VaultForge — Flashcards

A Spark & Anvil printable flashcard deck. Quiz yourself on the VaultForge cast's curriculum — one card at a time.

How to use these cards

Fold each sheet down the middle line so the answers are hidden. Read the **question** on the left, say your answer out loud, then **unfold** to check the right side. Testing yourself — then checking — is one of the most powerful ways to remember. Get one wrong? That's the best kind of practice: try it again tomorrow.

Want real flashcards? Cut along the fold line and the rows into cards — question on one side, answer on the other.

---

1. Encode vs encrypt; confidentiality

✂ Question (fold →)	Answer
The FIRST question a defender asks about a scrambled message is:	**what technique produced this, so I know how it could be read or broken** — Identifying the technique guides both securing and analyzing it.
Encryption is reversible with the key; a HASH is:	**not reversible (one-way)** — Encryption is designed to be reversed with the key; a hash is deliberately one-way.
"Plaintext" is:	**the original readable message before encryption** — Plaintext is the readable input; ciphertext is the scrambled output.
The reason we study historical ciphers (Caesar, Vigenère) is to:	**understand the concepts of keys, patterns, and weaknesses — not to secure real data today** — Classical ciphers teach the ideas; they are far too weak for real modern security.
Encoding is used for:	**representing data in a transmittable format (no secrecy)** — Encoding formats data for transport/storage; it provides no confidentiality.
The key difference between ENCODING and ENCRYPTING is that encryption:	**requires a secret key to reverse, while encoding is public and reversible by anyone** — Encoding (like ASCII or Base64) is a public format anyone can reverse; encryption needs a secret key.
The single biggest idea of this kit is:	**encryption keeps secrets with a key; encoding just changes format** — Encryption (keyed secrecy) is fundamentally different from encoding (public format).
A message that is encrypted but sent with the key attached is:	**insecure — the key must be protected separately** — Shipping the key with the ciphertext defeats the purpose; keys must stay secret.
The main goal of ENCRYPTION is to:	**keep information confidential from anyone without the key** — Encryption protects confidentiality: only those with the key can read the message.
Confidentiality, integrity, and availability together are known as:	**the CIA triad** — The CIA triad (Confidentiality, Integrity, Availability) frames security goals.
Base64 is an example of:	**encoding, not encryption** — Base64 is a public, keyless, fully reversible encoding — it provides no confidentiality.
The study of making and breaking codes is called:	**cryptology (cryptography + cryptanalysis)** — Cryptology spans cryptography (making) and cryptanalysis (breaking).
A defender who understands encryption can:	**choose appropriate protections for the sensitivity of the data** — Understanding encryption lets defenders match protection to data sensitivity.
Confidentiality means:	**only authorized parties can read the information** — Confidentiality restricts who can READ the data.
"Ciphertext" is:	**the scrambled output after encryption** — Ciphertext is the encrypted, unreadable-without-the-key result.
If a challenge says "decode this Base64," the right tool is:	**a public Base64 decoder — no key needed** — Base64 is keyless encoding; you simply decode it, no cryptanalysis required.

If two people want to exchange secret messages, they minimally need:	**an agreed cipher and a way to share/derive the key securely** — Secure messaging needs both the algorithm and secure key handling.
A one-time-use secret shared safely once can enable:	**a very strong symmetric cipher (in principle)** — A securely shared secret key underlies strong symmetric encryption.
The white-hat principle for studying attacks is:	**learn how weaknesses work to DEFEND systems, never to harm real ones** — Ethical (white-hat) practice studies weaknesses to defend, with permission — never to cause harm.
A "cipher" is:	**an algorithm for transforming plaintext into ciphertext and back** — A cipher is the transformation method; the key parameterizes it.
A system where the SAME key encrypts and decrypts is called:	**symmetric** — Symmetric encryption uses one shared secret key for both directions.
Which best shows the difference between secrecy and integrity?	**encryption hides the message; a hash detects if it changed** — Encryption = confidentiality (hide); hashing = integrity (detect change).
Encryption provides confidentiality but does NOT by itself guarantee:	**that the message wasn't altered (integrity) or who sent it (authenticity)** — Confidentiality is separate from integrity and authenticity — other tools (hashes, signatures) handle those.
"Kerckhoffs's principle" (previewed) says a system should be secure even if:	**the algorithm is public — only the key must be secret** — Security should rest on the secret KEY, not on hiding the algorithm.
If an attacker sees only ciphertext and cannot read it without the key, encryption has:	**succeeded at its confidentiality goal** — If ciphertext is unreadable without the key, confidentiality is achieved.

2. Shift cipher; the key

✂ Question (fold →)	Answer
Encrypting the letter M with shift 5 gives:	**R** — M+5 = R (M,N,O,P,Q,R).
Encrypting the letter Z with a Caesar shift of 1 gives:	**A** — Z+1 wraps past the end back to A.
The Caesar cipher teaches the core idea that:	**a cipher = a transformation controlled by a key** — It introduces the key-controlled transformation at the heart of all ciphers.
With a shift of 3, the letter A becomes:	**D** — A shifted forward by 3 is D (A→B→C→D).
With shift 3, the letter Y becomes:	**B** — Y+3 wraps past Z: Y→Z→A→B = B.
The most common letter in a long Caesar-enciphered English text likely maps back to:	**E** — E is the most common English letter, so the most common ciphertext letter often decrypts to E.
The 'key space' of the Caesar cipher is:	**tiny (25 useful keys)** — A tiny key space (25) is why it is insecure.
To make a shift cipher stronger, you might:	**use a DIFFERENT shift for each position (polyalphabetic, like Vigenère)** — Varying the shift per position (Vigenère) resists simple frequency analysis.
With a shift of 1, the word CAB becomes:	**DBC** — Each letter +1: C→D, A→B, B→C = DBC.
The Caesar cipher is a type of:	**monoalphabetic substitution cipher** — Every letter maps to one fixed substitute — a monoalphabetic substitution.
A Caesar cipher encrypts by:	**shifting every letter forward by a fixed number of positions** — The Caesar cipher shifts each letter by a constant amount (the key).
A Caesar cipher provides confidentiality only against:	**someone who doesn't know it is a simple shift** — It only stops the most casual reader; any analyst breaks it instantly.
Frequency analysis works on a Caesar cipher because:	**letter frequencies are just shifted, so the pattern survives** — A single shift preserves the frequency pattern, so the common letter reveals the shift.
If you intercept 'KHOOR' and suspect a Caesar cipher, a shift of 3 back gives:	**HELLO** — K,H,O,O,R each −3 → H,E,L,L,O = HELLO.
If shift is unknown, the fastest manual attack on a short message is:	**try all 25 shifts and read for sense** — With only 25 keys, exhaustive trial is fast and reliable.
ROT13 is a Caesar cipher with shift:	**13** — ROT13 shifts by 13; applying it twice returns the original.
To DECRYPT a Caesar cipher with shift 3, you:	**shift each letter BACKWARD by 3** — Decryption reverses encryption: shift back by the same key.
Applying ROT13 twice to a message gives:	**the original message back** — 13 + 13 = 26 = a full wrap, so ROT13 is its own inverse.
"Brute force" against a Caesar cipher	**trying all possible shifts until the text reads sensibly** — You test each

means:	of the 25 shifts and pick the readable result.
In a Caesar cipher, the KEY is:	**the number of positions to shift** — The key is the shift amount (0–25).
When a shift goes past Z, it:	**wraps around to the start of the alphabet** — The alphabet wraps: after Z you continue at A (modular arithmetic).
A shift of 0 (or 26) produces ciphertext that is:	**identical to the plaintext** — Shifting by 0 (or 26) changes nothing.
With shift 3, decrypting D gives:	**A** — D shifted back 3 is A (D→C→B→A).
If the most common ciphertext letter is H and it should be E, the shift is:	**3** — E→H is +3, so the shift is 3.
The Caesar cipher is weak because:	**there are only 25 possible non-zero keys to try** — Only 25 shifts exist, so brute-forcing every key is trivial.

3. Monoalphabetic substitution; frequency analysis

✂ Question (fold →)	Answer
If ciphertext letter 'Q' is by far the most common, a good first guess is Q =	**E** — The most common ciphertext letter most likely maps to E.
A defender learns frequency analysis mainly to:	**understand why classical ciphers fail and choose stronger ones** — Knowing the attack shows why weak ciphers fail and motivates strong ones.
To resist frequency analysis, a cipher should:	**flatten the letter frequencies (e.g. polyalphabetic)** — Flattening frequencies (as Vigenère partly does) defeats simple frequency analysis.
If ciphertext frequencies are almost perfectly FLAT, the cipher is probably:	**not a simple monoalphabetic substitution** — Flat frequencies suggest a polyalphabetic or stronger cipher, not simple substitution.
The synthesis of this kit is:	**a big key space does not help if the cipher leaks frequencies** — Substitution shows that key-space size alone is not security — leaked structure breaks it.
Compared to Caesar, a full substitution cipher has:	**a far larger key space but the same frequency weakness** — Bigger key space, but frequency analysis still cracks both.
Frequency analysis is HARDER when the message is:	**very short** — Short messages don't have enough letters for reliable frequency statistics.
Despite the huge key space, substitution ciphers are broken by:	**frequency analysis** — Frequency analysis exploits the preserved letter-frequency pattern, bypassing brute force.
A 'homophonic' substitution assigns common letters:	**several possible ciphertext symbols to flatten frequencies** — Homophonic substitution gives frequent letters multiple symbols to hide their frequency.
The most frequent letter in typical English is:	**E** — E is the most common English letter, followed by T, A, O.
Cracking a substitution cipher is mostly a process of:	**hypothesize a mapping, test it, and refine** — You build the mapping incrementally from frequency + pattern clues, then refine.
A one-letter word in English ciphertext is almost certainly:	**A or I** — Single-letter English words are almost always A or I.
The historical figure credited with early frequency analysis is:	**Al-Kindi** — The 9th-century scholar Al-Kindi described frequency analysis.
Removing spaces from substitution ciphertext:	**makes frequency analysis a bit harder but not impossible** — Hiding word boundaries slows analysis but frequencies still leak.
Frequency analysis works because:	**each plaintext letter's frequency shows up on its fixed substitute** — A fixed mapping carries each letter's frequency to its substitute, revealing the mapping.
A cipher that maps E→X everywhere and T→Q everywhere is a:	**monoalphabetic substitution** — One fixed substitute per letter throughout = monoalphabetic substitution.

The reason a longer ciphertext is easier to crack by frequency is:	**more letters give more reliable statistics** — More data makes the frequency estimates accurate, aiding the attacker.
The letter frequencies of ciphertext under a monoalphabetic cipher form:	**the same shape as English, just relabeled** — The frequency profile keeps English's shape under a fixed relabeling.
A substitution cipher hides the LETTERS but not the:	**word lengths and letter-frequency pattern** — Word spacing and frequencies leak, which is exactly what cracks it.
The number of possible substitution keys (26 letters) is:	**enormous (26 factorial)** — A full substitution has 26! ≈ 4×10^26 keys — far more than Caesar's 25.
Common English DIGRAPHS (letter pairs) like 'TH' and 'HE' help because:	**their frequent patterns reveal more mappings** — Frequent pairs like TH/HE/IN give extra footholds for cracking.
The best first step when handed an unknown substitution ciphertext is:	**count letter frequencies** — Tallying frequencies is the classic opening move against substitution.
A monoalphabetic substitution cipher maps:	**each plaintext letter to one fixed ciphertext letter** — Every letter has one consistent substitute throughout the message.
If 'XLI' appears often as a 3-letter word, it likely decrypts to:	**THE** — THE is the most common English 3-letter word; a frequent 3-letter cipherword is often THE.
Frequency analysis is a form of:	**cryptanalysis (breaking, not making)** — It is a cryptanalysis technique for breaking ciphers.

4. Polyalphabetic; keyword

✂ Question (fold →)	Answer
The Kasiski examination finds the key length by:	**looking for repeated ciphertext sequences and their spacing** — Repeated segments tend to be spaced by multiples of the key length.
To DECRYPT Vigenère, you:	**shift each letter BACK by its keyword-letter shift** — Decryption subtracts the keyword shifts instead of adding them.
A LONGER Vigenère keyword makes the cipher:	**harder to break (more columns, flatter frequencies)** — Longer keys spread letters more and shorten each column, resisting analysis.
Vigenère was historically called:	**'le chiffre indéchiffrable' (the unbreakable cipher)** — It was long thought unbreakable until Kasiski/Babbage methods emerged.
Encrypting plaintext 'HELLO' with keyword 'AB' (shifts 0,1,0,1,0) gives:	**HFLMO** — H+0=H, E+1=F, L+0=L, L+1=M, O+0=O → HFLMO.
The length of the keyword is called the:	**period (or key length)** — The repeating keyword length is the cipher's period.
Vigenère is strong compared to Caesar mainly because it:	**defeats a single-frequency-count attack** — It breaks the single-frequency attack by using many shifting alphabets.
The synthesis of Vigenère is:	**varying the shift beats frequency analysis, but a repeating key still leaks** — Polyalphabetic shifting helps, but key repetition remains the exploitable weakness.
Encrypting plaintext A with keyword letter C (shift 2) gives:	**C** — A shifted by 2 (keyword C) is C.
Modern encryption improves on Vigenère by:	**using keys and operations that don't leave repeating patterns** — Modern ciphers avoid the repetition and structure that doom Vigenère.
In Vigenère, the keyword letter 'B' corresponds to a shift of:	**1** — A=0, B=1, C=2... so keyword letter B = shift 1.
Vigenère resists simple frequency analysis because it:	**spreads each plaintext letter across several ciphertext letters** — The same plaintext letter maps to different ciphertext letters, flattening frequencies.
The Index of Coincidence is a tool used to:	**estimate the key length by measuring how English-like the columns are** — The IoC statistic helps estimate the period by detecting English-like frequency clustering.
If you suspect a keyword of length 4, you split the ciphertext into:	**4 columns and solve each separately** — A period-4 key means every 4th letter shares a shift → 4 Caesar sub-problems.
The first step in breaking Vigenère is usually to:	**find the key LENGTH (period)** — Once you know the period, you split the text into columns and solve each as a Caesar cipher.
If a Vigenère keyword is 'A' repeated, the cipher reduces to:	**no shift (plaintext unchanged)** — Keyword letter A = shift 0, so every letter is unchanged.
The weakness Vigenère still has is:	**the key REPEATS, creating exploitable patterns** — Because the key repeats, structure re-appears and can be attacked.

Encrypting 'AAAA' with keyword 'ABCD' gives:	**ABCD** — A+0=A, A+1=B, A+2=C, A+3=D → ABCD.
The Vigenère cipher differs from Caesar by using:	**a repeating keyword to shift letters by different amounts** — Vigenère applies a series of Caesar shifts determined by a repeating keyword.
Vigenère is called a POLYALPHABETIC cipher because it:	**uses multiple substitution alphabets across the message** — Different positions use different shift alphabets — polyalphabetic.
Once the key length is known, each 'column' of Vigenère is just:	**a Caesar cipher you can frequency-analyze** — Each column shares one shift, so it reduces to a solvable Caesar cipher.
If the keyword is as long as the message and never repeats, you approach a:	**one-time pad (very strong)** — A truly random, non-repeating key as long as the message is a one-time pad — theoretically unbreakable.
A repeating-key cipher becomes weak once the attacker knows:	**the key length** — Knowing the period reduces Vigenère to several Caesar ciphers.
The Vigenère 'tableau' (table) is used to:	**look up the ciphertext letter from the plaintext row and key column** — The tableau is a lookup grid crossing plaintext letter with key letter.
To encrypt with keyword 'KEY', the shifts applied cycle as:	**K, E, Y, K, E, Y, ...** — The keyword repeats over the plaintext, giving a cycling shift pattern.

5. The clock math behind ciphers

✂ Question (fold →)	Answer
'25 mod 26' equals:	**25** — 25 is less than 26, so 25 mod 26 = 25 (no wrap).
Modular arithmetic is often called:	**clock arithmetic** — It wraps around a fixed modulus like hours wrap around a clock.
Modular ADDITION is used in:	**shift ciphers (Caesar, Vigenère)** — Shift ciphers add the key mod 26.
'0 mod 26' equals:	**0** — 0 mod 26 = 0.
Reducing a huge shift like 100 mod 26 first is smart because:	**it gives the equivalent small shift (100 mod 26 = 22)** — 100 mod 26 = 22, so a shift of 100 is the same as a shift of 22.
Two numbers are 'congruent mod n' when they:	**have the same remainder when divided by n** — a ≡ b (mod n) means a and b leave the same remainder mod n.
Encrypting letter value 24 with shift 5, mod 26, gives value:	**3** — 24 + 5 = 29; 29 mod 26 = 3.
3 and 29 are congruent mod 26 because:	**both leave remainder 3** — 29 mod 26 = 3, matching 3 mod 26 = 3.
Modular arithmetic is 'closed' meaning:	**adding/multiplying two residues gives another residue in 0..n−1** — Operations stay within the residue set 0..n−1.
'17 mod 5' equals:	**2** — 17 = 3×5 + 2, so 17 mod 5 = 2.
'(5 − 8) mod 26' equals:	**23** — 5 − 8 = −3; −3 mod 26 = 23 (wrap up by 26).
Modular arithmetic underlies public-key crypto through:	**operations that are easy forward but hard to reverse** — Modular exponentiation is easy to compute but hard to invert — the basis of RSA-style crypto.
Modular arithmetic matters for ciphers because it:	**keeps shifted letters inside the alphabet** — The modulus wraps shifts back into the 0–25 letter range.
Why is mod 26 (not mod 25) used for the 26-letter alphabet?	**there are 26 letters, values 0–25** — 26 letters map to values 0–25, so the modulus is 26.
The synthesis of this kit is:	**modular arithmetic is the reversible clock-math engine beneath shift ciphers** — Wrap-around modular arithmetic is exactly what makes shift ciphers work and reverse.
The result of 'a mod n' is always:	**between 0 and n−1** — A remainder mod n lies in 0..n−1.
The reason ciphers rely on modular math is that it:	**provides reversible, bounded transformations perfect for fixed alphabets** — Bounded, reversible (with inverses) operations fit fixed alphabets and enable both encrypt and decrypt.
The Affine cipher uses the formula E(x) = (a·x + b) mod 26; this requires 'a' to be:	**coprime to 26 (so it is invertible)** — For decryption to work, 'a' must have a modular inverse mod 26, i.e. be coprime to 26.
'a mod n = 0' means:	**n divides a evenly** — A remainder of 0 means n divides a with nothing left over.

Decrypting is the inverse: to undo '+5 mod 26' you compute:	**−5 mod 26 (i.e. +21)** — Subtracting 5 (equivalently adding 21) mod 26 reverses the shift.
Caesar shift math uses modulo:	**26 (the alphabet size)** — Letters wrap around 26, so shift math is done mod 26.
A 'modular inverse' of a mod n is a number that, times a, gives:	**1 (mod n)** — The modular inverse a^{-1} satisfies $a \cdot a^{-1} \equiv 1 \pmod{n}$.
'(20 + 10) mod 26' equals:	**4** — 30 mod 26 = 4.
'30 mod 26' equals:	**4** — 30 − 26 = 4.
On a 12-hour clock, 10 + 5 equals:	**3** — 15 mod 12 = 3 (15 wraps past 12 by 3).

6. Reorder, don't replace

✂ Question (fold →)	Answer
Compared to substitution, transposition:	**keeps the same letters but changes their positions** — Substitution swaps letters; transposition rearranges them.
Transposition ciphers are vulnerable to:	**anagramming and known common words/patterns** — Since letters are unchanged, attackers rearrange (anagram) toward sensible text.
To make transposition stronger you can:	**use a longer, irregular key and combine with substitution** — Longer irregular keys plus substitution (a product cipher) resist analysis.
A transposition cipher's key describes:	**HOW the positions are rearranged** — The key encodes the permutation of positions.
Combining transposition WITH substitution gives:	**a stronger 'product cipher'** — Layering both techniques (a product cipher) resists each single attack.
If you know a transposition used a 4-column grid, you should:	**write the ciphertext into 4 columns and try reading orders** — Reconstruct the grid width and test column read-orders.
The number of columns in a columnar transposition is set by:	**the key length** — The keyword length determines the number of columns.
A columnar transposition writes text into a grid by rows and reads it out by:	**columns (in a key-determined order)** — Columnar transposition reads columns in an order set by the key.
Reading 'THE' written down columns of a 3-wide grid row 'T H E' gives (one letter per column):	**T, H, E** — One letter per column in that single row reads T, H, E straight down.
A rail-fence with 2 rails on 'WEAREDISCOVERED' reads rail 1 then rail 2; this is a form of:	**transposition** — Rearranging by rails is transposition — no letter is replaced.
The best clue that a ciphertext is a transposition (not substitution) is:	**its letter frequencies match plain English** — English-matching frequencies + scrambled words suggest transposition.
To DECRYPT a transposition cipher you must:	**know the rearrangement pattern (the key) and undo it** — You reverse the specific reordering used to encrypt.
The letters of the ciphertext under transposition are:	**exactly the same multiset as the plaintext** — The same letters appear the same number of times, just reordered.
Why does transposition ALONE fail against a determined analyst?	**the plaintext letters are all present to be rearranged into sense** — Nothing is hidden about which letters exist, so the message can be reassembled.
Modern block ciphers use transposition-like steps called:	**permutations (the 'P' in substitution-permutation networks)** — SP-networks combine substitution (S) and permutation (P) steps.
Transposition preserves message LENGTH, which tells an analyst:	**the number of letters, aiding grid guesses** — Knowing the letter count helps guess grid dimensions.
A 'rail fence' cipher writes the	**in a zigzag across rows, then reads row by row** — The rail fence zigzags

message:	letters across rails, then reads them off by row.
A 'double transposition' applies the reordering:	**twice, greatly increasing difficulty** — Applying transposition twice was historically much harder to break.
If you write 'SECRET' down 3 columns row by row and read columns, you get a:	**transposition (reordered) ciphertext** — Reading a row-filled grid by columns reorders letters — transposition.
If you reverse the whole message 'HELLO' you get:	**OLLEH** — Reversal is the simplest transposition: HELLO → OLLEH.
Anagramming attacks are practical when the message is:	**short enough to rearrange by hand or search** — Shorter messages have fewer arrangements to search.
A single transposition is weak because:	**the plaintext letters are all still present to rearrange** — All original letters remain, so patient rearrangement recovers the message.
A key property that survives transposition (and can betray it) is:	**the letter-frequency count is unchanged** — Transposition doesn't change WHICH letters appear, so frequencies match plaintext — a giveaway.
The synthesis of this kit is:	**reordering hides letter POSITIONS but not letter IDENTITY, so it is weak alone** — Transposition conceals order but leaves identities/frequencies exposed — strong only when combined.
A transposition cipher works by:	**rearranging the ORDER of the letters, not replacing them** — Transposition keeps the letters but scrambles their positions.

7. The key vs the algorithm

✂ Question (fold →)	Answer
Key MANAGEMENT is about:	**generating, sharing, storing, and retiring keys safely** — Good key management (create/share/store/rotate) is as important as the cipher.
Why is a public algorithm with a strong key better than a secret weak one?	**secrecy of the algorithm always leaks eventually; key strength endures** — Algorithms get reverse-engineered or leaked; a strong secret key is the durable defense.
A cryptosystem whose security depends on nobody knowing how it works is:	**violating Kerckhoffs's principle** — Depending on hidden design is exactly the obscurity trap Kerckhoffs warns against.
The KEY and the ALGORITHM differ in that the key is:	**the changeable secret; the algorithm is the fixed public method** — The algorithm is the public procedure; the key is the secret parameter.
The famous restatement 'the enemy knows the system' is:	**Shannon's maxim, echoing Kerckhoffs** — Shannon's maxim assumes the adversary knows the system — so security must rest on the key.
Public scrutiny of an algorithm tends to make it:	**stronger over time as flaws are found and fixed** — Open analysis surfaces and fixes weaknesses, hardening the algorithm.
The one thing that MUST stay secret in a good cryptosystem is:	**the key** — Only the key needs to be secret; the algorithm can be public.
The synthesis of this kit is:	**design so only the KEY is secret; never depend on hiding the algorithm** — Kerckhoffs: put all the secrecy in the key, and let the algorithm be open.
If an attacker learns the ALGORITHM but not the KEY, a Kerckhoffs-compliant system is:	**still secure** — Losing the algorithm's secrecy doesn't matter if the key remains secret.
Modern standards like AES are:	**publicly published and analyzed** — AES is a public standard whose security rests on the key, not secrecy of the design.
The 'key space' is:	**the set of all possible keys** — The key space is every possible key an attacker might have to try.
Kerckhoffs's principle states that a cryptosystem should be secure even if:	**everything except the key is public knowledge** — Security must rest on the secret key, not on hiding the algorithm.
Reusing the same key for many messages is risky because:	**patterns across messages can leak information** — Key reuse creates exploitable relationships between ciphertexts.
A larger key space makes brute force:	**exponentially harder** — More possible keys means far more work to try them all.
A defender applying Kerckhoffs's principle would:	**use a public, vetted algorithm and protect the key** — Rely on vetted public algorithms and focus effort on protecting keys.
If a key is compromised, the right response is to:	**rotate to a new key and re-secure affected data** — Compromised keys must be rotated (replaced) promptly.
Kerckhoffs's principle is important	**open, well-reviewed standards with secret keys are the norm** —

today because:	Modern crypto follows Kerckhoffs: public algorithms, secret keys.
A 128-bit key has roughly how many possibilities?	**about 3.4 × 10^38 (2^128)** — 2^128 ≈ 3.4×10^38 — astronomically large.
If an organization's whole security rested on nobody knowing its cipher, a single leak would:	**compromise everything at once** — Obscurity-based security collapses entirely once the hidden method leaks.
If two systems use the same public algorithm but different keys, an attacker who breaks one:	**cannot automatically read the other** — Different keys isolate systems; breaking one key doesn't reveal another.
'Security through obscurity' means relying on:	**keeping the ALGORITHM secret for protection** — Obscurity hides the method rather than using a strong secret key — a fragile approach.
Choosing a key at RANDOM from a large space matters because:	**predictable keys are easy to guess** — A key must be unpredictable; predictable keys shrink the effective key space.
Kerckhoffs's principle implies obscurity is:	**a weak foundation — the design should survive being public** — If secrecy depends on hiding the algorithm, one leak breaks everything.
A password is essentially a kind of:	**key that unlocks access** — A password acts as a secret key controlling access.
A benefit of PUBLIC algorithms is that:	**many experts can review them for weaknesses** — Open algorithms get scrutinized publicly, exposing flaws before attackers exploit them.

8. Shared key vs key pair

✂ Question (fold →)	Answer
The KEY-SHARING problem that symmetric crypto faces is:	**how to share the secret key securely in the first place** — Symmetric crypto needs a secure channel to share the key — the classic distribution problem.
Asymmetric crypto solves key distribution because:	**public keys can be shared openly without compromising security** — Public keys are safe to publish, so no secret needs pre-sharing.
If your PRIVATE key is stolen, an attacker can:	**decrypt messages meant for you (and impersonate you in signing)** — A stolen private key lets the thief decrypt your messages and forge your signatures.
Only the recipient can decrypt it using their:	**private key** — Only the matching PRIVATE key unlocks it.
AES is an example of a:	**symmetric cipher** — AES is a widely used symmetric block cipher.
Symmetric and asymmetric crypto are best seen as:	**complementary tools used together in real systems** — Real systems combine them: asymmetric for exchange, symmetric for speed.
The main DISADVANTAGE of asymmetric crypto is that it is:	**much slower than symmetric encryption** — Public-key operations are computationally heavier, so it's slower than symmetric.
The number of keys needed for N people to all talk pairwise with SYMMETRIC keys grows:	**quadratically (a key per pair)** — Symmetric pairwise keys grow as N(N−1)/2 — a scaling problem asymmetric crypto eases.
The reason asymmetric crypto is 'revolutionary' is that it:	**removed the need to secretly pre-share a key** — Public-key crypto let parties communicate securely without a prior shared secret — transformative.
Symmetric encryption is preferred for LARGE data because it is:	**fast** — Symmetric ciphers are efficient for bulk data.
In public-key encryption, you encrypt a message to someone using their:	**public key** — You lock the message with the recipient's PUBLIC key.
RSA is an example of a:	**asymmetric (public-key) system** — RSA is a classic public-key (asymmetric) system.
The keys in an asymmetric pair are related so that:	**what one key locks, only the other can unlock** — The public and private keys are mathematical inverses for locking/unlocking.
If Alice wants only Bob to read her message, she encrypts with:	**Bob's public key** — Encrypting with Bob's public key ensures only Bob (with his private key) can read it.
Asymmetric (public-key) encryption uses:	**a pair of keys — a public key and a private key** — Asymmetric crypto uses a mathematically linked public/private key pair.
With asymmetric crypto, N people need only:	**one key pair each (N public keys published)** — Each person publishes one public key, avoiding the pairwise explosion.
A PUBLIC key is safe to:	**share with anyone** — Public keys are meant to be distributed freely.
	symmetric is fast but needs a shared secret; asymmetric solves

The synthesis of this kit is:	**sharing but is slow — so combine them** — Each solves the other's weakness; hybrid systems use both.
Encrypting with someone's public key guarantees:	**confidentiality (only they can read it)** — Public-key encryption gives confidentiality to the key's owner.
A hybrid system (like TLS) uses asymmetric crypto mainly for:	**the initial secure key exchange** — TLS uses public-key crypto to agree a symmetric session key, then symmetric for the data.
Symmetric encryption uses:	**one shared secret key for both encrypt and decrypt** — Symmetric crypto shares one secret key for both directions.
You need to encrypt a 2 GB video to send to a partner who already shares a secret key with you. Best choice:	**symmetric encryption (fast for large data)** — With a shared key and large data, symmetric encryption is the fast, correct tool.
In practice, systems combine both by:	**using asymmetric crypto to share a symmetric key, then symmetric for the data** — Hybrid systems exchange a fast symmetric key via slow asymmetric crypto — best of both.
The main advantage of asymmetric crypto is that it:	**lets strangers communicate securely without pre-sharing a secret** — You can publish a public key so anyone can send you secrets without a prior shared secret.
A defender choosing encryption for a big file transfer to a known partner with a shared secret would use:	**symmetric encryption (fast)** — With a shared secret already in place, symmetric is the fast, appropriate choice.

9. Locked-box exchange; public/private keys

✂ Question (fold →)	Answer
A man-in-the-middle can defeat an UNauthenticated key exchange by:	**substituting their own keys for both parties** — Without identity verification, the attacker swaps in their keys and relays traffic.
Others VERIFY a digital signature using the signer's:	**public key** — Verification uses the signer's PUBLIC key — the inverse of signing.
The 'locked box' analogy for public-key crypto is:	**anyone can snap your open padlock shut, but only your key opens it** — A published padlock (public key) lets anyone lock a box only you (private key) can open.
A key-exchange scheme is only as trustworthy as:	**the authentication of who holds each key** — Without verifying identities, exchanged keys could belong to an attacker.
A 'man-in-the-middle' attack on key exchange works by:	**secretly relaying and swapping keys between the two parties** — An attacker in the middle can substitute their own keys if identities aren't verified.
In Diffie–Hellman, each party keeps secret their:	**private value** — Each keeps a private exponent secret and shares only a public value.
HTTPS uses a handshake that first:	**authenticates the server and agrees a shared session key** — The TLS handshake authenticates the server (certificate) and negotiates a session key.
The synthesis of this kit is:	**public/private key pairs let strangers exchange secrets and verify identities — if authentication holds** — Key pairs enable secret exchange + signatures, anchored by authenticated identities.
An eavesdropper watching a Diffie–Hellman exchange sees the public values but:	**cannot feasibly compute the shared secret** — The hard math (discrete log) stops eavesdroppers from deriving the secret.
A digital signature provides:	**authenticity and integrity (who sent it + that it's unchanged)** — Signatures prove the sender and that the content wasn't altered.
A digital SIGNATURE is created with the signer's:	**private key** — You sign with your PRIVATE key so others can verify with your public key.
Forward secrecy means that if a long-term key is later stolen:	**past session keys remain safe** — Forward secrecy ensures compromising a long-term key doesn't expose earlier sessions.
The public-key handshake solves the ancient problem of:	**establishing a shared secret with someone you've never met** — It lets strangers agree a secret securely over open channels.
To defend against man-in-the-middle, key exchange needs:	**authentication of the parties' identities (e.g. certificates)** — Verifying identities (certificates/signatures) stops key substitution.
Diffie–Hellman key exchange lets two people:	**derive the same shared secret without ever transmitting it** — DH mixes public values so both compute the same secret an eavesdropper can't.
The defensive lesson of handshakes is:	**encryption without authentication can be defeated by a man-in-the-middle** — Confidentiality needs authenticated key exchange or MITM undoes it.
If a certificate is EXPIRED or	**warn the user before proceeding** — Browsers warn on invalid certificates

untrusted, a browser should:	because identity can't be trusted.
The security of Diffie–Hellman rests on a math problem that is:	**easy to compute forward but hard to reverse** — Modular exponentiation is easy; the discrete-log inverse is hard.
A Certificate Authority (CA) is trusted to:	**confirm that a public key really belongs to a claimed identity** — CAs vouch that a public key belongs to the stated owner.
The 'web of trust' is an alternative to CAs where:	**users vouch for each other's keys directly** — In a web of trust, people sign each other's keys to build trust without a central CA.
A public key can be shared on a website because:	**it can only LOCK messages, not unlock them** — The public key only encrypts/verifies; it can't decrypt/sign, so sharing it is safe.
A key EXCHANGE protocol lets two parties:	**agree on a shared secret over an insecure channel** — Key exchange establishes a shared secret even when eavesdroppers watch the channel.
A CERTIFICATE binds a public key to:	**an identity, vouched for by a trusted authority** — A certificate ties a public key to an identity, signed by a CA.
If Bob signs a document and Alice verifies with Bob's public key, Alice learns:	**Bob signed it and it wasn't altered** — Successful verification proves Bob's signature and the document's integrity.
After the handshake, the bulk of HTTPS data is encrypted with:	**a fast symmetric session key** — The negotiated symmetric session key encrypts the actual data efficiently.

10. One-way functions; fingerprints; integrity

✂ Question (fold →)	Answer
MD5 and SHA-1 are now considered:	**weak (collisions are findable) — avoid for security** — MD5/SHA-1 have practical collision attacks and shouldn't be used for security.
A key property of a good hash is that it is:	**one-way (hard to reverse to the input)** — You can't feasibly recover the input from the digest.
If the computed hash differs from the published hash, the file was:	**altered or corrupted** — A mismatch means the file changed — a tampering or corruption signal.
To confirm a downloaded installer wasn't tampered with, you compare its SHA-256 hash to:	**the hash the publisher officially posted** — Matching the official published hash confirms integrity.
To VERIFY a downloaded file is intact, you:	**hash it and compare to the published hash** — Matching the file's hash to the published one confirms integrity.
A digital signature typically signs:	**the hash of the message, not the whole message** — Signing the compact hash is efficient and binds the whole message via its digest.
If a website emails you your password in plaintext, that suggests:	**they are NOT hashing passwords properly (a red flag)** — Being able to send your actual password means they store it recoverably — bad practice.
Because hashing is one-way, a stolen database of salted password hashes:	**still requires expensive guessing to recover passwords** — Salted, slow hashes force attackers into costly per-password guessing.
If one bit of the input changes, a good hash's output:	**changes drastically (avalanche effect)** — The avalanche effect means tiny input changes flip about half the output bits.
Storing PASSWORDS, systems should store:	**a salted hash of the password, not the password itself** — Salted hashes let systems verify passwords without storing them in readable form.
A cryptographic hash function takes any input and produces:	**a fixed-size fingerprint (digest)** — A hash maps any input to a fixed-length digest.
A hash is 'deterministic', meaning:	**the same input always gives the same output** — Determinism lets you compare hashes to verify data.
The right defensive stance on hashing is:	**use a strong, slow, salted hash for passwords and a strong hash for integrity** — Passwords: slow+salted (Argon2/bcrypt); integrity: strong hash (SHA-256).
Hashing helps blockchains by:	**chaining blocks via their hashes so tampering is detectable** — Each block includes the prior block's hash, so altering history breaks the chain.
Hashing is used mainly to check:	**integrity (that data hasn't changed)** — Comparing hashes detects any change to the data.
The 'integrity' guarantee from a hash means:	**you can detect if data changed, not who changed it** — A bare hash detects change; proving WHO needs a signature or keyed hash (HMAC).
A 'checksum' is a simpler cousin of a hash used mainly for:	**detecting accidental errors (not malicious tampering)** — Checksums catch accidental corruption but aren't secure against deliberate tampering.

Password hashing SHOULD use a SLOW function (like bcrypt/Argon2) to:	**make brute-force guessing expensive** — Deliberately slow hashes raise the cost of guessing many passwords.
A 'salt' added before hashing a password:	**makes identical passwords hash differently and defeats precomputed tables** — A unique salt per password stops rainbow-table attacks and hides duplicate passwords.
The synthesis of this kit is:	**a hash is a one-way fingerprint proving integrity, not a secret-keeping tool** — Hashing verifies integrity via an irreversible fingerprint — distinct from encryption.
An HMAC adds a secret key to a hash to provide:	**integrity AND authenticity** — HMAC keys the hash so only key-holders can produce a valid tag — integrity + authenticity.
Unlike encryption, hashing:	**has no key and cannot be decrypted** — Hashing is keyless and one-way; encryption is keyed and reversible.
A 'collision' occurs when:	**two different inputs produce the same hash** — A collision is two distinct inputs sharing one digest — good hashes make these infeasible to find.
A recommended modern hash family is:	**SHA-256 (SHA-2)** — SHA-256 (part of SHA-2) is a current standard general-purpose hash.
A 'rainbow table' attack precomputes:	**hashes of common passwords to reverse-lookup a stolen digest** — Rainbow tables map precomputed hashes back to inputs — salting defeats them.

11. Hidden in plain sight

✂ Question (fold →)	Answer
Steganography and encryption together follow the principle of:	**defense in depth (layered protection)** — Layering hiding + scrambling is defense in depth.
Historically, invisible ink is an example of:	**steganography** — Invisible ink hides the existence of writing — classic steganography.
A photo file that looks normal but is twice the expected size might contain:	**data hidden by steganography** — Unexpected size can indicate embedded hidden data.
A watermark differs from covert steganography in that a watermark is:	**often meant to be detectable (proving ownership)** — Watermarks may be detectable for ownership; covert steg aims to stay unnoticed.
Steganography is the art of:	**hiding the EXISTENCE of a message inside an innocent cover** — Steganography conceals that a message exists at all, unlike encryption which scrambles a known message.
A CTF challenge that provides an image and hints at a hidden flag is testing:	**steganography extraction skills** — An image-with-hidden-flag challenge tests steg extraction.
If you suspect steganography, a good first tool is:	**a metadata/EXIF viewer and a hex viewer** — Metadata viewers and hex editors reveal hidden fields and anomalies.
Steganography does NOT provide:	**strong confidentiality on its own** — Hiding existence isn't the same as making found data unreadable — pair with encryption.
The word 'steganography' comes from Greek roots meaning:	**'covered writing'** — 'Steganos' (covered) + 'graphein' (writing) = covered writing.
A giveaway of LSB image steganography under analysis is:	**statistical anomalies in the low-bit distribution** — Embedding disturbs the natural statistics of the least-significant bits.
The defensive takeaway about steganography is:	**monitor for anomalies, because hidden data can bypass content filters** — Since steg evades content inspection, defenders watch for statistical/metadata anomalies.
The synthesis of this kit is:	**steganography hides that a message exists — powerful when layered with encryption, weak alone** — Hiding existence complements, but doesn't replace, encryption.
Combining steganography with encryption gives:	**hidden AND unreadable-if-found data** — Encrypt then hide: even if the hidden data is found, it's still encrypted.
A message hidden in an image but NOT encrypted is:	**readable once extracted** — If found and extracted, plaintext hidden data is simply readable.
A 'cover' in steganography is:	**the innocent file that carries the hidden message** — The cover is the ordinary-looking carrier (image/audio/text).
'Steganalysis' is:	**the practice of detecting hidden messages** — Steganalysis is the counter-discipline: detecting steganography.

A file that is larger than expected for its visible content might:	**contain hidden data (a steg clue)** — Unexpected size can hint at embedded hidden content.
A forensic examiner looks for hidden data by:	**checking metadata, file size, and bit-level patterns** — Examiners inspect metadata, sizes, and low-bit statistics for anomalies.
Steganography FAILS as a defense if:	**the hiding method is known and the cover is analyzed** — Once the method/cover is understood, hidden data can be extracted — hence pair it with encryption.
Audio files can hide data in:	**inaudible frequency ranges or low-order sample bits** — Audio steg uses inaudible ranges or low sample bits.
Changing the LEAST-significant bit of a pixel color:	**barely changes the color, so it's visually hidden** — The lowest bit changes color imperceptibly, hiding data in the noise.
The white-hat use of steganography knowledge is to:	**detect data exfiltration hidden in innocent-looking files** — Defenders study steg to catch data smuggled out inside ordinary files.
The key difference between steganography and cryptography is:	**steganography hides existence; cryptography hides meaning** — Crypto makes a message unreadable; steg makes it unnoticed.
Hiding text in the METADATA of a file (e.g. comments) is:	**a simple steganographic technique** — Data tucked into file metadata/comments is a basic hiding method.
A classic modern steganography method hides data in:	**the least-significant bits of image pixels** — LSB steganography tweaks the lowest pixel bits, invisibly to the eye.

12. Crib / pattern attacks

✂ Question (fold →)	Answer
The synthesis of this kit is:	**known plaintext turns predictability into a crack — strong ciphers must resist it** — Cribs exploit predictability; modern security must be immune to known-plaintext.
Knowing plaintext P and ciphertext C for a Caesar cipher lets you:	**compute the shift directly (C − P)** — For a shift cipher, one known letter pair gives the key immediately.
A one-time pad resists known-plaintext attack because:	**the key is as long as the message and never reused** — With a random, non-reused, full-length key, known plaintext reveals only that segment's key.
The white-hat value of studying crib attacks is to:	**recognize predictable patterns that weaken systems and remove them** — Defenders find and eliminate predictable structure that would give attackers cribs.
The reason WWII codebreakers valued captured codebooks is that they:	**provided known plaintext to bootstrap attacks** — Captured material gave known plaintext to jump-start cryptanalysis.
Modern ciphers are designed to resist known-plaintext attacks by:	**not leaking key information even when plaintext is known** — Strong ciphers stay secure even if the attacker knows plaintext/ciphertext pairs.
If an analyst guesses a crib in the WRONG position, they:	**get nonsense and try another position** — A misaligned crib yields nonsense; the analyst slides it to other positions.
A repeating-key cipher is especially vulnerable to cribs because:	**the crib reveals the key, which then repeats over the message** — Recovering key letters via a crib breaks the whole message since the key repeats.
The Enigma was famously broken partly using cribs like:	**predictable German phrases (e.g. weather reports)** — Bletchley Park exploited predictable phrases (cribs) to attack Enigma.
The best way to deny an attacker cribs is to:	**use strong modern encryption where known plaintext gives no advantage** — Modern ciphers make cribs useless — the real defense.
A crib attack is a form of:	**cryptanalysis (breaking)** — Crib/known-plaintext attacks are cryptanalysis techniques.
A cipher resistant to known-plaintext attack is said to have:	**strong key-plaintext independence** — The key stays hidden even given plaintext/ciphertext pairs.
Common cribs in messages include:	**predictable words like greetings, dates, or 'THE'** — Standard openings, dates, and frequent words are classic cribs.
Against Vigenère, a crib helps by:	**revealing key letters at the crib's position** — Aligning a crib exposes the keyword letters where it sits.
If you know a message starts with 'DEAR', aligning it with the ciphertext start:	**may reveal the first key letters** — Aligning a known opening exposes the corresponding key material.
Knowing a Caesar-enciphered message begins with the word 'THE' lets you:	**compute the shift immediately** — One known plaintext word reveals the shift for the whole Caesar message.

The defensive lesson from crib attacks is:	**avoid predictable, repeated message structure** — Predictable structure gives attackers cribs — avoid it (and use strong ciphers).
A known-plaintext attack requires the attacker to have:	**at least some matching plaintext–ciphertext** — KPA needs some plaintext known to correspond to observed ciphertext.
A 'crib' is:	**a guessed or known fragment of the plaintext** — A crib is a likely word/phrase the analyst expects in the plaintext.
Predictable auto-generated email footers can act as:	**cribs that weaken a poorly-encrypted message** — Boilerplate footers are predictable text an attacker can exploit as a crib.
Cribs are more useful when the message format is:	**standardized (headers, salutations, timestamps)** — Standard formats provide reliable predictable text to use as cribs.
A chosen-plaintext attack is stronger because the attacker can:	**pick specific plaintexts to encrypt and study the outputs** — Choosing inputs lets the attacker probe the cipher systematically.
Known-plaintext attacks illustrate that a cipher's security must not depend on:	**the secrecy of the message content** — If knowing content breaks the cipher, it's weak; security must rest on the key.
A known-plaintext attack uses:	**a piece of plaintext you already know to help break the rest** — Knowing some plaintext (a 'crib') exposes key information to crack more.
A crib of a KNOWN filename inside an encrypted archive can help attack:	**weak or homemade encryption** — Predictable structure aids attacks on weak ciphers, not strong modern ones.

13. Strength, entropy, reuse

✂ Question (fold →)	Answer
Rate-limiting login attempts defends against:	**online brute-force and guessing** — Limiting attempts slows guessing attacks dramatically.
Even a strong password is weakened if:	**it is reused or phished** — Reuse and phishing defeat even strong passwords.
The synthesis of this kit is:	**strength comes from entropy (length + randomness) and never reusing passwords** — High entropy plus non-reuse (plus 2FA) is what actually protects accounts.
If a service is breached, you should:	**change that password and any reused copies elsewhere** — Change the exposed password and every place you reused it.
Modern guidance favors:	**long passphrases over short complex passwords** — Current guidance (e.g. NIST) favors length/passphrases and discourages forced frequent changes.
A password MANAGER helps by:	**generating and storing unique strong passwords for each site** — Managers create/store unique strong passwords so you don't reuse or memorize them.
Password 'entropy' measures:	**how unpredictable (how many possibilities) a password is** — Entropy quantifies unpredictability — more possibilities means a stronger password.
Between 'P@ss1' and a 5-random-word passphrase, the stronger is:	**the passphrase (far more entropy)** — Length/randomness of a multi-word passphrase far exceeds a short symbol-laden password.
Which password is strongest?	**a long random passphrase of unrelated words** — Length + randomness (a long unrelated-word passphrase) beats short predictable passwords.
The biggest factor in password strength is usually:	**length** — Length increases the search space the most; longer generally beats clever-but-short.
Password 'strength meters' can mislead because they:	**reward complexity rules over true length/randomness** — Many meters overvalue symbol rules and undervalue length/true randomness.
A 4-digit PIN has low entropy because:	**there are only 10,000 possibilities** — 10^4 = 10,000 combinations is small and quickly brute-forced without rate limits.
Reusing the same password across sites is dangerous because:	**one breach exposes all your accounts (credential stuffing)** — A leak on one site lets attackers 'stuff' the same credentials into your other accounts.
Storing passwords, a site should keep:	**a salted slow hash** — Salted slow hashes (Argon2/bcrypt) are the standard for password storage.
Two-factor authentication (2FA) adds security by requiring:	**something you have or are, in addition to your password** — 2FA adds a second factor (phone/token/biometric) beyond the password.
A 'credential stuffing' attack relies on:	**reused passwords leaked from other breaches** — Attackers replay leaked reused credentials across many sites.
Adding one more RANDOM character to a password increases the possibilities:	**multiplicatively (by the size of the character set)** — Each random character multiplies the possibilities by the alphabet size — exponential growth with length.

A 'brute-force' password attack:	**tries every possible combination** — Brute force exhaustively tries combinations — length makes this infeasible.
The single best habit for account security is:	**a unique strong password per site plus 2FA** — Unique passwords (via a manager) + 2FA is the strongest practical habit.
A passphrase like four random unrelated words is strong because:	**its length creates enormous entropy while staying memorable** — Several random words yield high entropy and are easier to remember than random symbols.
Entropy is measured in:	**bits** — Password entropy is expressed in bits (log2 of the number of possibilities).
The reason NOT to use personal info (birthday, pet) is that it is:	**easily guessed or found on social media** — Personal details are discoverable/guessable, lowering effective entropy.
Biometrics (fingerprint/face) are best used as:	**one factor among several, not a sole secret** — Biometrics can't be changed if leaked, so they work best as one factor in multi-factor auth.
The defensive summary for passwords is:	**length + uniqueness + a manager + 2FA** — Prioritize length, uniqueness (a manager), and 2FA over arbitrary complexity rules.
A common attack that tries many common passwords is a:	**dictionary attack** — Dictionary attacks try lists of common passwords first.

14. Recognize the fake (defensive)

✂ Question (fold →)	Answer
The white-hat/defensive goal in studying phishing is to:	**recognize and stop attacks, and teach others to** — Studying phishing is to detect, prevent, and educate — never to attack.
A phishing email with your name and recent purchase is more dangerous because it:	**builds credibility with personal details** — Personal details make the lure more believable — the spear-phishing edge.
Phishing is an attack that:	**tricks people into revealing secrets or clicking malicious links** — Phishing manipulates PEOPLE (social engineering), not the cryptography.
If you accidentally entered your password on a phishing site, you should:	**change that password immediately and enable 2FA** — Assume it's compromised: change it (and reuses) and turn on 2FA.
Before clicking a link in an email, you should:	**hover to inspect the real destination URL** — Hovering reveals the true URL, which often mismatches the claimed sender.
Two-factor authentication helps against phishing by:	**requiring a second factor even if the password is stolen** — Even a phished password fails without the second factor (though phishing-resistant 2FA is best).
A login page at 'paypa1-secure.com' is suspicious because:	**the domain is a look-alike, not the real one** — Look-alike/misspelled domains impersonate the real site.
'Smishing' is phishing conducted via:	**SMS text messages** — Smishing uses SMS texts to phish.
A shortened/obscured link is risky because it:	**hides the true destination until clicked** — Link shorteners conceal the real URL, so expand/inspect before trusting.
The best response to a suspicious 'verify your account' email is to:	**go directly to the official site yourself, not via the link** — Navigate to the real site independently rather than trusting the emailed link.
The most reliable defense against social engineering is:	**a healthy habit of verifying requests through a trusted channel** — Verifying unusual requests independently defeats most social engineering.
A message claiming to be your bank but sent from a free email address is:	**likely a phishing attempt** — Banks don't use free/random email domains — a sender mismatch is a red flag.
'Spear phishing' differs from ordinary phishing in that it is:	**targeted at a specific person using personal details** — Spear phishing tailors the lure to a specific target using researched details.
An attacker calling to say 'I'm from IT, I need your password to fix your account' is using:	**social engineering (pretexting)** — Legit IT never needs your password — this is a social-engineering pretext.
Checking the actual sender ADDRESS (not just the display name) matters because:	**display names can be faked while the real address reveals the fraud** — Attackers spoof display names; the underlying address often exposes the fake.
A classic phishing red flag is:	**urgent pressure to act immediately** — Manufactured urgency pressures targets to skip careful checking.

An email says 'Your package is held — pay a fee at this link now.' The safest action is:	**ignore the link and check the carrier's official site directly** — Verify via the official site yourself; never act on the emailed link's urgency.
The synthesis of this kit is:	**phishing attacks people, not math — awareness, verification, and 2FA are the defenses** — Since phishing targets humans, human-centered defenses (awareness/verify/2FA) matter most.
The human is often called the:	**weakest link in security** — Social engineering targets people because humans are often the weakest link.
'Baiting' lures victims with:	**something enticing (like a free USB drive) that carries malware** — Baiting dangles an enticing item (free download, dropped USB) to trigger a click/insert.
'Pretexting' is social engineering that uses:	**a fabricated scenario/identity to gain trust** — Pretexting invents a believable story/role (e.g. 'IT support') to extract info.
Organizations reduce phishing risk by:	**training staff and simulating phishing to build awareness** — Awareness training and simulations build the human 'firewall'.
A legitimate-looking email that asks you to 'confirm your password' should be treated as:	**phishing — real services don't ask for your password by email** — Reputable services never ask you to send/confirm your password via email.
The single most protective habit against phishing is:	**slow down and verify before acting on any urgent request** — Pausing to verify defeats the urgency that phishing relies on.
'Vishing' is phishing conducted via:	**voice calls** — Vishing = voice phishing (phone-based social engineering).

15. Find the flag in the data

✂ Question (fold →)	Answer
A defender uses forensics after an incident to:	**understand what happened and prevent recurrence** — Post-incident forensics reveals the cause and informs defenses.
File METADATA (like EXIF in photos) can reveal:	**creation time, device, and sometimes location** — EXIF/metadata often holds timestamps, device model, and GPS.
The right mindset for forensics is:	**careful, methodical, and evidence-preserving** — Forensics values methodical, non-destructive, well-documented analysis.
The first step examining an unknown file is usually to:	**identify its true type via the magic number** — Confirming the real format (magic number) guides all further analysis.
A file named 'photo.png' whose first bytes are the PDF signature '%PDF' is really:	**a PDF (renamed)** — The '%PDF' magic number reveals the true type regardless of the .png name.
Digital forensics is the practice of:	**recovering and analyzing digital evidence methodically** — Forensics recovers and analyzes data (files, metadata, artifacts) systematically.
A common CTF forensics task is to:	**extract a hidden flag from a file's data or metadata** — Forensics CTF challenges hide a flag in metadata, bytes, or embedded content.
Base64-looking text inside a file might be:	**encoded data worth decoding** — Base64 blobs often decode to readable data or files.
Tools like a hex viewer, 'strings', and an EXIF reader are the forensic analyst's:	**basic toolkit for inspecting data** — These inspection tools form the everyday forensic toolkit.
'Strings' analysis of a binary file extracts:	**human-readable text embedded in it** — The 'strings' technique pulls readable text out of binary data.
Deleted files can often be recovered because:	**the data may remain on disk until overwritten** — Deletion typically just unlinks the file; the bytes persist until overwritten.
A forensic analyst preserves evidence integrity by:	**hashing the original and working on copies** — Hashing proves the copy matches the original; analysis is done on copies.
Finding readable text in an image file's trailing bytes suggests:	**appended hidden data (steganography or a comment)** — Data after the image's end marker is often hidden/appended content.
A file's true type is best determined by its:	**'magic number' (signature bytes at the start)** — Signature bytes (magic numbers) identify the real format regardless of the extension.
To verify two files are IDENTICAL, compare their:	**hashes** — Matching hashes confirm identical content.
A '.jpg' file whose bytes start with 'PK' is really:	**a ZIP archive (renamed)** — 'PK' is the ZIP signature — the extension was changed to disguise it.
A log file entry showing many failed logins then one success suggests:	**a possible brute-force or guessing attempt** — A burst of failures then success is a classic guessing/brute-force signature.

An analyst who finds Base64 that decodes to 'flag{...}' has:	**recovered the challenge flag** — Decoding to a 'flag{...}' pattern means the hidden flag was recovered.
The synthesis of this kit is:	**evidence hides in bytes, metadata, and logs — methodical inspection reveals it** — Careful, methodical inspection of data/metadata/logs surfaces the evidence (or the flag).
A 'timeline' in forensics is built from:	**timestamps across files, logs, and metadata** — Correlating timestamps reconstructs the sequence of events.
The 'chain of custody' documents:	**who handled the evidence and when** — Chain of custody records handling to keep evidence trustworthy.
A 'hex viewer' lets an analyst:	**inspect the raw bytes of a file** — Hex viewers show raw bytes, revealing hidden structure or embedded data.
Forensics differs from a live attack in that it is:	**analysis of data/evidence, usually after the fact** — Forensics analyzes evidence (often post-incident); it isn't live exploitation.
Metadata can be a PRIVACY risk because photos may leak:	**the location (GPS) where they were taken** — EXIF GPS can inadvertently reveal where a photo was taken.
Network forensics analyzes:	**captured traffic (packets) to understand activity** — Packet captures reveal connections, protocols, and sometimes payloads.

16. Multi-stage CTF

✂ Question (fold →)	Answer
Working in a TEAM on a CTF is valuable because:	**different specialties (crypto/forensics/web) cover more categories** — Teams pool complementary skills across categories.
Reflecting AFTER a CTF (what worked, what to learn) is valuable because it:	**turns the solve into durable skill** — Post-solve reflection (the AAR habit) consolidates the learning.
A challenge giving ciphertext with a shift pattern is likely testing:	**a classical cipher (e.g. Caesar/Vigenère)** — Shift-patterned ciphertext points to a classical cipher challenge.
The 'flag' in a CTF is:	**a token you recover to prove you solved the challenge** — The flag is a proof-of-solve token (often 'flag{...}').
A challenge giving an image and hinting at hidden data tests:	**steganography** — An image-with-hidden-flag is a steganography challenge.
The learning value of CTFs is that they:	**build real defensive skills in a legal, safe setting** — CTFs build practical security skills ethically and safely.
A 'crypto' CTF category tests:	**encoding, ciphers, hashing, and key concepts** — Crypto challenges cover encoding, ciphers, hashes, and key concepts.
The FIRST step on any CTF challenge is to:	**identify what technique the challenge is testing** — Naming the category/technique (crypto, forensics, steg...) guides your approach — the white-hat habit.
If Base64 decoding yields more ciphertext, the next step is to:	**identify and break that cipher** — Layered challenges: decode, then tackle the revealed cipher.
A CTF flag format like 'flag{...}' helps solvers:	**recognize when they've found the answer** — A known format signals a correct find.
A CTF challenge titled 'Shifty' providing scrambled letters most likely wants you to:	**identify and break a shift/classical cipher** — The title + scrambled letters point to a classical shift-cipher challenge.
A Capture-the-Flag (CTF) challenge is:	**a legal, sandboxed puzzle to practice security skills** — CTFs are legal, sandboxed exercises for learning security by solving puzzles.
The best summary of the white-hat mindset is:	**understand weaknesses to defend, always with authorization** — White-hat: learn weaknesses to defend, only with authorization — the ethical core.
The ethical boundary a CTF teaches is:	**practice offense to build DEFENSE, within authorized scope only** — CTFs teach that offensive knowledge serves defense, only within authorized scope.
Recovering a flag from a salted-hash challenge would rely on:	**the concept that hashes are one-way (so you match, not reverse)** — You can't reverse a hash; challenges hinge on matching/known values or weaknesses.
A challenge giving a	

mysterious file with no extension tests:	**forensics (identify + extract)** — An unknown file to identify and extract from is forensics.
A challenge combining an image, a Base64 blob, and a shift cipher is a:	**multi-technique (chained) challenge** — Multiple artifacts signal a chained, multi-technique challenge.
The reason CTFs use a sandbox is to:	**let learners practice attacks safely without harming real systems** — Sandboxes contain the practice so no real systems are harmed.
A multi-stage CTF often requires you to:	**chain techniques — decode, then decrypt, then extract** — Stages chain: e.g. Base64 → cipher → steg → flag.
The white-hat rule that MUST hold in every CTF is:	**only attack the sandbox provided, never real systems** — CTF ethics: operate only within the provided sandbox, never real/unauthorized systems.
If a CTF clue says 'the key is the algorithm's only secret,' it is invoking:	**Kerckhoffs's principle** — That phrasing is Kerckhoffs's principle — only the key is secret.
A challenge that provides a public key and ciphertext tests understanding of:	**asymmetric encryption concepts** — Public-key + ciphertext points to asymmetric-crypto concepts.
When stuck, a strong strategy is to:	**revisit what category/technique the clues point to** — Re-reading the clues to re-identify the technique often unblocks progress.
The synthesis of VaultForge is:	**cryptography and security are about protecting people — learn the attacks to build the defenses, ethically** — The through-line: understand offense to build defense, always ethically and for protection.
Good CTF practice is to:	**document each step so you can reproduce and learn from it** — Documenting steps aids reproduction, learning, and teamwork.

Keep going

You practiced **400 cards** from VaultForge. Come back to the ones you missed — spaced-out practice makes them stick.

Spark & Anvil is a 501(c)(3) public charity. Every app, story, and book is free, forever — no ads, no in-app purchases, no tracking.

Keep exploring online

Play it now	More free books
	
`https://spark-and-anvil.org/play/vaultforge`	`https://spark-and-anvil.org/books`

Scan a code with any phone camera, or type the address. Every app, story, and book is free, forever — no account, no tracking.