OTP-Reuse Lab

EMCC Cybersecurity & Coding Club — Ethical Hacking Practice

What This Lab Teaches

- Understand OTP reuse and crib-dragging.
- Hands-on plaintext recovery with lab ciphertexts.
- Connect to real-world crypto failures.

Club Ethics & Safety

- Only analyze provided ciphertexts.
- Do not intercept or test real systems.
- Focus on learning and defense.

Why This Happens

When two messages reuse the same OTP key bytes:

$ct1 = pt1 \oplus k$

$ct2 = pt2 \oplus k$

Then $ct1 \oplus ct2 = pt1 \oplus pt2$ — the key cancels. Crib-dragging uses guessed fragments to recover plaintext by XORing guessed plaintext with the XOR of ciphertexts.

Real-World Examples

- Historical: reused OTP pads exposed secret communications.
- WEP: RC4 IV reuse broke Wi-Fi encryption.
- AES-GCM: nonce reuse can allow message forgeries.
- ECDSA: reused nonces leak private keys.
- IoT: identical device keys in production.

Hands-On Steps (In Depth)

Step 1 — Open the Lab

Open the GitHub repo and click 'Open in Colab' or open otp_reuse_lab.ipynb in Jupyter. Run all cells to generate ciphertexts (c0, c1, ...). Functions available: xor_bytes, crib_drag, automated_crib_search, show_recovered_fragment, is_printable.

Step 2 — Compute XOR

Choose two ciphertexts (e.g., c0 and c1) and compute their XOR. This gives $pt0 \oplus pt1$, which you'll analyze.

Code

```
x = xor_bytes(cts[0], cts[1]) print('len(x)=', len(x)) print(x[:120].hex())
```

Step 3 — Manual Crib-Dragging

Pick a crib (a short guessed plaintext) and slide it across the XOR result to find readable text.

Code

```
crib = b'Meet ' for off in range(0, len(x)-len(crib)+1):
    cand = crib_drag(x, crib, off)     if is_printable(cand, threshold=0.95):
        print('offset', off, '->', cand)
```

Step 4 — Verify Guess

When you find readable output, derive key fragment and use it to decrypt another ciphertext at same offset. This confirms your guess.

Code

```
k_frag = xor_bytes(cts[0][offset:offset+len(crib)], crib)
recovered = xor_bytes(cts[1][offset:offset+len(crib)], k_frag)
print('recovered:', recovered)
```

Step 5 — Cross-Apply & Tile Fragments

Use show_recovered_fragment(crib) to check other ciphertexts. Collect verified fragments with offsets and assemble them by byte index to rebuild longer text.

Step 6 — Automated Crib Search

Run automated_crib_search(0,1, common_bytes, show_hits=30). Treat these as leads — always verify manually.

Step 7 — Document Findings

Record: which ciphertext pair you analyzed, the crib(s) used, offsets, recovered fragments (raw bytes and decoded), and verification outputs (key fragment hex).

Tips

- Use cribs with spaces for alignment.

- Longer cribs reduce false positives.

- If no hits, try a different ciphertext pair.

- Decode weird bytes with latin1.

How Defenders Stop This

- Never reuse keys, IVs, or nonces.

- Use CSPRNGs and unique per-message keys.

- Enforce unique nonces in AEAD (AES-GCM, ChaCha20-Poly1305).

- Use hardware key stores for critical secrets.

- Monitor for duplicate IVs or key blobs.

Discussion Prompts

- Why is this an operational failure and not a math one?

- Which modern systems might fail similarly?

- How would you detect reuse in logs or telemetry?

Key Takeaway

Encryption is only as strong as the randomness and discipline behind it. Reuse turns strong math into readable text.

Prepared for EMCC Cybersecurity & Coding Club — Peer-led Learning

Eastern Maine Community College