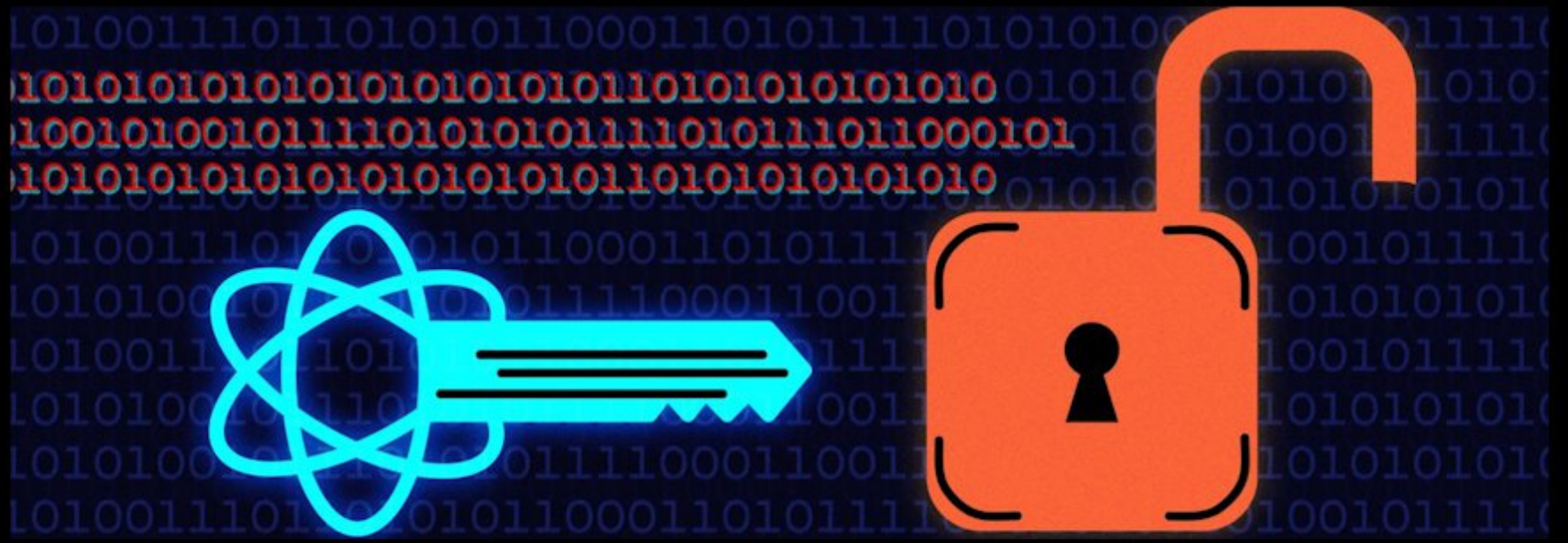
1UP
1337

HI-SCORE
31337



PQC FOR HACKERS

EIJAH

Welcome

Hack the Planet!

Introduction

- Eijah
- Atari Vampire
- Eric Anderson

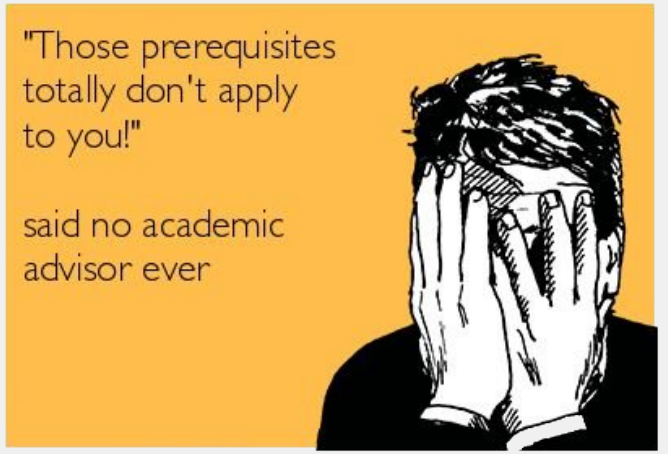




0xAA856A1BA814AB99FFDEBA6AEFBE1C04

Workshop Prerequisites

- This is an intermediate workshop and requires that you have some prior experience programming on a Linux environment with GCC, CMake, and C/C++. Basic knowledge of cryptography, OpenSSL and a familiarly with the Linux operating system is also helpful.
- Verify that you have the following:
  - An x86_64 laptop
  - The latest version of Virtual Box installed
  - The PQC for Hackers workshop files
    - Linux Virtual Image
    - PDF presentation
- If you don't have the workshop files, we have copies available on USB.
- You can also connect to the DEF CON network and download from https://codesiren.com/defcon34
  - This is the **only** time at DEF CON that you can trust a USB
  - The website also has instructions for alternative deployments
    - NOTE! The code was compiled and tested on Debian 13.6.0
- My colleagues will help you get setup at this time



Workshop Goals

- Understand why Post-Quantum Cryptography (PQC) matters
- Implement all of the PQC algorithms specified in the NSA's Commercial National Security Algorithm (CNSA) Suite 2.0 specification using C++ and OpenSSL
- Understand the pros/cons of PQC algorithms:
  - Crypto agility
  - Large key/signature sizes
  - Performance overhead
  - Implementation complexity
  - Novelty of Lattice-based algorithms
- Leave the Workshop with working code knowledge
- Figure out new ways to use the code
- Meet some new friends and have fun!

Workshop Options

- Although this is an intermediate workshop, it's also designed to be very flexible because not everyone who's attending has the same background, experience, tolerance, and skills.
- You can progress through the workshop in a variety of ways:
  - You can take a passive approach with a focus on listening, learning, and understanding the course material
  - You can compile and run the solutions as opposed to writing the code yourself
  - You can follow along with me as I code each solution real-time
  - You can code all solutions yourself and go totally Rambo
  - Or any combination thereof
- All the code (3rd parties, engine, examples) will be available for you to take home and modify after the workshop

Build Pipeline

- Debian 13.6.0
  - Foundational kernel for many Linux-based distros
- GCC
  - Who doesn't love GCC?
- CMake
  - Ease of use, standard
- C++ 20
  - Performance, control, interoperability, modern
- Boost
  - Leading 3rd-party C++ library with many APIs adopted in C++ versions
- OpenSSL 3.5.2
  - Open Source, De-facto standard, PQC early adopter, multi-platform
- VS Code
  - One of the best free IDEs available
- Virtual Box
  - Free, general virtualization software

Lesson -1

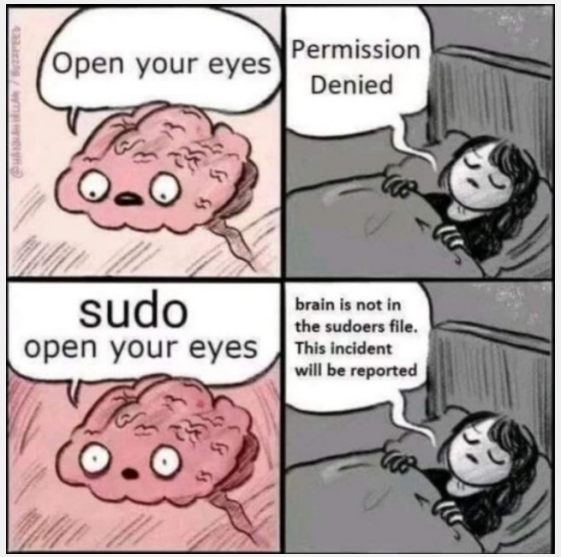
Setup & Configuration

Lesson Goals

- Mount the Linux virtual image
- Login

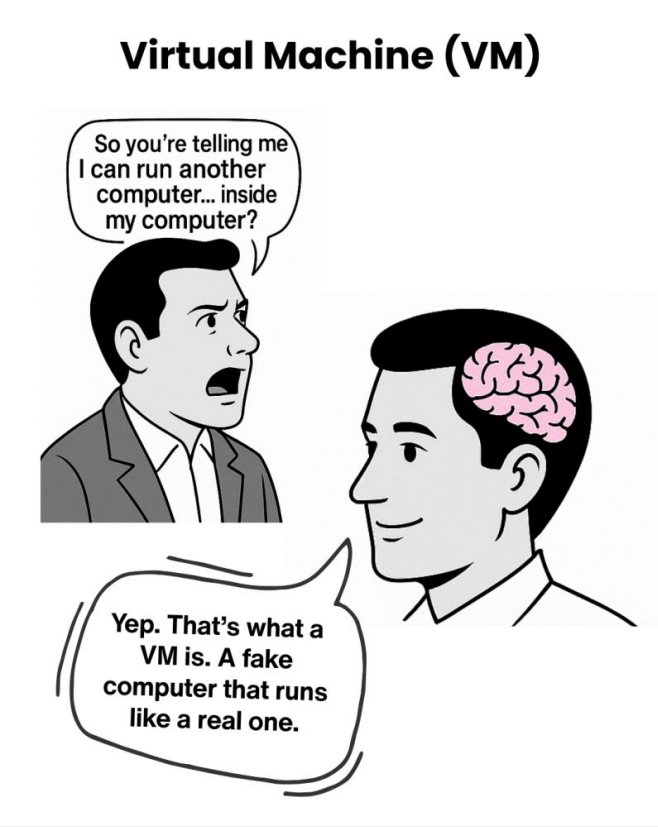
Linux Virtual Machine (VirtualBox)

- Locate the pqc_vm.zip file on your machine and extract it
- Open VirtualBox and navigate to File -> Preferences
- Change to advanced (if not already set) and click OK
- Go to File -> Import Appliance and select the pqc.ova as the source
- On the import dialogue that appears, you can expand Settings and modify the CPU and RAM if needed for your machine
  - CPU is set for 4 cores and 8192 MB of RAM by default for a better experience, but you can change if needed
- Click Finish



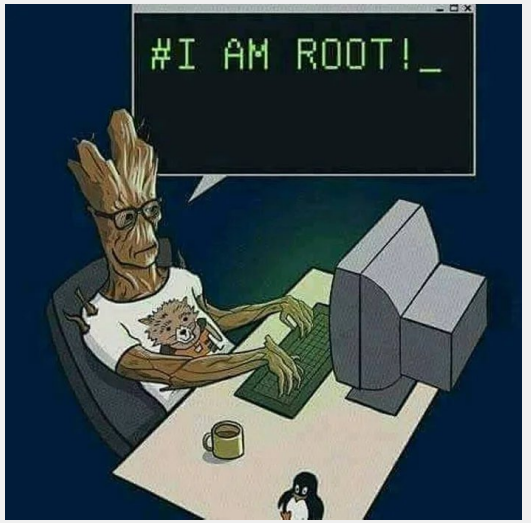
Linux Virtual Machine (VirtualBox)

- Start the “PQC for Hackers” virtual machine
- Login with:
  - User: user
  - Password: defcon34
  - User has been given sudo access
  - The root account has the same password
- Open a command prompt
- Navigate to /defcon_34
  - Explore the workshop files
- Open VS Code (shortcut on the desktop)
  - Explore the workshop files



Linux Virtual Machine (Other)

- This workshop is designed around the x86_64 instruction set. In the event that you don't have an x86_64 laptop, you have a few options.
  1. If using Mac, you can use UTM to emulate x86-64.
     - Follow the detailed instructions in README.txt
     - Note that emulation will be slow - this is normal
  2. If using Linux Native and you have a Debian-based distro, you can attempt to compile the code manually.
     1. Follow the detailed instructions in README.txt
     2. Linux distros can be a bit tricky, so this may or may not work.
  3. You can participate in the workshop passively and compile the code on your own at a later time.
- My colleagues will help you get setup at this time



Lesson 0

The Quantum Leap

Cryptographic Primitives

- Confidentiality
  - Protect information from being disclosed to unauthorized parties. Encryption.
- Authenticity
  - Ensure information came from a trusted source. Digital Certificates.
- Integrity
  - Protect information from being altered by unauthorized parties. Digital signatures, message authentication codes, and data hashing.
- Non-repudiation
  - Unwavering accountability. Ensures verifiable proof of message origin and transmission. Provable in a Court of Law. Digital signatures.

Traditional Cryptography

- Our entire society uses traditional public-key encryption
  - Rivest–Shamir–Adleman (RSA)
  - Elliptic Curve (EC)
  - Finite Field Diffie-Hellman key exchange
- The math…
  - Discrete Logarithm problem
  - The assumption that factoring large integers is computationally intractable
  - Not a problem with the current state of computing
  - But it becomes a problem with Quantum Computers
- What would happen if all of that suddenly collapsed?
  - All cryptography was broken overnight
  - No information could be encrypted

Quantum Computing

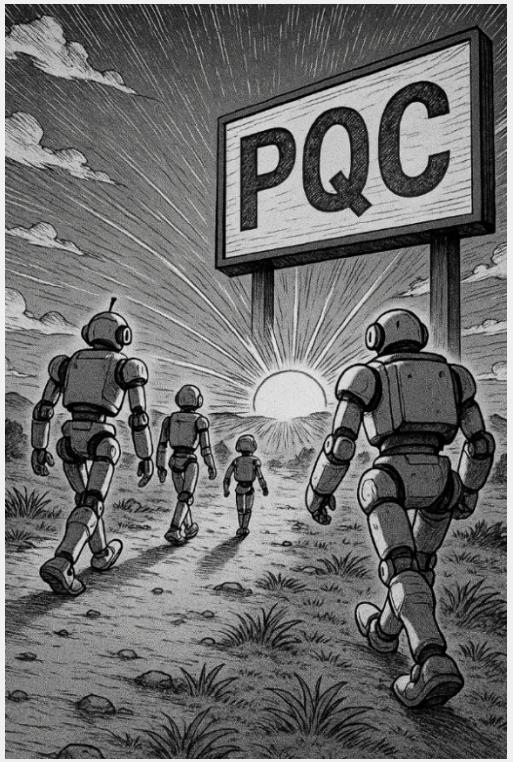
- What are Quantum Computers?
  - Quantum Entanglement and the Speed of Light
    - “Spooky Action at a Distance” - Einstein
  - Qubits can exist in multiple states at once (0, 1, or both)
  - Can explore multiple possibilities simultaneously (superposition)
  - Harness quantum entanglement for interconnected processing
  - Speed potential exponentially greater than classical computers.
- Quantum Algorithms
  - Shor's algorithm (1995 by Peter Shor)
    - Quantum algorithm that finds prime factors of an integer
  - Grover's algorithm (1996 by Lov Grover)
    - Quantum search algorithm that finds unique input to a black box function that produces a particular output (solves the task of function inversion)

Quantum Threat

- What is the Quantum Apocalypse?
  - The birth of Quantum Enlightenment
  - Event Horizon where all modern-day encryption will be broken
  - What would happen if all of that was suddenly broken overnight?
  - Harvest-now-decrypt-later (Store-now-decrypt-later)
- Modus Operandi of an adversary who achieves Quantum Enlightenment first?
  - Burn it all down
    - The hacker way. Fun, but unlikely.
  - Say nothing and decrypt whatever they want for years to come
    - Boring, but likely

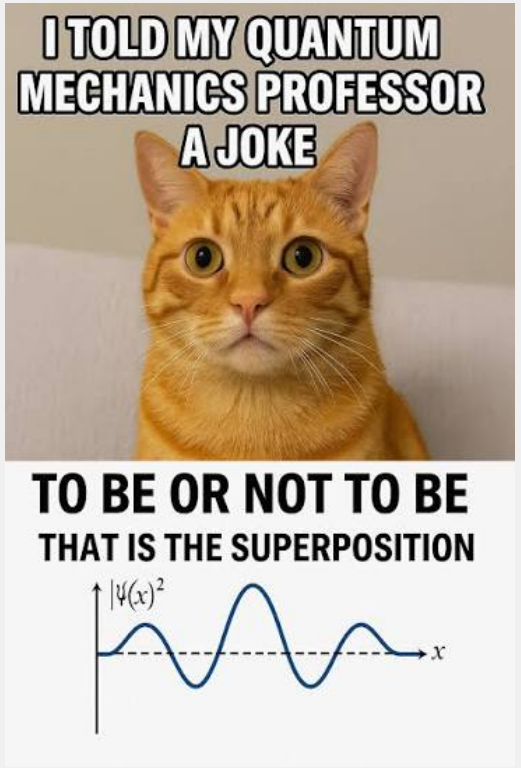
Post-Quantum Cryptography (PQC)

- The development of algorithms that are thought to be secure against quantum computers
- Certifications
  - NIST CAVP algorithms
  - NIST CMVP FIPS 140-3
- Commercial National Security Algorithm (CNSA) Suite 2.0
  - Cipher: AES-256 in GCM mode
  - Hash: SHA-384/512
  - Key Agreement: ML-KEM-1024 (FIPS 203)
  - Signatures: ML-DSA-87 (FIPS 204)
  - Firmware/software signing: LMS/XMSS (NIST SP 800-208)



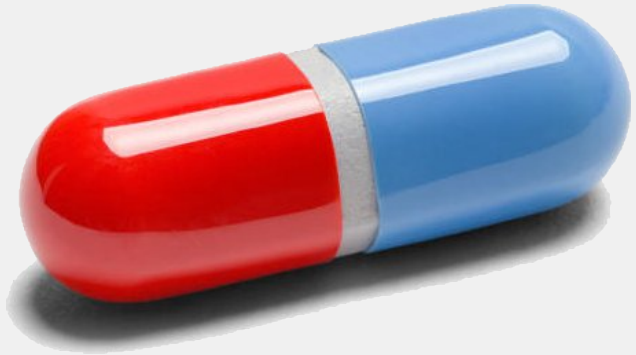
Post-Quantum Cryptography (PQC)

- The Challenges
  - Crypto agility
  - Storage overhead (key, signature sizes)
  - Performance overheads (keygen, keysig, keyver)
  - Implementation complexity (new and legacy apps)
  - Crypto agility and backwards compatibility (Lattice-based)
  - Unproven technology
- Is it worth the cost?
  - Doing nothing
  - Doing something



Lesson 1

Hello, Hacker World!

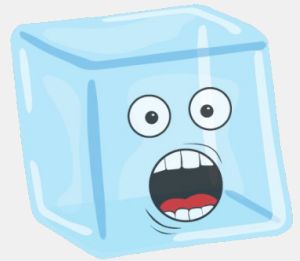


Lesson Overview

- Learn about the Workshop code base
- Verify that you have your build environment setup correctly
- Print out “Hello, Hacker PQC World!” to the screen
- Use a variety of tools
  - Debian Linux 3.16.0
  - C++ 20
  - Boost Libraries
  - The GNU Compiler
  - CMake & Ninja build systems
  - OpenSSL (Includes PQC algorithms as of 3.5.0)

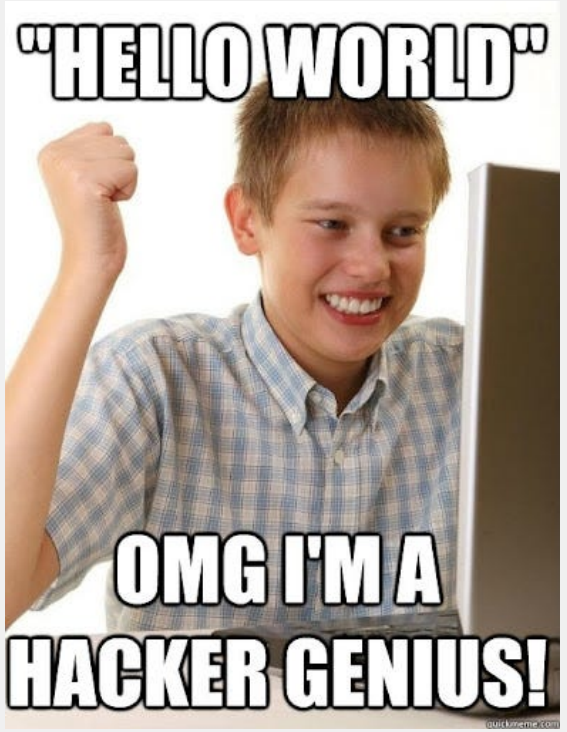
Lesson Format

- In each lesson folder there is a main_1.cpp and solution_1.cpp file
- The main_1.cpp file is used to create your code in
- The solution_1.cpp is a fully implemented solution that you can reference for help or use instead of writing your own code
- If you want to always build the solution, you can optionally set a variable for this
  - To set, open /defcon_34/code/CMakeLists.txt
  - Change line “set(DEFCON_SOLUTION_1 off)” from off to on
- In /defcon_34 you will find two scripts: build.sh and clean.sh. The scripts will be run later in the appropriate lessons
- Lessons 2-8 follow the same naming and variable conventions



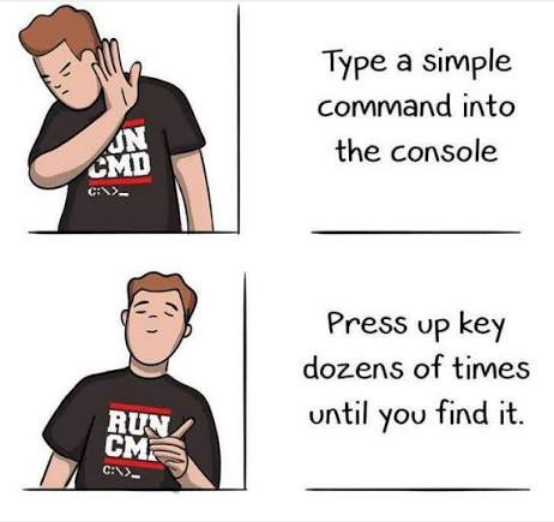
Lesson Exercise

- Open VS Code - Applications > Development > Visual Studio Code
- Navigate to /defcon_34/code/workshop/lesson_1
- Modify main_1.cpp
  - If you need help with the lesson, take a look at solution_1.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_1



Lesson 2

Command-Line Parsing



Lesson Overview

- We’re going to do a decent amount of command-line parsing in this workshop. Let’s take a few minutes to learn a decent API.
- The Boost program_options library allows developers to obtain name/value pairs from the user.
  - The syntax for declaring options is simple, and the library itself is small.
  - Error reporting is better. All the problems with the command line are reported, while hand-written code can just mis-parse the input.
  - The usage message are automatically generated, to avoid falling out of sync with the real list of options.
  - Options can be read from anywhere. Sooner or later the command line will be not enough for your users, and you'll want config files or maybe even environment variables. These can be added without significant effort.

Lesson Exercise

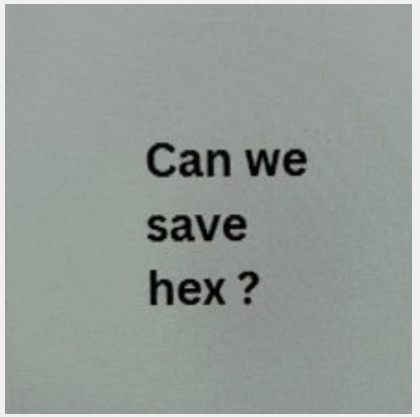
- Navigate to /defcon_34/code/workshop/lesson_2
- Modify main_2.cpp
  - If you need help with the lesson, take a look at solution_2.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_2 -i <integer> -s <string>
  - String needs to be enclosed in quotes if it contains spaces

Lesson 3

Base 16 (Hex)

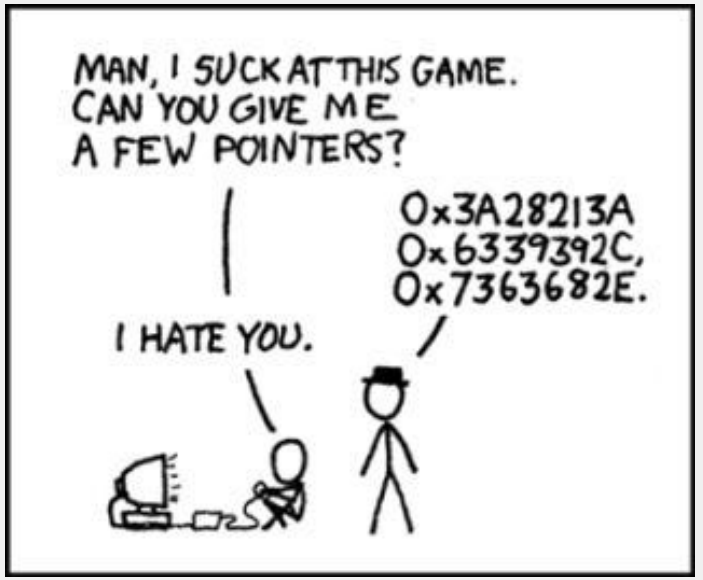
Lesson Overview

- Sometimes there's preliminary code that we must create before we can start the real code
- We need a base16 class so that we can convert from plaintext-to-hex and hex-to-plaintext
- This will help us debug our code since we'll be able to print out any raw variables that we have, e.g. keys, signatures, etc in a user-friendly format.
- Let's create a C++ class that uses the new std::from_chars function (added in C++17) to convert plaintext into hex notation



Lesson Exercise

- Navigate to /defcon_34
- Navigate to /defcon_34/code/workshop/lesson_3
- Modify main_3.cpp
  - If you need help with the lesson, take a look at solution_3.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_3 <value>



Lesson 4

Hashes

Lesson Overview

- SHA-512 is a one-way cryptographic hash function
- Used to verify 2 or more sets of data are the same (integrity checks of messages, files, packages)
- We're going to implement the SHA-512 hash using OpenSSL
- Let's create a C++ class that performs the SHA-512 hash operations of any length of data

Lesson Exercise

- Navigate to /defcon_34/code/workshop/lesson_4
- Modify main_4.cpp
  - If you need help with the lesson, take a look at solution_4.cpp
- Navigate to /defcon_34/code
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_4 <plaintext>

Workshop Break

- 10 Minutes

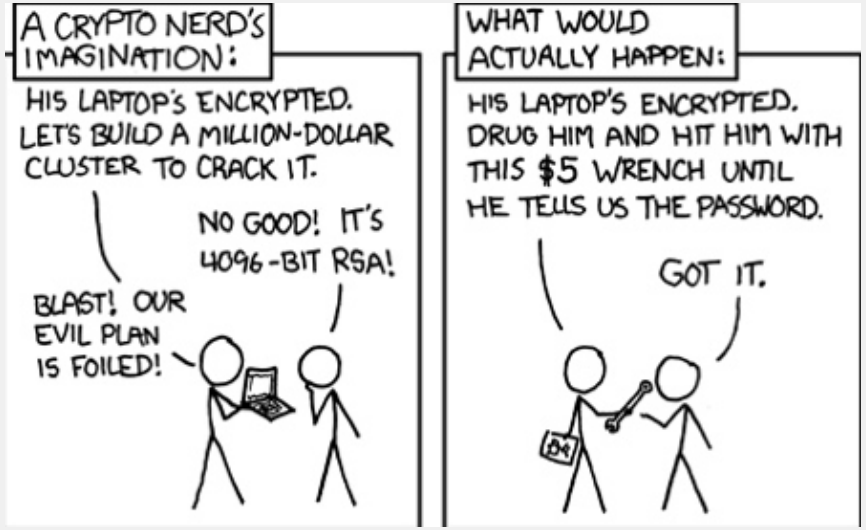
Lesson 5

Ciphers

Using Base64 as “encryption”:

Lesson Overview

- Ciphers are used to guarantee confidentiality in messages
- Advanced Encryption Standard (AES)
  - Rijndael cipher selected as the winner of the NIST AES selection process
  - Certified as part of NIST FIPS 197
  - Modes: ECB, CBC, GCM
- AES-CBC (Cipher Block Chaining)
  - Initialization Vector (IV)
  - Ciphertext padding (can be disabled)
  - Only provides confidentiality
- AES-ECB (Electronic Codebook)
  - No Initialization Vector (IV)

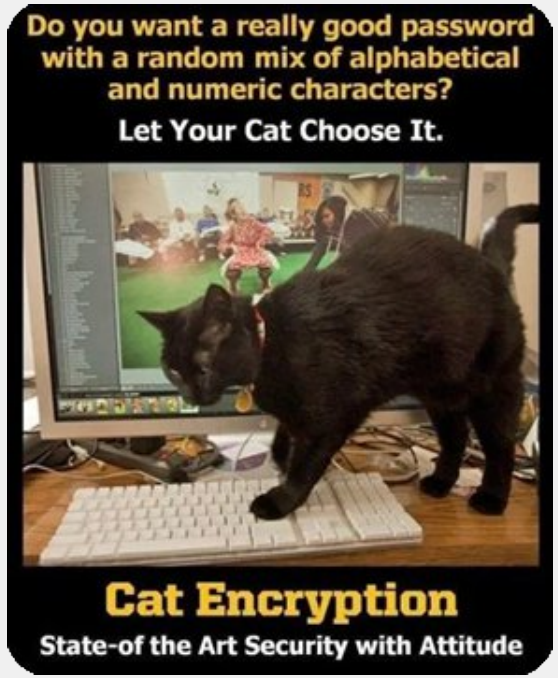


Lesson Overview

- AES-GCM (Galois/Counter Mode)
  - Ciphertext sizes are not limited to modulus 16
  - Authenticated Encryption and Associated Data (AEAD)
    - Combines secrecy and authentication in a single cryptographic primitive
    - Optional Additional Authenticated Data (AAD) field
  - Initialization Vector (IV)
    - Defaults to 12 bytes
  - Message Digest (TAG)
    - Guarantees integrity of data
    - Defaults to 16 bytes

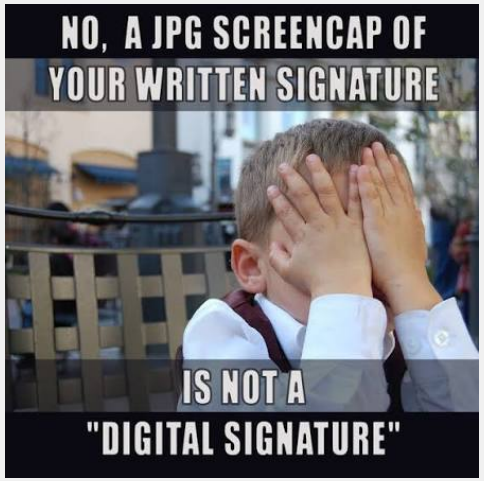
Lesson Exercise

- Navigate to /defcon_34/code/workshop/lesson_5
- Modify main_5.cpp
  - If you need help with the lesson, take a look at solution_5.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_5 -k <key> { -c [ciphertext] | -p [plaintext] }



Lesson 6

- Digital Signature Algorithms



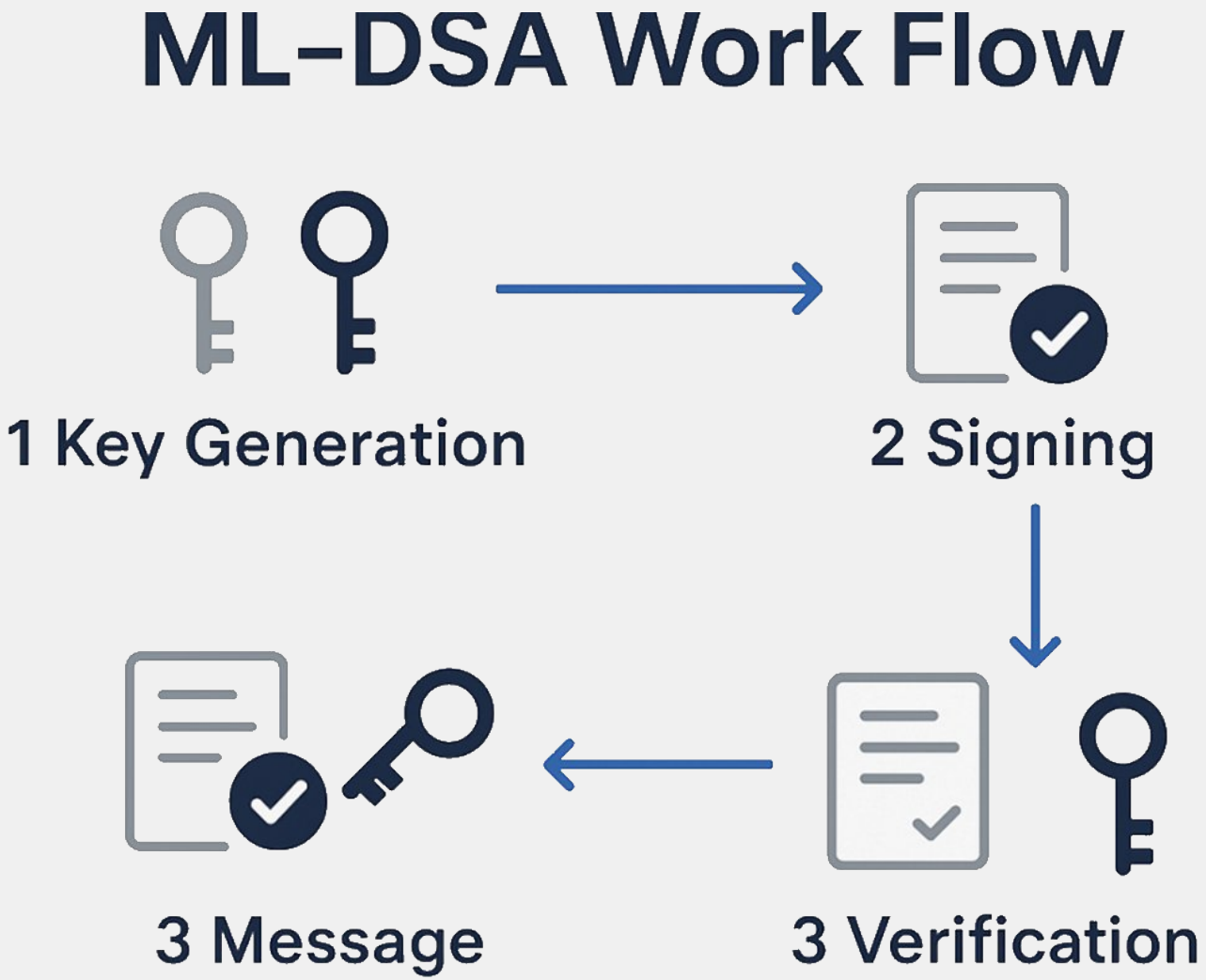
Lesson Overview

- Used for authentication, signing, and secure connections.
- Enforces authenticity, data integrity, and non-repudiation
- Provides secure digital signatures designed to resist attacks from future large-scale quantum computers.
- Relies on the hardness of Module Learning with Errors (MLWE) and Module Short Integer Solution (MSIS)
- Module-Lattice-Based Digital Signature Algorithm (ML-DSA)
  - Created by Cryptographic Suite for Algebraic Lattices (CRYSTALS)
  - Originally named “Dilithium”
  - Replace legacy algorithms like RSA and ECDSA
  - Officially published by NIST in August 2024 (FIPS 204)

Lesson Overview

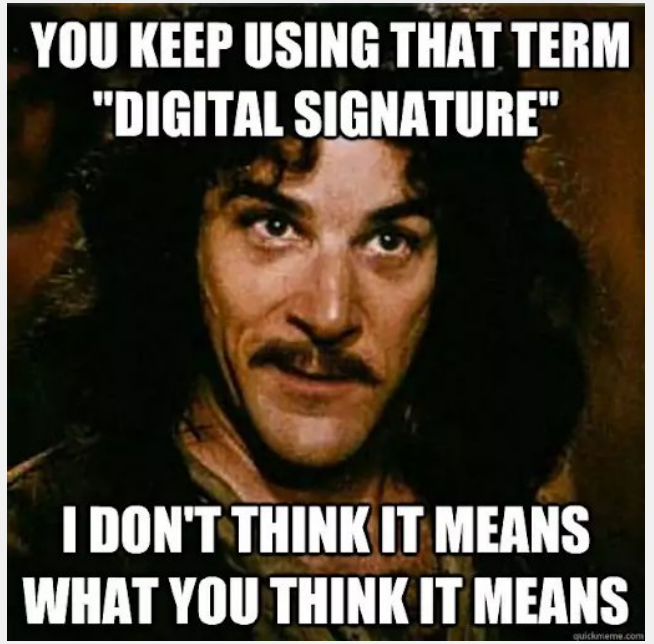
- Standard Parameter Sets
  - **ML-DSA-44:** Provides baseline security strength (128 bit/NIST Level 2)
  - **ML-DSA-65:** Offers intermediate security, widely recommended for general-purpose applications (192 bit/NIST Level 3)
  - **ML-DSA-87:** Delivers the highest security level for long-term sensitive data protection (256 bit/NIST Level 5)
- Trade-offs
  - **Size:** Public keys and signatures are significantly larger
  - **Performance:** Key generation and signing are exceptionally fast, but verification requires more computational overhead than signing.
  - **Memory:** Microcontrollers and embedded hardware may struggle with the higher RAM spikes required to process lattice polynomial multiplications.
  - **Hardware Acceleration:** Legacy cryptographic chips (HSMs) and Trusted Platform Modules (TPMs) lack the specialized vector extensions needed to process ML-DSA efficiently.

Lesson Overview



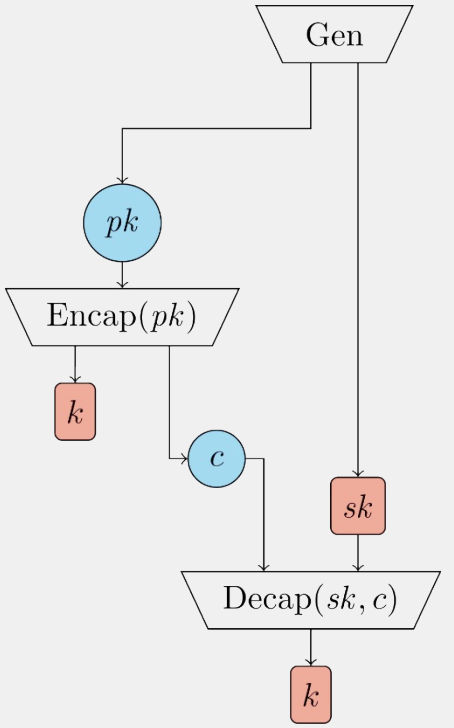
Lesson Exercise

- Navigate to /defcon_34/code/workshop/lesson_6
- Modify main_6.cpp
  - If you need help with the lesson, take a look at solution_6.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_6 <message>



Lesson 7

- Key Encapsulation Mechanism



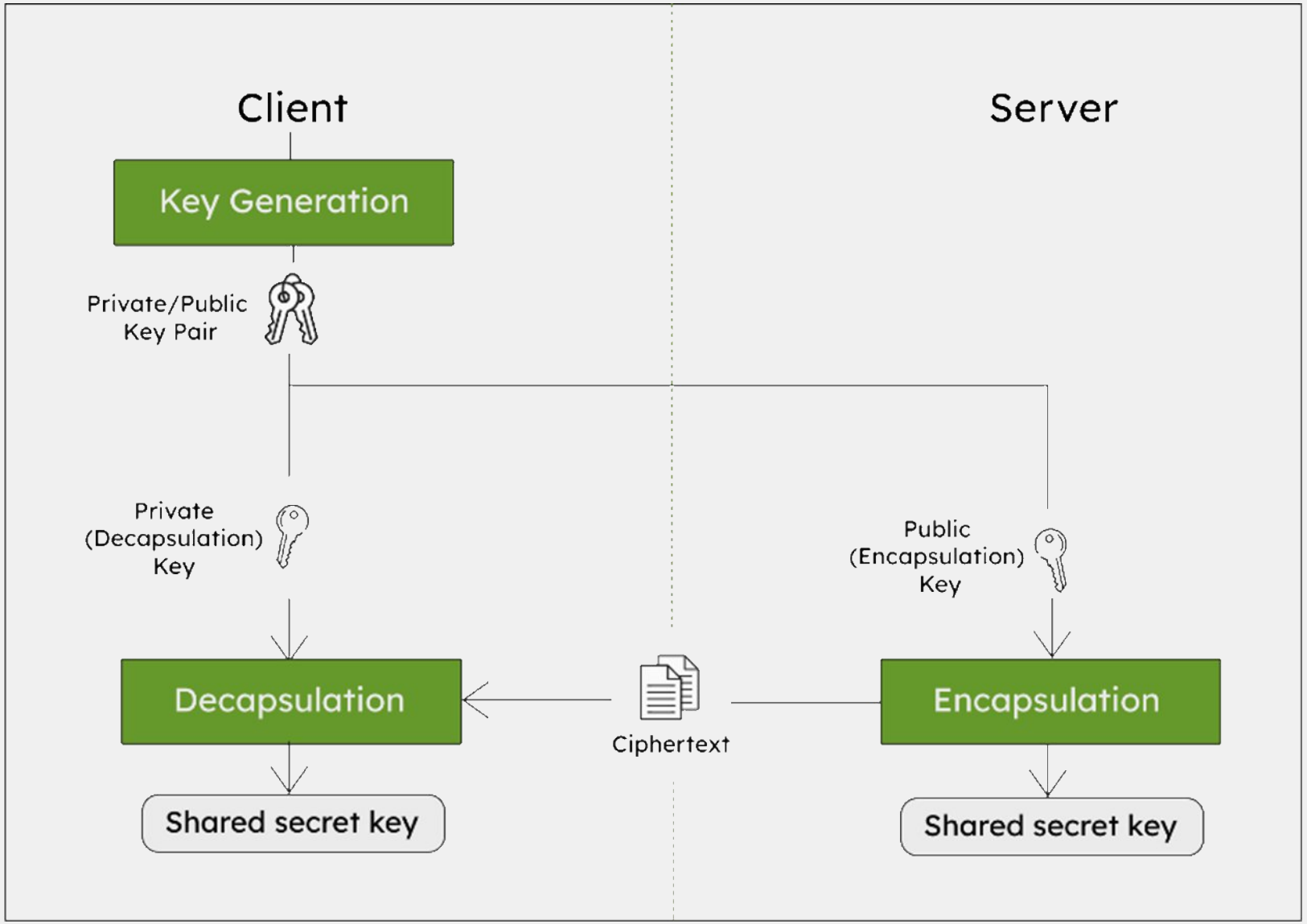
Lesson Overview

- Allows 2 parties to generate a the same secret key without being susceptible to a MITM attack
- Designed to be resistant to cryptanalytic attacks from powerful quantum computers
- Uses a variant of the Learning with Errors (M-LWE) lattice problem as its basic trapdoor function
- Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)
  - Created by Cryptographic Suite for Algebraic Lattices (CRYSTALS)
  - Originally named “Kyber”
  - Replaces quantum-vulnerable public-key cryptography algorithms including RSA key transport and Elliptic Curve Diffie-Hellman (ECDH)
  - Officially published by NIST in August 2024 (FIPS 203)

Lesson Overview

- Standard Parameter Sets
  - **ML-KEM-512:** Provides baseline security strength (128 bit/NIST Level 1)
  - **ML-KEM-768:** Offers intermediate security, widely recommended for general-purpose applications (192 bit/NIST Level 3)
  - **ML-KEM-1024:** Delivers the highest security level for long-term sensitive data protection (256 bit/NIST Level 5)
- Cryptographic Steps
  - **1) Key Generation:** A party creates a public key and a private key using structured lattice math.
  - **2) Encapsulation:** Another party uses the public key to lock a shared secret into a ciphertext packet and sends it over a public network.
  - **3) Decapsulation:** The original key owner uses their private key to unlock the ciphertext and recover the identical shared secret.

Lesson Overview



Lesson Exercise

- Navigate to /defcon_34/code/workshop/lesson_7
- Modify main_7.cpp
  - If you need help with the lesson, take a look at solution_7.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_7

Lesson 8

- PQC Handshake

Lesson Overview

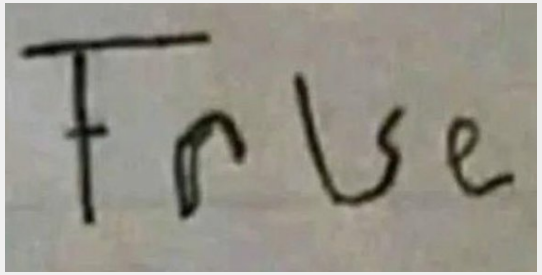
- How would we design a protocol for performing a PQC handshake for the purpose of shared key derivation?
- Let's use all the PQC algorithms we've learned so far
  - ML-DSA and ML-KEM in the handshake
  - We could then use AES-256 in GCM mode to secure message
  - SHA-512 for identities and Base16 for readability
- Which of the core security principles would be want to use for this protocol?
  1. Confidentiality
  2. Authenticity
  3. Integrity
  4. Non-repudiation

Lesson Overview

- What are some of the components that we would want to include in the PQC handshake request?
  - Datetime and/or sequence number to prevent replay attacks
  - Sender's ML-DSA public key
  - Sender's ML-KEM public key
  - Sender's ML-DSA signature of all previous bytes
- What are some of the components that we would want to include in the PQC handshake response?
  - Datetime and/or sequence number to prevent replay attacks
  - Receiver's ML-DSA public key
  - Receiver's ML-KEM ciphertext (generated from sender's ML-KEM public key)
  - Receiver's ML-DSA signature of all previous bytes

Lesson Overview

- Design a simple PQC handshake protocol
  - Request, Response
- Generate a PQC request handshake
  - Dilithium persistent keys, Kyber ephemeral keys, message signature
- Simulate a PQC response handshake
  - Cryptographic message verification, Kyber encap
- Consume the PQC response
  - Message verification, Kyber decap, Derive shared AES-256 key



Lesson Exercise

- Navigate to /defcon_34/code/workshop/lesson_8
- Modify main_8.cpp
  - If you need help with the lesson, take a look at solution_8.cpp
- Navigate to /defcon_34
- Run ./build.sh
  - If you need to clean the project, run ./clean.sh
- Navigate to /defcon_34/bin
- Run ./lesson_8

Summary

"It's in that place I put that thing that one time"

— Hackers, The Phantom Phreak

Knowledge is Pwnage

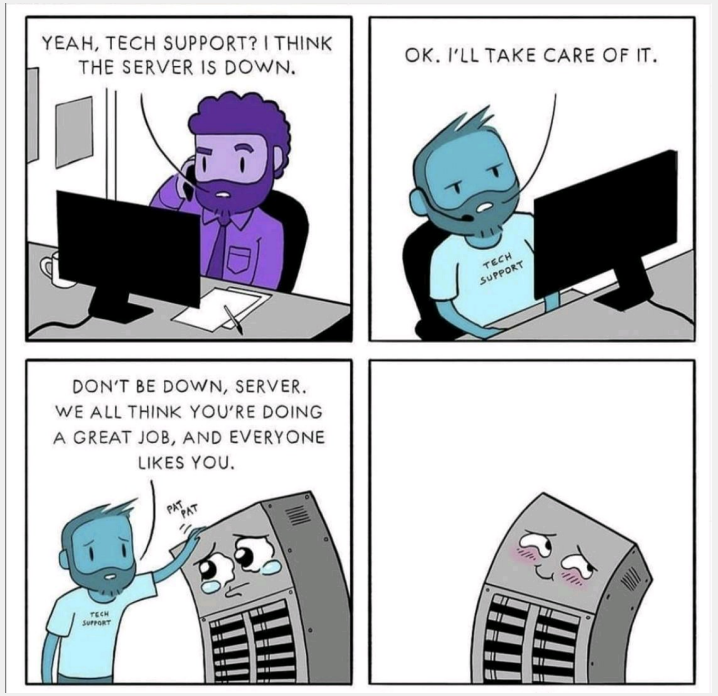
- We might not all have something to hide, but we all have something worth protecting.
- What are the possible uses of Post-Quantum Cryptography?
- What are your ideas?

Closing Thoughts

- This workshop is for educational purposes.
- The PQC codebase has been simplified
  - FIPS 140-3 certification (CAVP, CMVP)
  - Additional algorithms
    - Deterministic Random Bit Generator (DRBG)
    - CNSA Suite 1.0
  - Options
    - ML-DSA, ML-KEM seeds
    - Additional Authenticated Data (AAD)
    - Testing: IKME, MU, RND, etc.
    - Contexts

Next Steps

- How do I learn how to become an awesome programmer?
  - The best way to learn is to do
- Open source communities, online resources, books, etc.
- Remember… you're hackers… you can do anything!
- Thanks for spending the day with us!
- We hope you've enjoyed this DEF CON workshop
- We do this for you. <3



Check us out…

https://www.codesiren.com

References

- https://www.debian.org
- https://gcc.gnu.org
- https://www.boost.org
- https://cmake.org
- https://openssl.org
- https://media.defense.gov/2025/May/30/2003728741/-1/-1/0/CSA_CNSA_2.0_ALGORITHMS.PDF
- https://en.wikipedia.org/wiki/Grover%27s_algorithm
- https://en.wikipedia.org/wiki/Shor%27s_algorithm
- https://en.wikipedia.org/wiki/Digital_Signature_Algorithm
- https://en.wikipedia.org/wiki/Key_encapsulation_mechanism
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- https://en.wikipedia.org/wiki/SHA-2