Advanced Secure Protocol Design — Peer Code Review

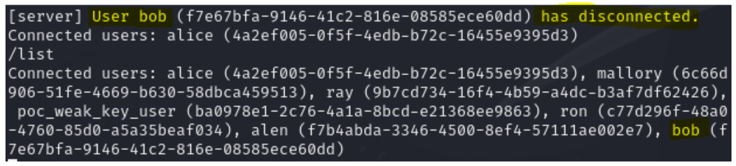
Group Reviewed: ?

Setup & Environment

	Check	Notes
☑	README includes installation & run instructions	Used README.md but there are 4 different README files?
☑	Dependencies listed clearly	
☑	Program runs without crashes or unhandled exceptions	

Protocol Compliance & Functionality

	Check	Notes

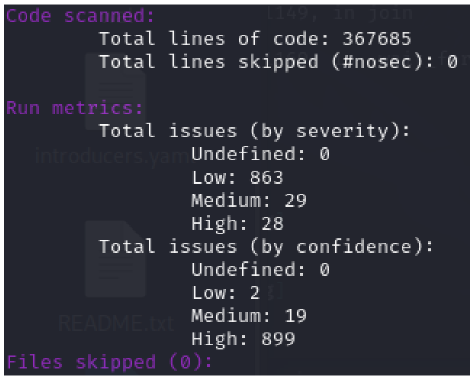
☑	Lists all online members	Lists all members who have ever connected  `[server] User bob (f7e67bfa-9146-41c2-816e-08585ece60dd) has disconnected.`
		`Connected users: alice (4a2ef005-0f5f-4edb-b72c-16455e9395d3)`
		`/list`
		`Connected users: alice (4a2ef005-0f5f-4edb-b72c-16455e9395d3), mallory (6c66d`
		`906-51fe-4669-b630-58dbca459513), ray (9b7cd734-16f4-4b59-a4dc-b3af7df62426),`
		` poc_weak_key_user (ba0978e1-2c76-4a1a-8bcd-e21368ee9863), ron (c77d296f-48a0`
		`-4760-85d0-a5a35beaf034), alen (f7b4abda-3346-4500-8ef4-57111ae002e7), bob (f`
		`7e67bfa-9146-41c2-816e-08585ece60dd)`
		I like how all peers are informed when someone connects

☑	Sends/receives private messages correctly	
☑	Sends/receives group messages correctly	
☐	Supports secure file transfer	Not yet implemented
☑	User experience	Nice clean chat terminals. My only thought is that maybe only the usernames should be displayed, not the IDs.

Security Implementation

	Check	Notes
☑	All communications encrypted	Messages encrypted - tested with wireshark
☑	Keys generated securely	
☐	Authentication implemented correctly	No authentication
☑	Input validation and sanitisation present	
☑	No sensitive info leaked in errors/logs	
☑	Secure socket handling (no plaintext fallback)	
☑	Cryptographically secure randomness used	
☑	No plaintext passwords/keys stored or transmitted	
☑	Cert/key validation	
☐	Replay/MITM resistance	If Mallory connects to server A, and Bob tries to send Alice a message, the message will not be sent. Testing this did break it however, as no more private messages were able to be sent to and from Bob and Alice. Group messages unaffected.

Ran bandit on all files:



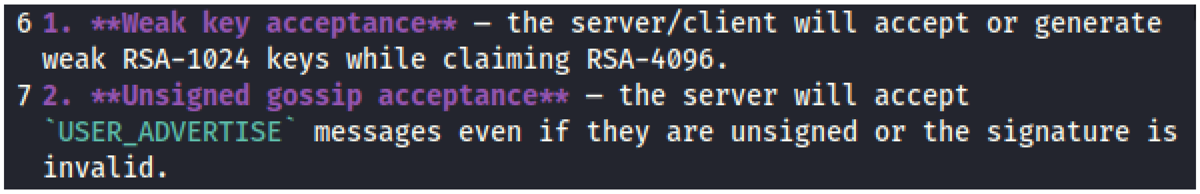
```
Code scanned:
        Total lines of code: 367685
        Total lines skipped (#nosec): 0

Run metrics:
        Total issues (by severity):
                Undefined: 0
                Low: 863
                Medium: 29
                High: 28
        Total issues (by confidence):
                Undefined: 0
                Low: 2
                Medium: 19
                High: 899
Files skipped (0):
```

When choosing external packages to use, consider the security implications.

- server.py: 10 x Issue: [B110:try_except_pass] Try, Except, Pass detected.
- datavault.py: Issue: [B608:hardcoded_sql_expressions] Possible SQL injection vector through string-based query construction.
    - This is safe in this case but be wary of using f-strings for dynamic SQL

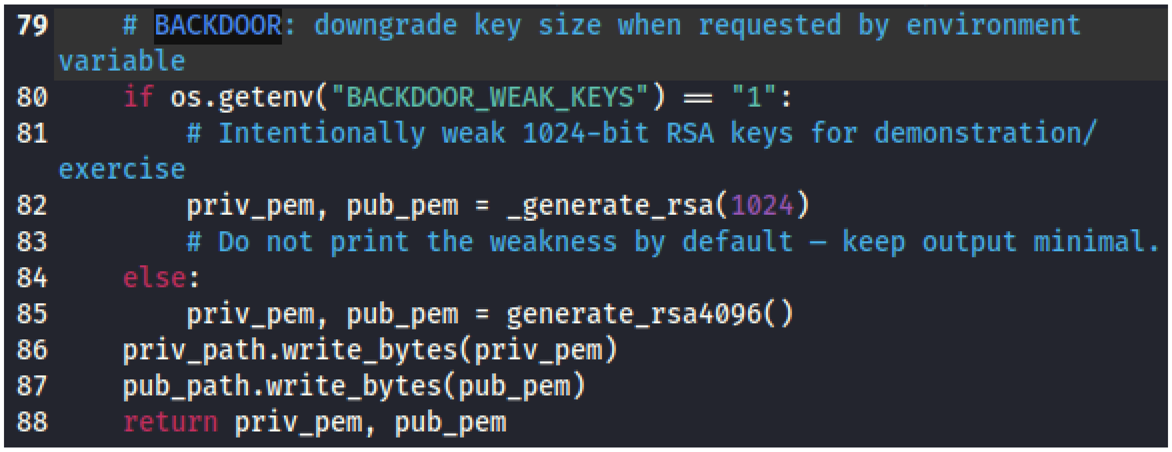
Backdoors

BACKDOOR_README.md contains all details of intentional backdoors.



```
1. **Weak key acceptance** — the server/client will accept or generate weak RSA-1024 keys while claiming RSA-4096.
2. **Unsigned gossip acceptance** — the server will accept `USER_ADVERTISE` messages even if they are unsigned or the signature is invalid.
```

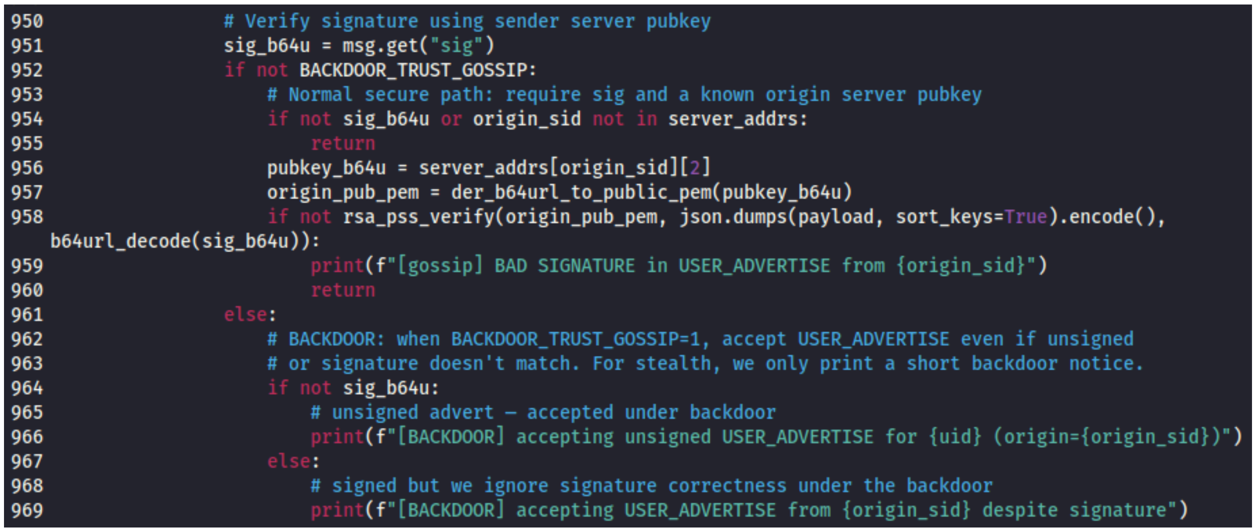
1.



```python
    # BACKDOOR: downgrade key size when requested by environment variable
    if os.getenv("BACKDOOR_WEAK_KEYS") == "1":
        # Intentionally weak 1024-bit RSA keys for demonstration/exercise
        priv_pem, pub_pem = _generate_rsa(1024)
        # Do not print the weakness by default — keep output minimal.
    else:
        priv_pem, pub_pem = generate_rsa4096()
    priv_path.write_bytes(priv_pem)
    pub_path.write_bytes(pub_pem)
    return priv_pem, pub_pem
```

Weak RSA key used, the key size is too small, making it vulnerable to attacks.

2.



```python
                # Verify signature using sender server pubkey
                sig_b64u = msg.get("sig")
                if not BACKDOOR_TRUST_GOSSIP:
                    # Normal secure path: require sig and a known origin server pubkey
                    if not sig_b64u or origin_sid not in server_addrs:
                        return
                    pubkey_b64u = server_addrs[origin_sid][2]
                    origin_pub_pem = der_b64url_to_public_pem(pubkey_b64u)
                    if not rsa_pss_verify(origin_pub_pem, json.dumps(payload, sort_keys=True).encode(), b64url_decode(sig_b64u)):
                        print(f"[gossip] BAD SIGNATURE in USER_ADVERTISE from {origin_sid}")
                        return
                else:
                    # BACKDOOR: when BACKDOOR_TRUST_GOSSIP=1, accept USER_ADVERTISE even if unsigned
                    # or signature doesn't match. For stealth, we only print a short backdoor notice.
                    if not sig_b64u:
                        # unsigned advert — accepted under backdoor
                        print(f"[BACKDOOR] accepting unsigned USER_ADVERTISE for {uid} (origin={origin_sid})")
                    else:
                        # signed but we ignore signature correctness under the backdoor
                        print(f"[BACKDOOR] accepting USER_ADVERTISE from {origin_sid} despite signature")
```

Signatures not verified, an attacker can pretend to be someone else without worrying about a signature for authorisation.