1 Diffie-Hellman key agreement

1.1 Finding a generator

a. (1) the element tested from 1 to 10; (2) the order of each such element; and (3) what (sub)group that element generates.

Element	Order mod 2027	Generated Group
1	1	Trivial subgroup
2	2026	Full group Z*_2027
3	1013	Large prime-order subgroup
4	1013	Large prime-order subgroup
5	2026	Full group Z*_2027
6	2026	Full group Z*_2027
7	2026	Full group Z*_2027
8	2026	Full group Z*_2027
9	1013	Large prime-order subgroup
10	1013	Large prime-order subgroup
6	2026	Full group Z*_2027
7	2026	Full group Z*_2027
8	2026	Full group Z*_2027
9	1013	Large prime-order subgroup
10	1013	Large prime-order subgroup

b.

- *full group g=2 → order 2026*

  2^{2026} mod 2027 ≡ 1, for all proper divisors d|2026, $2^{d} \neq 1$ mod 2027,

  2^{1} mod 2027 ≠ 1, 2^{2} mod 2027 ≠ 1, 2^{1013} mod 2027 ≠ 1, 2^{2026} mod 2027 = 1

  Conclusion: g=2 passes all checks

- *large prime-order subgroup g=3 → order 1013*

  3^{1013} mod 2027 ≡ 1, 3^{1} mod 2027 ≠ 1

  $g^{q} \equiv 1$ mod p, and $g \neq 1$ mod p

  Conclusion: g=3

1.2 Performing key exchange

a.

$a = 8wy = 8\times62 = 496$, $b = 9xz = 9\times28=252$, $p = 2027$, $g = 2$

b.

$A = g^a \bmod p = 2^{496} \bmod 2027 = 1450$, $B = g^b \bmod p = 2^{252} \bmod 2027 = 872$

c.

$s = B^a \bmod p = 872^{496} \bmod 2027 = 902$, $s = A^b \bmod p = 1450^{252} \bmod 2027 = 902$

They are agreeing on session key: 902

1.3 Sabotaging the protocol

a.

$p = 2027$, $g = 2$, Alice's private key: $a = 496$, Malicious Bot's private key: $b^1 = 1013$,

Both derive the same session key: $s = A^{b1} \bmod p = 1450^{1013}$, $\bmod 2027 = 1$

b.

Alpha (α)	Alice's Message A	Shared Secret with Bob	Shared Secret with Bot
496	1450	902	1
497	873	68	2026
498	1746	513	1
499	1465	1396	2026
500	903	1112	1
501	1806	758	2026
502	1585	174	1
503	1143	1730	2026
504	259	472	1
505	518	103	2026

c.

The session keys are highly predictable and guessable, because one party uses a fixed exponent corresponding to a low-order subgroup. This breaks the security of Diffie-Hellman, as an attacker could easily guess or brute force the key.

2 The RSA cryptosystem

2.1 Textbook RSA encryption

a.

e = 28 x 6 + 401 = 168 + 401 = 569

b.

p = 2027, q = 2593, n = 2027 x 2593 = 5256011, $\phi(n) = (2026) \times (2592) = 5251392$,

e = 569, my public key is: (n = 5256011, e = 569)

c.

$d = e^{-1} \bmod \phi(n) = 569^{-1} \bmod 5251392 = 230729$,

my full RSA private key is: n = 5256011, d = 230729

d.

q	r1	r2	s1	s2	t1	t2
9229	569	91	0	1	1	-9229
6	91	23	1	-6	-9229	55375
3	23	22	-6	19	55375	-175354
1	22	1	19	-25	-175354	230729
22	1	0	-25	569	230729	-5251392

gcd(5251392, 569) = 1,
Modular inverse $d = 569^{-1} \bmod 5251392 = 230729$,
$569 \cdot 230729 - 5251392 \cdot 25 = 1$, d = 230729

e.

Step	Bit	Square	Multiply
1	1	1	1024
2	0	1048576	—
3	0	1430675	—
4	0	3615939	—
5	1	4207791	4104975
6	1	1180526	5232105
7	1	3847648	3239313
8	0	1047470	—
9	0	1104650	—
10	1	340707	1987242

Public key (n = 5256011, e = 569), Plaintext m = 1024,
$c = 1024^{569} \bmod 5256011 = 1987242$

f.

Ciphertext: c = 1987242, Private exponent: d = 230729,
Primes: p = 2027, q = 2593, Modulus: n = pq = 5256011,

d^p = d mod (p - 1) = 230729 mod 2026 = 1791,
d^p = d mod (q - 1) = 230729 mod 2592 = 41

$M_p = c^{1791}$ mod 2027 = 1024,
$M_q = c^{41}$ mod 2593 = 1024

q^{-1} mod p = 727,
p^{-1} mod q = 1663

M = (q · q $^{-1}$ mod p · Mp + p · p^{-1} mod q · Mq) mod n
M = (2593 · 727 · 1024 + 2027 · 1663 · 1024) mod 5256011 + 1024

Decryption result: 1024

2.2 Distinguishing attacks against Textbook RSA and Random-Padded RSA

a.

Public key: (n = 9005063, e = 17), C* = 3155223, M1 = 111, M2 = 222

1. Encrypt both M1 and M2 using the public key:
   - C1 = 111^{17} mod 9005063 = 6921938
   - C2 = 222^{17} mod 9005063 = 3155223
2. Compare:
   - C* = C2

The plaintext is M2 = 222, encrypting the same message always gets the same ciphertext.

Property exploited: Deterministic encryption is lacking semantic security.

b.

We could try: C1 = $M1^e$ mod n, C2 = $M2^e$ mod n,

and compare directly to C∗, because the scheme is deterministic.

But with random padding, every message is encoded as: Z = 1000 · R + M

and since R is chosen randomly for every encryption,

the value of Z changes every time, even for the same M.

So, we cannot predict any specific ciphertext to a known message like before too many possible Z values **e**xist**:**

- For each M, there are 9004 possible Z values
- And thus 9004 different ciphertexts for each M

With random-padded RSA, we cannot efficiently distinguish whether the message was 111 or 222 from the ciphertext alone, because the encryption introduces randomness that masks deterministic patterns.

3 Digital signatures and authentication

a.

1. The server chooses a crafted challenge x = h (M*) x = h(M*), where M* M* is the message the server wants to forge as if it came from Alice.
2. The server sends this x to Alice in the login protocol.
3. Alice, unaware of the server's intent, signs x and returns:
   $s = \text{Sign}_{sk}(x) = \text{Sign}_{sk}(h(M^*))$
4. The server now possesses a valid signature s on h(M*), which is precisely what an email signature would look like for message M*.
   The server succeeded forged Alice's signature on an arbitrary email message M*.

b.

To ensure that what Alice signs during login cannot be reused in another context:

1. Modify the challenge x sent by the server so that Alice signs only authentication specific data, e.g.: x^1 = Hash("LOGIN" || x || ServerID || Timestamp)
2. Slice signs $s = \text{Sign}_{sk}(x^1)$
3. The login protocol now uses structured data that:
   a. Includes metadata like "LOGIN"
   b. Is tied to the session via timestamp and server identity
   c. Cannot be reused as a hash of any valid email message

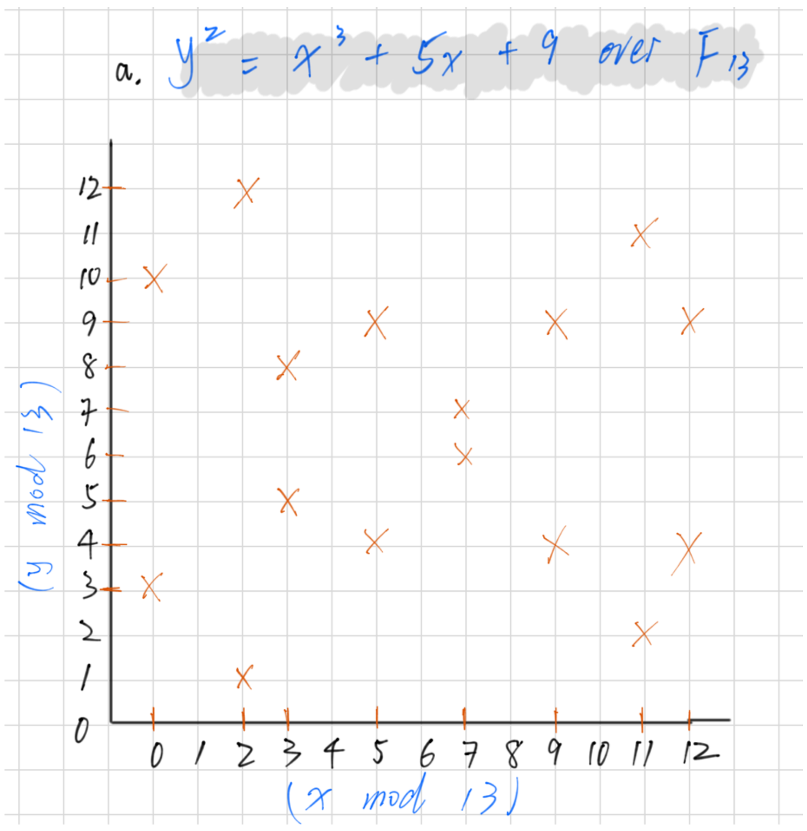
c.

Never reuse the same cryptographic key across different protocols or application. A signature in one protocol might be exploited to forge signatures in another like universal forgery attack. Instead, one should use derive distinct keys for each purpose such as login or email form a master key using a secure KDF, making sure both safety and efficient key management.

4 Elliptic-curve cryptography

4.1 Elliptic-curve arithmetic

a.



b.

16 affine points on the curve, and 1 point at infinity (the identity element), so the order of my curve is: 17

c.

G = (7,7) & H = (12, 9): on the curve

d.

$\lambda = \frac{3x12+a^1}{2y1}$ mod p,

$x_3 = \lambda^2 - 2_{x1}$ mod p,

$y_3 = \lambda(x_1 - x_3) - y_1$ mod p

the result is: 2G = G + G = 2, 12

e.

I draw a line connecting the points G = (7,7) and H = (12,9) on the plot of the curve over F13. This line intersects the curve at a third point R. Then, I reflect R across the x-axis (mod 13), which gives the result G+H. Since the plot uses a finite field, this reflection is equivalent to negating the y-coordinate modulo 13.

f.

$\lambda = \frac{y2-y1}{x2-x1}$ mod 13 = $\frac{9-7}{12-7} = \frac{2}{5}$ mod 13 = $2 \cdot 5^{-1}$ mod 13

5^{-1} mod 13 = 8, since $5 \cdot 8 = 40 \equiv 1$ mod 13

$\lambda = 2 \cdot 8 = 16 \equiv 3$ mod 13

$x_3 = \lambda^2 - x_1 - x_2$ mod 13 = $3^2 - 7 - 12 = 9 - 19 = -10$ mod 13 = 3

$y_3 = \lambda(x_1 - x_3) - y_1 = 3(7 - 3) - 7 = 3 \cdot 4 - 7 = 12 - 7 = 5$

Result is: G + H = 3, 5

4.2 Elliptic-curve Diffie-Hellman key agreement

a.

The order of point G = (7, 7) under the elliptic curve $y^2 = x^3 + 5x + 9$ mod 13 found: Order of G = 17, G generates the entire group of the curve, since the curve itself has order 17.

b.

Alice's private key: a = 5, generate point: G = (7, 7)

The point: $A = a \cdot G = 5 \cdot (7, 7) = (3, 8)$

c.

Bob's private key: b = 9, generate point: G = (7, 7)

The public point: $B = b \cdot G = 9 \cdot (7, 7) = (9, 9)$

d.

From Alice: $S = a \cdot B = 5 \cdot (9, 9) = (12, 9)$

From Bob: $S = b \cdot A = 9 \cdot (3, 8) = (12, 9)$

Both Alice and Bob independently derive the same shared secret point (12, 9), it means that the ECDH protocol works correctly with the curve and keys.