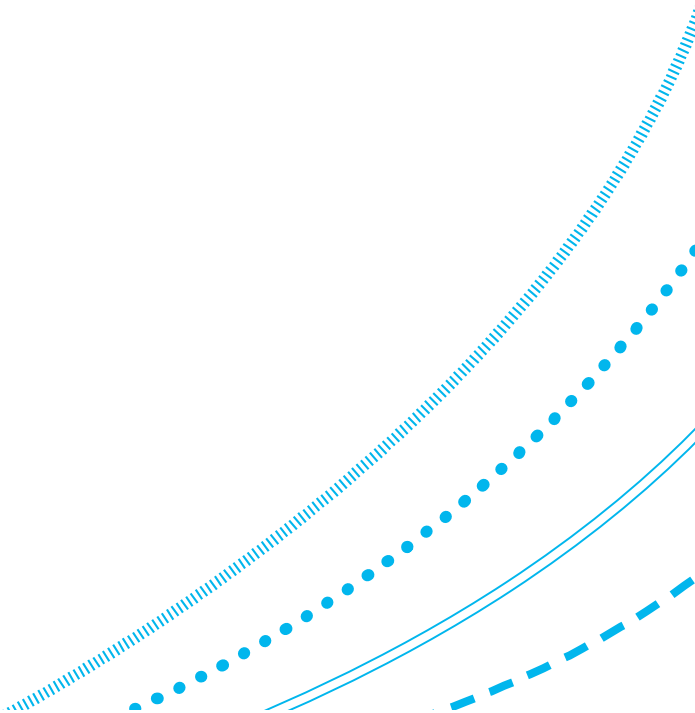
1. Asymmetric cryptography is usually too slow for enciphering and deciphering large messages.
2. We do not usually use it to encrypt the bulk of messages.
3. Mainly used to establish or transport symmetric **session keys** that are used to encrypt the subsequent session.

Roles of modern cryptography

1. **Communication**: encrypt / decrypt with key:
    1. **Symmetric**: different communicating parties use the same secret key for both encryption and decryption.
    2. **Asymmetric**: different parties use different private keys/secrets.
2. **Authentication**, **Data Integrity**: creation of a “fingerprint” or “message digest” for a digital object (e.g. a message or files) and that is hard to fake and that identifies the creator. For example, digital signatures, certificates.

Public-private key pairs

1. Many asymmetric schemes involve public-private key pairs.
2. Let the private key be s (for secret).
3. Let the public key be v (for visible).



Asymmetric encryption

1. Encrypt, *E*, with public key and decrypt, *D*, with private key.
2. E.g., sending a secret, symmetric session key as the message.
3. Anyone with the public key can send a confidential message to the owner of the private key.
4. Need $D(s, E(v, m)) = m$, for all inputs m .

Asymmetric signature

1. Encrypt. with the private key., decrypt with public key.
2. Anyone with the public key can verify that the message was signed by someone who has the private key.
3. E.g., RSA signatures.
4. Need $D(v, E(s, m)) = m$, for all inputs m.

Trapdoor

1. A **trapdoor function** is a function for which inverses can be efficiently computed, but where this computation relies on a secret.
2. In cryptography, the secret relates to (deriving) a private key.
3. Some trapdoors are based on `simple' number theoretic problems.
    1. Discrete knapsack, discrete logarithm problem, prime factorization, …
4. Some based on generalization of these:
    1. E.g. to Group-theoretic problems, elliptic curves, shortest-vector problems in lattice-based cryptography, ….

Absolute value (Abs)

1. For any (positive or negative) integer, X, we can consider its **absolute value**, |X|,
    1. |X|= X if X is non-negative, or
    2. |X| = - X if X is negative.

Integer division and Abs

1. Fix a positive integer N > 0. For any non-negative integer, X, there is a unique pair of non-negative integers (q, r) with r < N such that

$$X = q\,N + r$$

The integer q is the **quotient** and r is the **remainder**.

A quotient is a quantity produced by the division of two numbers.

Prime numbers

1. A **prime** number is a positive integer which only divides (exactly) by itself and 1.
    1. E.g., 2,3,5,7 are primes.
    2. E.g.,. 4,6,8,9 are not prime.
2. **Prime factorization**: Every positive integer can be written *uniquely* as a product of primes.
    1. E.g., 4 = 2 x 2,
    2. E.g., 12 =2 x 2 x 3.

Prime factorisation as a Trapdoor

1. Easy to multiply primes (this is the trapdoor):
    1. Given primes p, q, it is easy to find N = p q.
2. Usually hard to decompose numbers into their prime factors:
    1. Given N, it is hard to compute primes p, q, such that N = p q.

Greatest common divisor (GCD)

- The **greatest common divisor** of two numbers *n* and *m* is the largest integer that divides them both:
  - This number is written **gcd**(*n*, *m*)
  - E.g., gcd(12, 8) = 4,
  - E.g., gcd(9, 12) = 3.

Relatively prime numbers

1. Two integers *n* and *m* are **relatively prime**, if their greatest common divisor is 1:
    1. **gcd**(*n*, *m*) = 1
    2. *n* and *m* do not share any common positive prime factors (divisors) except 1
2. Examples:
    1. 4 and 9 are relatively prime
    2. If *n* is prime, then it is relatively prime to every other *m*.

Modular arithmetic

1. For integers X, Y, we write:

$$X = Y \pmod{N}$$

if N divides the absolute value $|Y - X|$

2. That is, there is some integer q such that

$$|Y - X| = N\,q$$

- Examples:
  - 2 mod 5 = 2
  - 5 mod 4 = 1

Modular arithmetic (cont'd)

1. Equivalently, for non-negative integers X,Y, we write

   $$X = Y \pmod{N}$$

   if the remainders for X and Y upon division by N are equal.

2. In particular, $X = r \pmod{N}$, where r is the remainder of X divided by N.
3. We can **multiply** integers mod p:

   $$(x \bmod p)(y \bmod p) = x\,y \bmod p.$$

Modular arithmetic (cont'd)

1. Fact: if p is prime, then multiplicative inverses exist for all $x > 0$:
2. For any x with $0 < x < p$, there is a unique y with $0 < y < p$ such that

$$x\,y = 1 \bmod p.$$

Z_p

1. $Z_p = \{ 0,1,2,\ldots,p-1 \}$
2. Can consider elements of Z_p as integers modulo p.
3. We can **multiply** elements of Z_p:
    1. So, g, h in Z_p can be multiplied to give another element gh in Z_p.
    2. $(x \bmod p)(y \bmod p) = x\ y \bmod p$
4. Take p in the notation Z_p to be prime always from now on.

Z_p^*

1. Define $Z_p^* = \{1,2,....p-1\}$
2. For any g in Z_p^*, there is a unique h in Z_p^* such that:

   $$gh = 1$$

3. We call h the **inverse** of g and write it as g^{-1}.
4. Z_p^* is a (multiplicative) **group** when p is prime.

Exponentials in Z_p and Z_p^*

1. Easy to calculate exponentials in Z_p .
2. Given $g = a \bmod p$ in Z_p and an integer x, we can calculate:

$$g^x = a^x \bmod p = (a \bmod p) \ldots (a \bmod p)$$

where there are x factors of (a mod p) on the right.

3. Fact: $(g^x)^y = g^{xy} \bmod p$

Discrete logarithm trapdoor

1. Under certain assumptions, it can be guaranteed that for any g, h in Z_p^*, that there is some (smallest) integer x such that:

$$h = g^x .$$

That is, given $0 < g, h < p$ there is a smallest integer x such that:

$$h = g^x \bmod p$$

x is the **discrete logarithm** of h (base g , mod p)

Discrete logarithm trapdoor

1. Recall: *x,* the **discrete logarithm** of *h* (base *g* , mod *p*) is the smallest integer x such that:

$$h = g^x \bmod p$$

2. The **discrete logarithm problem (DLP)** for Z_p^* and a given *g* is to be able to find the discrete log, x, for any given *h*.
3. It is considered `hard' to calculate such discrete logs in the general case:
    1. No efficient algorithm is known.
    2. However, it is not known to be NP-complete.

The Diffie-Hellman scheme

1. Used in:
    1. SSL/TLS (sometimes)
    2. Internet Key Exchange (IKE) underlying IPsec

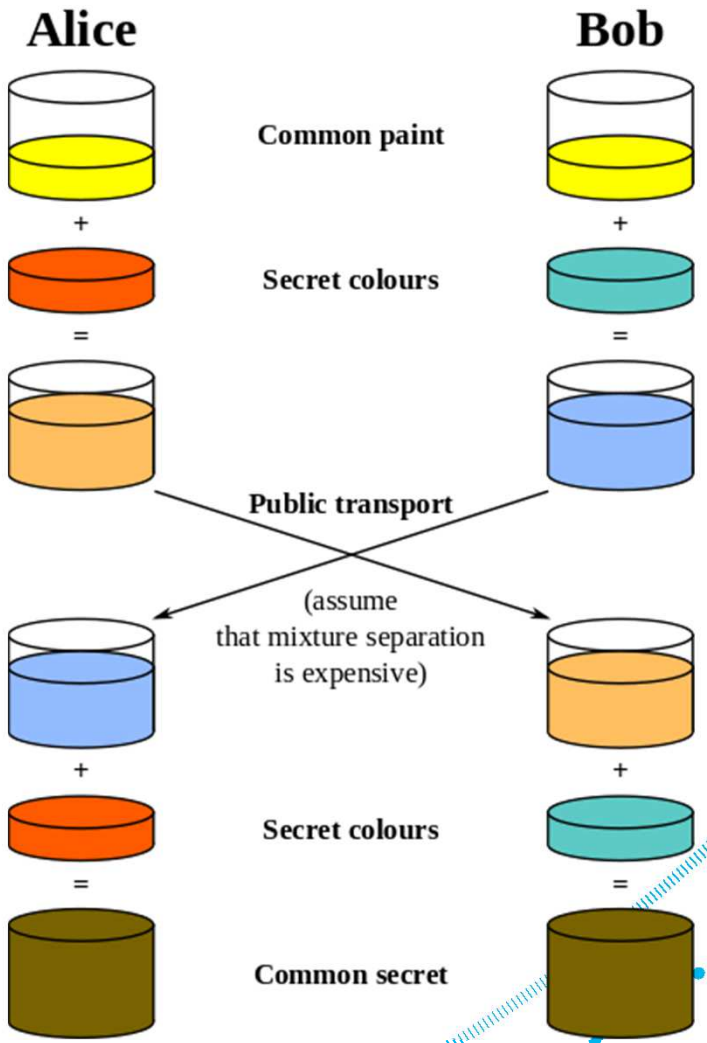
2. Therefore, used in many applications:
    1. E.g. browsers (https) (sometimes), whatsapp, …

The Diffie-Hellman scheme

1. Establishes a shared secret between two parties over an insecure channel.
2. The two participants contribute separately to the creation of the shared secret, but their contributions are only communicated in a scrambled form.
3. Main application is as a **key agreement protocol**: the shared secret is used to generate a symmetric session key.
4. One of the first protocols from asymmetric cryptography:
    1. It is not `encryption' in the conventional sense.
    2. There is no unscrambling to be done.

Idea

1. The crucial part of the process is that Alice and Bob exchange their secret colours in a mix only.
2. Finally, this generates an identical key that is mathematically difficult to reverse for another party that might have been listening in on them.
3. Note that the starting color (yellow) is arbitrary but is agreed on in advance by Alice and Bob.
    1. The starting color is assumed to be known to any eavesdropping opponent. It may even be public.

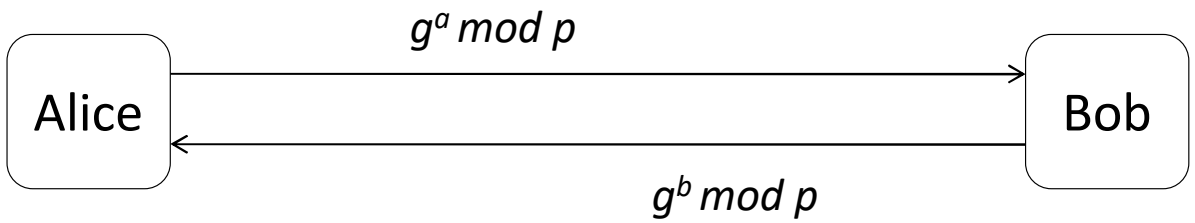


The Diffie-Hellman protocol

- A (Alice) and B (Bob) agree two positive integers, g and p.
  - These are public (not secret). They can be shared openly.
  - These two numbers must satisfy some constraints.

- Alice chooses a secret integer a.
- Alice computes $A = g^a \bmod p$
- Alice sends A to Bob.

- Bob chooses a secret integer b.
- Bob computes $B = g^b \bmod p$.
- Bob sends B to Alice.



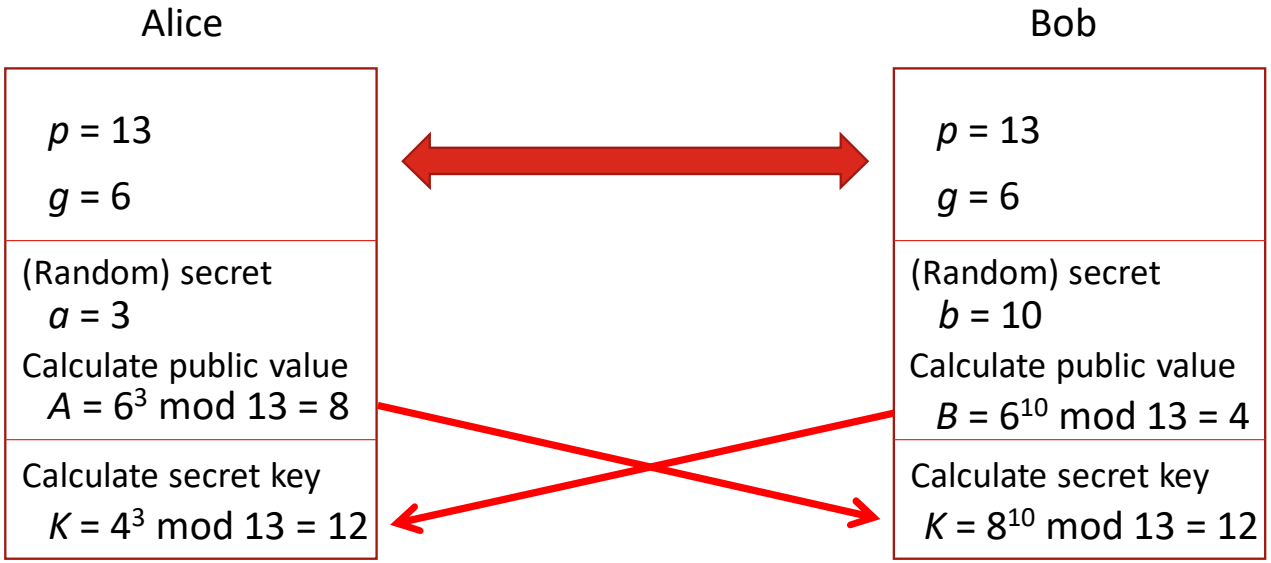
- Alice receives B.
- Alice computes $B^a \bmod p$.

- Bob receives A.
- Bob computes $A^b \bmod p$.

- Alice and Bob now have a shared number:

$$B^a \bmod p = (g^a)^b \bmod p = g^{ab} \bmod p = (g^b)^a \bmod p = A^b \bmod p.$$

Diffie-Hellman key exchange



1. The numbers used here are for example purposes only (they are too small).

Important technicalities for D-H

1. *p* must be a **prime** number (and not too small).
2. *g* must **generate** $Z_p^* = \{ 1,2,\ldots,p-1 \}$: that is, every integer *n* with $0 < n < p$ must be expressible as:

$$n = g^a \bmod p$$

   for some *a*.

3. *a* and *b* are integers:

$$0 < a < p-1, \qquad 0 < b < p-1,$$

   chosen, essentially, uniformly at random.

Secrecy of shared key: why D-H works

1. The attacker's problem: given $g^a \bmod p$ and $g^b \bmod p$, compute $g^{ab} \bmod p$.
2. This is believed to be computationally hard when the constraints on *a, b, g, p* are met. That is, it takes too long even with the best machines and algorithms available.
3. One strategy the attacker might use is to try to compute *a* and *b*. However, given some y, finding an integer *x* that gives:

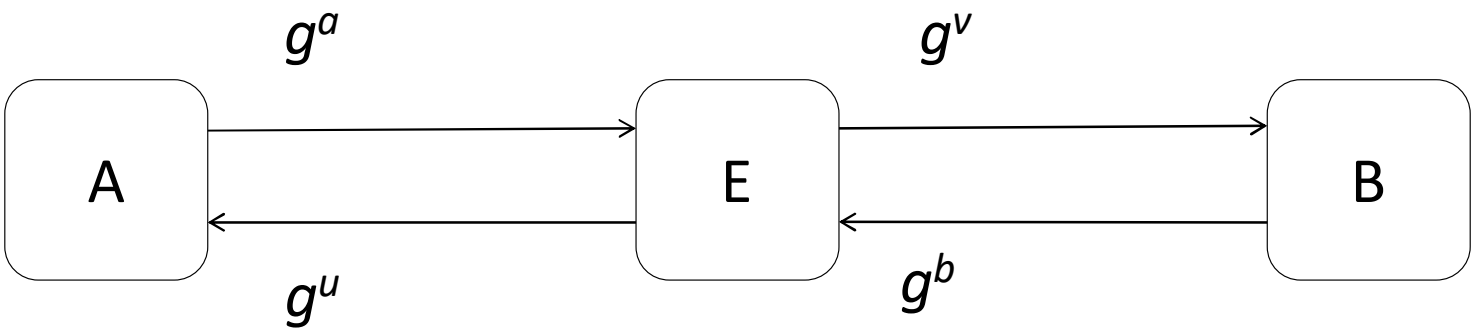
$$y = g^x \bmod p$$

is an instance of the discrete logarithm problem (DLP).

D-H computational problem and the basic protocol

1. Given g and $a = g^x$ and $b = g^y$, compute c such that $c = g^{xy}$.
2. If you can solve the DLP, then you can solve the DHP.
3. Foundation:
    1. Finding the integer x that gives g^x in Z_p^* is an instance of the **discrete logarithm problem (DLP).**
    2. This is believed to be computationally hard (nobody knows an efficient algorithm).
    3. This fact is used in other schemes Elgamal, DSA.

Man-in-the-middle (E)



1. Problem: The basic protocol is unauthenticated. Vulnerable to MiM attack.
2. Various theoretical fixes (station-to-station, MQV, etc.)
3. Most real (and secure) versions have an extra level of authentication.

Perfect forward secrecy

1. ``*An authenticated key exchange protocol provides* ***perfect forward secrecy*** *if disclosure of long-term secret keying material does not compromise the secrecy of the exchanged keys from earlier runs*.''

   Diffie, Whitfield; van Oorschot, Paul C.; Wiener, Michael J. (June 1992). "Authentication and Authenticated Key Exchanges". *Designs, Codes and Cryptography*. 2 pages=107–125 (2): 107.

Types of Diffie-Hellman

1. There are three versions of Diffie-Hellman used in SSL/TLS:
    1. Anonymous D-H.
    2. Fixed D-H.
    3. Ephemeral D-H.
2. **Anonymous Diffie-Hellman** uses Diffie-Hellman, without authentication.
3. **Fixed Diffie-Hellman** embeds the server's public parameter in the certificate, and the Certificate Authority (CA) then signs the certificate:
    1. That is, the certificate contains the Diffie-Hellman public-key parameters, and those parameters never change.

Types of Diffie-Hellman (cont'd)

- **Ephemeral Diffie-Hellman** uses temporary, public keys. Each run of the protocol uses a different public key.
  - The authenticity of the server's temporary key can be verified by checking the signature on the key.
  - The signature is done with the server's long-term key.
  - Can be used to provide perfect forward secrecy.

End-to-end encryption

- “end-to-end encryption” often (but maybe not always) seems to now mean communication between peers without messages being readable by a central server, with keys often established between the peers by ephemeral DH.
- Whatsapp began to offer this around 2014.

Elgamal

1. The ElGamal cryptographic algorithm is an asymmetric key encryption scheme based on the Diffie-Hellman key exchange.
2. The ElGamal algorithm has three main components:
    1. Key generation
    2. Encryption
    3. Decryption

Components of ElGamal

1. Three components:
    1. Key generation
    2. Encryption
    3. Decryption
2. **Key generation**:
    1. **Public Parameters**: Select a large prime number p and a generator g of the multiplicative group Z^*_p.
    2. **Private Key**: Select a private key x such that $1 \leq x \leq p-2$.
    3. **Public Key**: Compute $h=g^x \bmod p$. The public key is (p, g, h) and the private key is x.

Components of ElGamal (cont'd)

3. **Encryption**: Given message M,
    1. Choose a random integer k such that $1 \leq k \leq p-2$.
    2. Compute $C_1 = g^k \bmod p$.
    3. Compute $C_2 = M \cdot h^k \bmod p$.
    4. The ciphertext is (c1,c2).

4. **Decryption**:
    1. Compute the shared secret $s = C_1^x \bmod p$.
    2. Compute $s^{-1} \bmod p$ (the modular inverse of s).
    3. Compute the original message $M = C_2 \cdot s^{-1} \bmod p$.

Elgamal encryption: toy example

- A generates public and private keys:
  - A selects $p = 2357$ and $g = 2$
  - A selects private key $a = 1751$
  - A calculates $g^a \bmod p = 2^{1751} \bmod 2357 = 1185$
  - A has public key $(2357, 2, 1185)$
- B encrypts a message $m = 2035$:
  - B selects a random integer $k = 1520$
  - B computes $G = 2^{1520} \bmod 2357 = 1430$
  - B computes $H = 2035 \times 1185^{1520} \bmod 2537 = 697$
  - B sends $(1430, 697)$ to A
- A decrypts:
  - A computes $Y = G^{p-1-a} = 1430^{2357-1-1751} = 1430^{605} \bmod 2357 = 872$
  - A computes $m = 872 \times 697 \bmod 2357 = 2035$.

Elgamal encryption works

	$D_s (E_v (m))$	
=	$D_S ((G,H))$	*with* $G = g^k$, $H = m(g^a)^k$
=	$G^{p-1-a} H$	
=	$g^{(p-1-a)k} m g^{ak}$	
=	$m g^{p-1}$	
=	m	See below.

1. All of these equations are mod p .
2. Since g generates p , it is a fact (Fermat's Little Theorem) that $1 = g^{p-1}$ in Z_p.

Elgamal and the (C)DHP

1. Adversary has $G = g^k$, and $H = m(g^a)^k$, and g and k, but not a or m. Wants to find m.
2. Attacker has g^k and g^a . If the attacker can solve the CDHP, then it can get g^{ak} (recall that this is effectively the shared key). It can then recover m from H.
3. So, the Elgamal problem is no harder than the CDHP.

The *computational Diffie-Hellman problem* (CDHP) is to compute g^{ak}.

Elgamal and the (C)DHP

1. A fortiori, if adversary can solve the DLP, then it can get *m*.
2. Direct argument for when the DLP can be solved by the attacker.
    1. If adversary can solve the discrete log. problem, then it can find *k* from the equation for *G.* It can then find *a* from the equation for *H.*
    2. Note that $m = H^{-1} G^{a}$.
    3. If adversary can find *a,* then it can find *m*.

Elgamal issues

1. Simple Elgamal encryption has many known weaknesses.
    1. **Slow processing:** ElGamal is slower compared to other encryption algorithms, especially when used with long keys. This can make it impractical for certain applications that require fast processing speeds.
    2. **Key size:** ElGamal requires larger key sizes to achieve the same level of security as other algorithms. This can make it more difficult to use in some applications.
    3. **Vulnerability to certain attacks:** ElGamal is vulnerable to attacks based on the discrete logarithm problem, such as the index calculus algorithm. This can reduce the security of the algorithm in certain situations.
2. You have to do a lot more work to turn it into an effective scheme.

RSA

1. RSA is based on the fact that it is difficult to factorise a large integer.
2. The public key consists of two numbers where one number is a multiplication of two large prime numbers.
3. And private key is also derived from the same two prime numbers.
4. If somebody can factorise the large number, the private key is compromised.
5. Therefore, encryption strength totally lies on the key size.
    1. If we double or triple the key size, the strength of encryption increases exponentially.
    2. RSA keys can be typically 1024 or 2048 bits long.

RSA minimal textbook version

- **Key generation**:
  - Pick two large, secret, prime numbers p, q
  - Compute $n = p\,q$.
  - Choose an encryption exponent e.
    - Must satisfy a certain condition.
  - Compute d from the data e, p, q.
  - **Public key**: pair (e, n)
  - **Private key**: pair (d, n)

- **Encryption**:

  $c = m^e \bmod n$.

- **Decryption**:

  $m = c^d \bmod n$.

RSA: Generating keys example

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 * 11 = 187$.
3. Calculate $\phi(n) = (p - 1)(q - 1) = 16 * 10 = 160$.
4. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$;
    1. we choose $e = 7$.
5. Determine d such that $de \equiv 1 \bmod 160$ and $d < 160$.
    1. The correct value is $d = 23$, because $23 * 7 = 161 = (1 * 160) + 1$;
6. The resulting keys are public key $PU = \{7, 187\}$ and private key $PR = \{23, 187\}$.

Two integers n and m are **relatively prime**, if their greatest common divisor is 1.

RSA: encryption example

1. plaintext input of $M = 88$.
2. $88^7 \bmod 187 = [(88^4 \bmod 187) * (88^2 \bmod 187) * (88^1 \bmod 187)] \bmod 187$
3. $88^1 \bmod 187 = 88$
4. $88^2 \bmod 187 = 7744 \bmod 187 = 77$
5. $88^4 \bmod 187 = 59{,}969{,}536 \bmod 187 = 132$
6. $88^7 \bmod 187 = (88 * 77 * 132) \bmod 187 = 894{,}432 \bmod 187 = 11$

RSA: decryption example

1. We calculate $M = 11^{23} \bmod 187$.
2. $11^{23} \bmod 187 = [(11^1 \bmod 187) * (11^2 \bmod 187) * (11^4 \bmod 187) * (11^8 \bmod 187) * (11^8 \bmod 187)] \bmod 187$
3. $11^1 \bmod 187 = 11$
4. $11^2 \bmod 187 = 121$
5. $11^4 \bmod 187 = 14{,}641 \bmod 187 = 55$
6. $11^8 \bmod 187 = 214{,}358{,}881 \bmod 187 = 33$
7. $11^{23} \bmod 187 = (11 * 121 * 55 * 33 * 33) \bmod 187 = 79{,}720{,}245 \bmod 187 = 88$

Details of RSA key generation

1. Compute:

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

   This is a formula for the **Euler totient** of n when it is of the form $n = p\,q$ for primes p and q.

2. The encryption exponent e must be chosen so that e and ϕ are relatively prime.

3. In practice, e is often simply chosen as a fixed prime. (e.g. openssl flavours 65537 or 3).

Details of RSA key generation (cont'd)

1. Compute d such that the property:

   $$e\,d = 1 \bmod \phi \text{ holds.}$$

2. This can be done efficiently using the data e, p, q via the **extended Euclidean algorithm**.
3. There is no known algorithm to do this efficiently without knowing p and q.

Another example

1. Choose $p = 7$, $q = 11$
2. Then $n = 77$.
3. Choose $e = 37$.
4. Compute $d = 13$ by extended Euclidean Algorithm, using n, p, q.
    1. Result $d = 13$
5. Suppose that the message is $m = 2$.
6. Then $c = 2^{37} \bmod 77 = 51$.
7. Decrypt: $m' = 51^{13} \bmod 77 = 2$.
    1. So, $m' = m$, as required.

RSA basic decryption property proof

	$D_s(E_v(m))$	
=	c^d	
=	$(m^e)^d$	
		Since e d = 1 mod ϕ, by definition there is some k such that e d = 1 + kϕ
=	$m^{1+k\phi}$	
=	$m\ (m^{\phi})^k$	
		(*) Fact: $m^{\phi} = 1$.
=	$m\ 1^k$	
=	m	

1. All equations here are mod *n*

The RSA problem

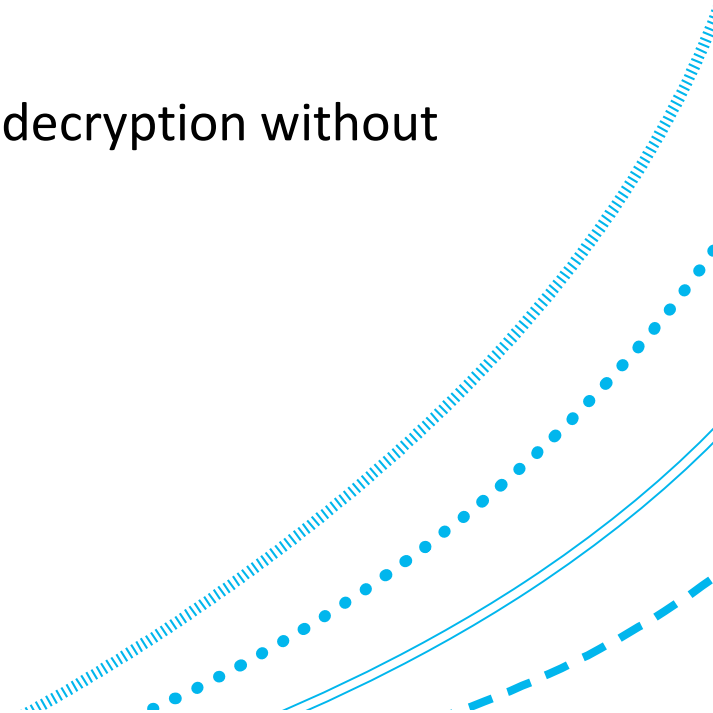
1. The RSA problem: Find m, given only m^e mod n and e and n.
2. The RSA problem is no harder than the problem than the problem of factoring an integer:
    1. In cipher terms, if an attacker has the public key (e, n), and if they can factor n to get the primes p , q, then they can calculate the private key d, and decrypt to get the original message, m.
3. If the RSA problem is hard, then RSA is secure against a chosen plaintext attack.

Factoring

1. There are factorisation algorithms that are smarter than big dumb search (brute force), e.g., the Number Field Sieve.
2. An N-bit RSA key does not provide 2^N bits of security. For example, a 2048-bit key only provides 112 bits of security.
3. Numbers over 1000-bits have been factored:
    1. 768-bit RSA no longer secure.
    2. 1024-bit RSA is starting to look insecure.
    3. At least 2048-bit RSA appears necessary.
4. Quantum algorithms (Shor and Grover) are orders of magnitude more efficient, and so make the problem much worse.

Limitations of textbook RSA

1. Entire books have been written on this topic
2. Have multiple users share the same modulus, n, even if they use different exponents (e_i, d_i). Allows for attacks by insiders, and also outsiders, without factoring.
3. Use of too small a public exponent (again, can allow for decryption without factoring).
4. Not secure against chosen-plaintext attack.
5. Not secure against certain chosen-ciphertext attacks.
6. … Lots more (e.g. timing attacks) …



PKCS & Elliptic curve cryptography

1. PKCS:
    1. A family of standards for RSA published by the company RSA Security.
    2. There is an ISO/IEC standard.
2. Elliptic curve cryptography:
    1. RSA needs many more bits to encrypt the session key (>10 times the number of bits in the secret (e.g. session key) being sent).
    2. For low-power devices (smart cards etc.) even the operations (e.g. multiplication) can be too computationally intensive.
    3. Elliptic curves with Elgamal-like schemes give an alternative.
    4. Fewer bits required to encrypt the secret.
    5. Less computationally intensive.