How to get your password back

Your password was split into 5 blocks. Any 3 blocks from the same set rebuild it exactly. One or two blocks reveal nothing. You don't need pass.new to do this: any of the ways below will work, even if pass.new no longer exists.

1. The easiest way: pass.new/split

Scan the QR codes on 3 blocks with a phone camera, or open the Recover tab and type the blocks in.

2. The offline recovery page

Download recover.html from pass.new/recovery (or the public copy of this kit on GitHub). Open it in any web browser, even with the internet switched off, and type in 3 blocks.

3. Any computer with Python 3

Download recover.py, or type in the program printed at the end of this guide, and run: python3 recover.py Paste the 3 blocks when asked. It needs nothing else installed.

4. An AI assistant

The next page is a ready-made prompt. It explains the exact format so any AI that can run Python can rebuild the password. Anything you paste into an online AI can be seen by the company that runs it, so for anything valuable use options 1 to 3 first, and move funds or change the password afterwards.

Good to know

- Type each block exactly as printed. Capitals, spaces and dashes don't matter, and O/0 and I/L/1 are treated the same.
- Every block shows a Set ID. All 3 blocks must have the same Set ID; blocks from different sets can't be mixed.
- If a block is damaged or has a typo, you'll be told which one. Use any other block instead: any 3 will do.
- The recovered password is checked against a fingerprint inside the blocks, so it is either exactly right or you get an error, never a slightly wrong password.
- Try the example blocks in test-blocks.txt first if you want to be sure a tool works.

Technical summary

Shamir's method, any 3 of 5, over GF(256) with the AES polynomial 0x11B. Block text is Crockford Base32. Block bytes: [version 1] [0x35] [block number 1-5] [4-byte set ID] [data] [first 4 bytes of SHA-256 of everything before]. Combined data: [2-byte length] [UTF-8 password] [first 4 bytes of SHA-256 of the password] [zero padding]. Full details are in the AI prompt on the next page.

AI prompt: rebuild the password with an AI assistant

```
PASS.NEW PASSWORD RECOVERY: A PROMPT FOR AN AI ASSISTANT

Read this first (for the person holding the blocks)
Anything you paste into an online AI can be seen by the company that runs it. If
this password protects money (such as a Bitcoin key) or anything important, try
the offline options first: pass.new/split, the offline recovery page (recover.html),
or recover.py on your own computer. If you do use an AI, move the funds or change
the password as soon as you have it back. The full kit, free and public domain, is
at https://github.com/pass-new/recovery-kit

How to use this prompt: copy everything below the dashed line into an AI assistant
that can run Python code. Replace the three "My block" lines at the end with your
own blocks, exactly as printed.
--------------------------------------------------------------------------------
I have 3 "blocks" made by pass.new. Any 3 of the 5 blocks in a set rebuild a
password exactly. Please write and run a Python 3 program (standard library only)
that follows this specification precisely. Don't guess or skip checks. If a check
fails, tell me which block has the problem.

1. Reading a block. Blocks are Crockford Base32 text. Alphabet:
   0123456789ABCDEFGHJKMNPQRSTVWXYZ
   Make the text uppercase. Ignore spaces, dashes, dots and line breaks. Read the
   letter O as 0, and I or L as 1. If the text contains "r=", use only the part
   after "r=". Turn each character into 5 bits (most significant bit first), join
   them, and cut them into 8-bit bytes. Throw away leftover bits at the end.

2. Checking a block. The last 4 bytes are a checksum: they must equal the first
   4 bytes of SHA-256 of all the bytes before them. Then:
   byte 0 = version, must be 1
   byte 1 = 0x35 (any 3 of 5)
   byte 2 = the block number x, from 1 to 5
   bytes 3 to 6 = set ID, which must be the same on all 3 blocks
   bytes 7 up to (but not including) the checksum = the block data y

3. Combining. Use Shamir's method over GF(256) with the AES polynomial 0x11B
   (x^8 + x^4 + x^3 + x + 1). In this field, adding is XOR. For each byte
   position k, the result is the XOR over the 3 blocks i of  y_i[k] * L_i,
   where L_i is the product over the other two blocks j of  x_j / (x_j XOR x_i),
   using GF(256) multiplication and division.

4. Unpacking. The combined bytes are: 2 bytes big-endian length n, then n bytes of
   UTF-8 text (the password), then 4 bytes that must equal the first 4 bytes of
   SHA-256 of those n password bytes, then zero padding. If the 4 bytes match,
   the password is exactly those n bytes decoded as UTF-8, including any spaces
   at the start or end. If they don't match, something is wrong: don't return a
   result.

5. First test your program on these blocks, which must give back exactly
   "correct horse battery staple" (28 characters):
   Test block 1:
     04TG 30ZE P24M 075Y T4RJ PV13 ANND RTHW ZB54 0RKP KHG5 R08E
     B110 57AQ GWQD GZB9 FXWY XY27 02TZ F1P6 ZRAD GYCG WAYT 8QG
   Test block 3:
     04TG 70ZE P24W 6HC1 FKMG G4QD MSXK GTYE TP7S GNAY 6J1T H61T
     3TJS ADNF CNZK JKBC F6T7 XSQG 53WT QCPS TJ9R J174 TBB2 VZ0
   Test block 5:
     04TG B0ZE P24N 3MCE JQWH YEW7 B9N6 AH30 7H08 QM1Y 82Y8 3KS3
     7ATW 1TTJ WJYM HDRG 8R29 4FEP Y3CX TRQE PE5V Z2AZ J742 RSR

6. Then run it on my blocks below, and show me the password on its own line,
   exactly, with its length in characters.

My block 1: (type your first block here)
My block 2: (type your second block here)
My block 3: (type your third block here)
```

recover.py: the recovery program

Type this into a file called recover.py (spaces at the start of lines matter), then run: python3 recover.py

```
#!/usr/bin/env python3
"""pass.new recovery: rebuild a password from any 3 of its 5 blocks.

Works offline with plain Python 3 (no installs). Run it, then type or paste 3 blocks:
    python3 recover.py
or pass them as arguments:
    python3 recover.py "BLOCK ONE" "BLOCK TWO" "BLOCK THREE"
Public domain. Format details: see README.md in this kit.
"""
import hashlib, itertools, sys

# Block text is Crockford Base32: no I, L, O or U. Case, spaces and dashes don't matter.
ALPHABET = "0123456789ABCDEFGHJKMNPQRSTVWXYZ"
VALUE = {c: i for i, c in enumerate(ALPHABET)}
VALUE.update({"O": 0, "I": 1, "L": 1})

def read_block(text):
    """Block text -> (block number 1-5, set id, data bytes). Checks the block's checksum."""
    if "r=" in text:                      # a scanned QR link: .../split#r=BLOCK
        text = text.split("r=", 1)[1]
    buffer = bits = 0
    raw = bytearray()
    for ch in text.upper():
        if ch in " -_.\t\r\n":
            continue
        if ch not in VALUE:
            raise ValueError(f"'{ch}' is not a character used in blocks")
        buffer = (buffer << 5) | VALUE[ch]
        bits += 5
        if bits >= 8:
            bits -= 8
            raw.append((buffer >> bits) & 0xFF)
        buffer &= (1 << bits) - 1
    body, check = bytes(raw[:-4]), bytes(raw[-4:])
    if len(body) < 8 or hashlib.sha256(body).digest()[:4] != check:
        raise ValueError("this block doesn't check out (probably a typo or a missing character)")
    if body[0] != 1 or body[1] != 0x35:  # version 1, "any 3 of 5"
        raise ValueError("this isn't a pass.new version 1 block")
    return body[2], body[3:7].hex().upper(), body[7:]

# Arithmetic in GF(256) with the AES polynomial x^8 + x^4 + x^3 + x + 1 (0x11B).
EXP, LOG = [0] * 510, [0] * 256
x = 1
for i in range(255):
    EXP[i], LOG[x] = x, i
    x ^= ((x << 1) ^ (0x1B if x & 0x80 else 0)) & 0xFF   # multiply by 3
for i in range(255, 510):
    EXP[i] = EXP[i - 255]

def mul(a, b):
    return 0 if a == 0 or b == 0 else EXP[LOG[a] + LOG[b]]

def div(a, b):
    return 0 if a == 0 else EXP[(LOG[a] + 255 - LOG[b]) % 255]

def combine(blocks):
    """Lagrange interpolation at x = 0, byte by byte (Shamir's method, 3 of 5)."""
    out = bytearray(len(blocks[0][2]))
    for i, (xi, _, yi) in enumerate(blocks):
        weight = 1
        for j, (xj, _, _) in enumerate(blocks):
            if i != j:
                weight = mul(weight, div(xj, xj ^ xi))
        for k, byte in enumerate(yi):
            out[k] ^= mul(byte, weight)
    return bytes(out)

def unpack(data):
    """[2-byte length] + UTF-8 password + first 4 bytes of SHA-256(password) + zero padding."""
    n = (data[0] << 8) | data[1]
    password, fingerprint = data[2:2 + n], data[2 + n:6 + n]
    if n == 0 or len(password) != n or hashlib.sha256(password).digest()[:4] != fingerprint:
```

recover.py (continued)

```
        return None
    return password.decode("utf-8")

def recover(texts):
    blocks = {}
    for text in texts:
        number, set_id, data = read_block(text)
        blocks[number] = (number, set_id, data)
    if len({b[1] for b in blocks.values()}) > 1:
        raise ValueError("these blocks come from different sets (the set IDs don't match)")
    if len(blocks) < 3:
        raise ValueError(f"you need 3 different blocks; you have {len(blocks)}")
    for three in itertools.combinations(blocks.values(), 3):
        password = unpack(combine(list(three)))
        if password is not None:
            return password
    raise ValueError("these blocks don't fit together; one may be damaged or from another set")

if __name__ == "__main__":
    texts = sys.argv[1:]
    while len(texts) < 3:
        try:
            line = input(f"Block {len(texts) + 1} of 3 (paste, then press Enter): ").strip()
        except EOFError:
            break
        if line:
            try:
                read_block(line)
                texts.append(line)
            except ValueError as err:
                print(f"  Problem: {err}. Please try that block again.")
    try:
        password = recover(texts)
    except ValueError as err:
        sys.exit(f"Could not recover the password: {err}")
    print("\nYour password is between the lines below (spaces at either end are part of it):")
    print("-" * 40)
    print(password)
    print("-" * 40)
    print(f"{len(password)} characters, verified as an exact match.")
```