STUDENT HANDOUT · WEEK 3

Change Management and Cryptographic Solutions

SY0-701 objectives 1.3, 1.4. Notes, exercises, review questions, and the lab.

Name	
Date	

THIS WEEK YOU WILL

1. Walk a change through approval, impact analysis, testing, a maintenance window, and a backout plan.
2. Predict the technical side effects of a change, from allow lists to dependencies.
3. Compare symmetric and asymmetric encryption and explain how key exchange joins them.
4. Choose an encryption level, a key storage tool, or an obfuscation method for a stated need.
5. Explain hashing, salting, key stretching, and digital signatures, and what each one proves.
6. Trace a certificate from CSR to a trusted root, and check its status with a CRL or OCSP.

KEY IDEAS

A change follows a path before it goes live

1. Request	The owner states what, why, when, and who is affected
2. Analyze	Impact analysis: the risk of changing and of not changing
3. Test	Try it in a test environment and record the results
4. Approve	The change advisory board approves or rejects it
5. Implement	Apply it in the maintenance window, backout plan ready
6. Document	Update diagrams, procedures, and version history

What a change can break

Implication	What to plan for
Allow and deny lists	An updated app may be blocked until lists change
Restricted activities	Do only what was approved, nothing extra
Downtime	Users lose the service; schedule and announce it
Service and app restarts	A restart can drop sessions or fail to come back
Legacy applications	Fragile and poorly documented; test with care
Dependencies	Other systems that break when this one changes

Symmetric is fast; asymmetric solves key sharing

Symmetric	Asymmetric
One shared secret key encrypts and decrypts	A key pair: a public key and a private key
Fast; used for bulk data on disks and in TLS	Encrypt with the public, decrypt with the private
Hard part: getting the key to the other side	Slower; used for key exchange and signatures
Examples: AES-128, AES-256, ChaCha20	Examples: RSA, ECC, Diffie-Hellman

Pick the level that matches the risk

Level	What it protects	Example
Full-disk	The whole drive, including the OS	BitLocker on a clinic laptop
Partition	One partition of a drive	An encrypted data partition on a server
Volume	A logical volume, which can span disks	An encrypted volume on a file server
File	Individual files or folders	An encrypted spreadsheet sent to billing
Database	An entire database at rest	Transparent data encryption on the EHR
Record	Single rows or fields in a table	Only the SSN column is encrypted

Keys need a safe place to live

TPM	A chip on the motherboard that protects one device's keys. (BitLocker key protection)
HSM	A hardened device that creates and guards keys at scale. (A CA's root signing key)

Key management system	Creates, rotates, revokes, and audits keys in one place. (A cloud key vault)
Secure enclave	An isolated processor area that protects data in use. (Phone fingerprint data)

Hide data instead of only encrypting it

Steganography	Hides a message inside an image, audio, or other file. (Data hidden in a JPEG)
Tokenization	Swaps real data for a token; a secure vault maps it back. (Card number to random token)
Data masking	Shows only part of the data, or realistic fake data. (Card ending in 4821)

One-way math that proves data is unchanged

Technique	What it does	Example
Hashing	Fixed-length fingerprint; cannot be reversed	SHA-256 of an installer
Salting	Adds a random value before hashing a password	Same password, different stored hash
Key stretching	Repeats hashing to slow down guessing	PBKDF2, bcrypt
Digital signature	A hash signed with the sender's private key	A signed lab result PDF
Blockchain	Blocks linked by hashes, so edits show	Each block holds the prior block's hash
Open public ledger	A shared record anyone can read and check	A cryptocurrency's transaction history

From key pair to trusted certificate

1. Key pair	Generate a private key that never leaves the server
2. CSR	Send the public key and the name to the CA
3. Validate	The CA checks that you control the name
4. Issue	The CA signs the certificate, chained to its root
5. Check status	Clients check revocation with a CRL or OCSP
6. Renew	Replace it before expiry; revoke it if the key leaks

EXERCISE 1: WHAT IS WRONG WITH THIS CHANGE REQUEST

8 MINUTES · PAIRS

Read the Lakeside change request on the lesson page. List each missing or weak part and what it should say instead.

#	Prompt	Your answer
1	Who owns the change, and who approved it?	
2	Which stakeholders were told, and when?	
3	What was tested, and where are the results?	
4	What is the backout plan, step by step?	
5	Is 10 a.m. on a Tuesday a good maintenance window?	
6	What depends on the EHR client?	
7	What documentation must change afterward?	

EXERCISE 2: MATCH THE SOLUTION TO THE CLINIC'S NEED

10 MINUTES · SMALL GROUPS

For each need, name the cryptographic solution or tool that fits best and one reason it beats the closest alternative.

#	Prompt	Your answer
1	Protect patient data on a laptop if it is stolen	
2	Store staff passwords in the scheduling app's database	
3	Let billing staff see only the last four digits of a card	
4	Keep real card numbers out of the clinic's systems	
5	Prove a lab result PDF came from the clinic, unaltered	
6	Secure the public patient portal at portal.lakeside.example	
7	Recover a laptop's encryption key after the user leaves	
8	Check right now whether a server's certificate was revoked	
9	Encrypt traffic to a test web server only IT staff use	

REVIEW QUESTIONS

1. What is a backout plan, and when is it written?

2. Why does salting stop a rainbow table attack on stored passwords?

3. Which key creates a digital signature, and which key verifies it?

4. What is the difference between a CRL and OCSP?

5. A card number is replaced by a random value, and only a secure vault can map it back. What is this?

KEY TAKEAWAYS

- **Every change needs a way back.** Approve, test, schedule, and write the backout plan before the change.
- **Key exchange, then AES.** Key exchange agrees on a symmetric key, which then encrypts the data.
- **Hashes prove nothing changed.** Salt and stretch password hashes; sign a hash to prove who sent it.
- **Trust flows from a root.** A certificate is trusted only if it chains to a root you already trust.

LAB 3: HASHING, ENCRYPTION, AND CERTIFICATES WITH OPENSSL

60 MINUTES

Use the Linux VM from Lab 1, which has OpenSSL installed; check with openssl version. Work in a new folder, mkdir ~/lab3 && cd ~/lab3. Step 8 needs internet access, so do it on your host computer or briefly switch the VM's adapter back to NAT, and switch it back afterward.

Steps

1. Hash a file. Run echo "Lakeside Family Clinic" > note.txt and openssl dgst -sha256 note.txt. Change one letter in the file with nano note.txt, hash it again, and record both hashes.
2. Encrypt with a symmetric key. Run openssl enc -aes-256-cbc -salt -pbkdf2 -iter 100000 -in note.txt -out note.enc and enter a passphrase. Try cat note.enc, then decrypt with openssl enc -d -aes-256-cbc -pbkdf2 -iter 100000 -in note.enc -out note.dec and compare with diff note.txt note.dec.
3. Create a key pair. Run openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits:3072 -out private.pem, then extract the public key with openssl pkey -in private.pem -pubout -out public.pem. Look at both with cat.
4. Encrypt with the public key. Run echo "Room 4 code 7731" > secret.txt, then openssl pkeyutl -encrypt -pubin -inkey public.pem -in secret.txt -out secret.enc, and decrypt with openssl pkeyutl -decrypt -inkey private.pem -in secret.enc.
5. Sign and verify. Run openssl dgst -sha256 -sign private.pem -out note.sig note.txt, then openssl dgst -sha256 -verify public.pem -signature note.sig note.txt and expect Verified OK. Change one character in note.txt and verify again.
6. Generate a CSR. Run openssl req -new -key private.pem -out portal.csr -subj "/CN=portal.lakeside.test/O=Lakeside Family Clinic" and read it with openssl req -in portal.csr -noout -text.
7. Create a self-signed certificate. Run openssl req -x509 -new -key private.pem -days 365 -out portal.crt -subj "/CN=portal.lakeside.test" and view it with openssl x509 -in portal.crt -noout -subject -issuer -dates. Note that the subject and issuer match.
8. Inspect a real certificate chain. On a machine with internet access, run openssl s_client -connect www.wikipedia.org:443 -servername www.wikipedia.org -showcerts < /dev/null and record the subject and issuer of each certificate the server sends. The issuer of the last one names the root CA your trust store must already hold.

Hand in

A lab report with the commands you ran, screenshots of the two different hashes, the decrypted file matching the original, Verified OK followed by the failed verification, and the certificate chain you inspected, plus answers to the questions below.

Questions to answer in your report

1. How much of the SHA-256 hash changed when you changed one letter? What property of hashing does this show, and which CIA goal does it support?
2. In step 2, what did the -salt and -pbkdf2 options add, and which attack does each one slow down?
3. Why did verification fail after you edited note.txt? What three things does a successful verification prove?
4. Your self-signed certificate and the Wikipedia certificate are both valid X.509 certificates. Why would a browser trust one and warn about the other?
5. Which file from this lab must never leave the server, and what would an attacker be able to do with it?