COMP 8005
Assignment 3

Design

Daryush Balsara
A01265967
Oct, 25 2025

Purpose

Create a password cracker that works with threading and can crack all 5 algos

cracker.py

Functions

Name	Description	Return value	Reason
_load_ctypes_crypt()	Attempts to load a system crypt() implementation via ctypes from several candidate library paths. Caches and returns the function pointer.	c_callable or None	Provides the low-level interface needed to verify yescrypt (and other crypt-based) hashes when passlib lacks a handler.
verify_yescrypt(candidate, target_hash)	Verifies a single candidate string against a yescrypt-formatted hash by calling the loaded crypt() function with an extracted salt.	bool	Required because passlib may not provide a yescrypt handler; this enables one-off verification via system crypt.

signal_handler(sig, frame)	Signal handler that sets a global stop event to request graceful shutdown.	None	Allows user to interrupt (Ctrl+C) and lets threads cleanly stop.
parse_shadow_entry(line, username)	Parses a single /etc/shadow-style line and returns the username and password field if it matches the requested username.	(username, pwdfield) or None	Extracts the stored hash for the target user; ignores malformed or different-user lines.
AtomicCounter.__init__(initial=0)	Creates a thread-safe counter with an internal lock.	AtomicCounter instance	Used for accurate progress accounting across threads without races.
AtomicCounter.increment(n=1)	Atomically increments the counter by n.	int (new value)	Thread-safe update used by workers to report progress.
AtomicCounter.value()	Returns the current counter value in a thread-safe manner.	int	Used when printing progress or final statistics.

batch_producer(out_q, batch_size, min_length, max_length, num_threads)	Generates candidate passwords (cartesian product of CHARSET) grouped into lists of batch_size, pushes them onto out_q, and finally places sentinel None values for workers to exit.	None (pushes batches into out_q)	Produces work for worker threads while allowing graceful termination.
crack_worker(task_q, ctx, target_hash, result_q, progress_counter, progress_report_batch=128)	Worker thread function: pulls batches from task_q, verifies each password (via verify_yescrypt for yescrypt or ctx.verify for passlib-supported schemes), reports progress via progress_counter, and places a found password in result_q.	None (may put found pwd into result_q)	Core verification loop; multithreaded to parallelize verification (within limits).

main(shadow_path, target_username, num_threads)	Orchestrates reading the shadow file, extracting the target hash, setting up queues and threads, starting producer and workers, monitoring for results or interrupts, and printing final status.	None (prints results or errors, exits)	Entrypoint that composes all components and handles lifecycle, timing, and output.

Variables

Name	Type	Description
CHARSET	str	All characters used to generate candidate passwords (A-Z a-z 0-9 and symbols).
MIN_LENGTH	int	Minimum password length to generate (script sets 2).
MAX_LENGTH	int	Maximum password length to generate (script sets 4).
BATCH_SIZE	int	Number of candidate passwords grouped in one batch sent to worker threads.
PROGRESS_INTERVAL	int	Interval at which a progress message is printed (approx).

DESIRED_SCHEMES	List[str]	Passlib schemes the script prefers to support (sha512, sha256, md5, bcrypt, yescrypt).
SUPPORTED	List[str]	Schemes actually found in the environment via registry.get_crypt_handler.
ctx	CryptContext	Passlib context initialized with the supported schemes for ctx.verify().
_ctypes_crypt_fn	callable or None	Cached pointer to system crypt() function loaded via ctypes, or None.
stop_event	threading.Event	Global event used to request shutdown of producer/workers.
task_q	queue.Queue	Queue holding batches of candidate passwords produced by batch_producer.
result_q	queue.Queue	Queue for a found plaintext password (size 1).
progress_counter	AtomicCounter	Thread-safe counter tracking how many candidates have been processed.
local_count	int	Worker-local counter used to reduce contention on shared progress counter.
use_yes	bool	Worker-local flag indicating whether the target hash is yescrypt (True) or not.
salt_parts, salt	List[str], str	Pieces extracted from the stored hash used as the salt when calling crypt().
producer	threading.Thread	The thread running batch_producer.

workers	List[threading.Thread]	List of worker threads running crack_worker.
found	str or None	The discovered plaintext password if found, otherwise None.
start	float	Timestamp when cracking started (used to compute elapsed time).
elapsed	float	Computed runtime in seconds (end - start).
e	Exception	Generic exception variable used in except blocks for reporting.

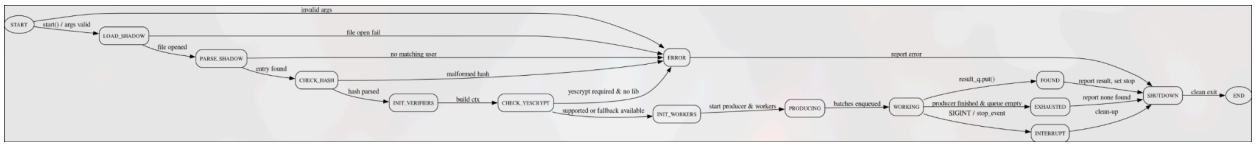
Pseudo Code

```
function _load_ctypes_crypt():
    if _ctypes_crypt_fn is cached:
        return cached function
    for libname in candidate_library_paths:
        try:
            lib = CDLL(libname)
            set lib.crypt return/arg types
            cache lib.crypt as _ctypes_crypt_fn
            return _ctypes_crypt_fn
        except Exception:
            continue
    cache None and return None

function verify_yescrypt(candidate: str, target_hash: str) -> bool:
    if inputs are not strings: return False
    if target_hash does not look like a yescrypt hash (e.g. not starting with "$y$"): return False
    crypt_fn = _load_ctypes_crypt()
    if crypt_fn is None:
        log error and return False
    extract appropriate salt portion from target_hash
    call crypt_fn(candidate_bytes, salt_bytes) -> out
    if out is None: return False
    decode out to string
    return (out_str == target_hash)
```

```
function parse_shadow_entry(line: str, username: str) -> (username, hash) | None:
    split line into parts by ":" up to 3 fields
    if parts length < 2: return None
    if parts[0] != username: return None
    pwdfield = parts[1]
    if pwdfield is empty: return None
    return (parts[0], pwdfield)

# -- atomic counter for progress reporting --
class AtomicCounter:
    constructor(initial=0):
        _val = initial
        _lock = new Lock()
    method increment(n=1):
        with _lock: _val += n; return _val
    method value():
        with _lock: return _val

function batch_producer(out_q, batch_size, min_length, max_length, num_threads):
    try:
        for L in range(min_length..max_length):
            if stop_event set: break
            for each combination in product(CHARSET, repeat=L):
                if stop_event set: break
                append combination to batch
                if batch size >= batch_size:
                    out_q.put(batch)
                    reset batch to []
            if batch not empty: out_q.put(batch)
    finally:
        # always send sentinel None to workers so they can exit
        for i in 1..num_threads:
            out_q.put(None)

function crack_worker(task_q, ctx, target_hash, result_q, progress_counter,
progress_report_batch):
    local_count = 0
    use_yes = target_hash starts with yescrypt marker
    while not stop_event:
        batch = task_q.get()
        if batch is None: break
        for pwd in batch:
            if stop_event: break
            try:
```

```
            if use_yes:
                ok = verify_yescrypt(pwd, target_hash)
            else:
                ok = ctx.verify(pwd, target_hash)
            if ok:
                result_q.put(pwd)
                set stop_event
                return
        except:
            ignore verification exception
        local_count += 1
        if local_count >= progress_report_batch:
            progress_counter.increment(local_count)
            local_count = 0
            if progress_counter.value() % PROGRESS_INTERVAL < progress_report_batch:
                print progress message
    if local_count > 0:
        progress_counter.increment(local_count)

function main(shadow_path, target_username, num_threads):
    install signal handler to set stop_event on SIGINT
    validate num_threads as positive integer
    open shadow_path and read lines, on error exit
    entry = first parse_shadow_entry(line, target_username) found
    if entry is None: error and exit
    username, target_hash = entry
    optionally set CPU affinity if available (best-effort)
    # prepare queues, counters, and worker threads
    task_q = new Queue(maxsize=num_threads*2)
    result_q = new Queue(maxsize=1)
    progress_counter = AtomicCounter(0)
    spawn num_threads threads running crack_worker(...)
    spawn producer thread running batch_producer(task_q, ...)
    start timing
    loop until workers finished or stop_event:
        try to get found password from result_q with timeout
        if found: print found, timing and processed count; set stop_event; break
        sleep briefly
    on KeyboardInterrupt: set stop_event
    join producer (short timeout), join workers (short timeout)
    if not found: print no password found, timing and processed count
```

FSM


START
LOAD_SHADOW
PARSE_SHADOW
CHECK_HASH
INIT_VERIFIERS
CHECK_YESCRYPT
INIT_WORKERS
PRODUCING
WORKING
FOUND
EXHAUSTED
INTERRUPT
SHUTDOWN
END
ERROR
invalid args
file open fail
no matching user
malformed hash
report error
result_q put()
report result, set stop
report none found
clean-up
clean exit
yescrypt required & no lib
start producer & workers
batches enqueued
producer finished & queue empty
SIGINT / stop_event