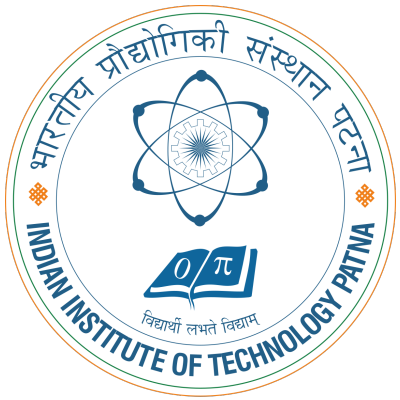
Chapter 2: Toolbox: Authentication, Access Control, and Cryptography

Dr. Mayank Agarwal

Department of CSE
IIT Patna

CS457 Big Data Security
Jan-April 2026



- The Three Foundational Pillars of Security
- From Identity to Access to Protection
- Building the Security Toolkit

Welcome to the security toolbox. Chapter 2 covers the three most fundamental technical controls: proving who you are (authentication), determining what you can do (access control), and protecting data (cryptography). These are the building blocks for all secure systems.

Chapter 2 Learning Objectives

- Understand the concepts and mechanisms of authentication.
- Comprehend different access control models and policies.
- Learn the basic principles, terminology, and applications of cryptography.
- Recognize how these three components work together in a security system.

By the end of this chapter, you will understand the core mechanisms that enforce the security principles we discussed in Chapter 1. These are not abstract concepts—they are the concrete tools you will implement and analyze.

The Security Toolbox: An Integrated View

- **Authentication:** "Who are you?" (Verifies identity)
- **Access Control:** "What are you allowed to do?" (Enforces policy)
- **Cryptography:** "How is information protected?" (Provides confidentiality, integrity, etc.)
- They operate in sequence: Identity ? Authorization ? Protected Action

Think of entering a secure building: You show your badge (authentication), the guard checks if you're allowed in this area (access control), and once inside, you lock sensitive documents in a safe (cryptography). All three are needed for complete security.

Part 1: Authentication

- **Definition:** The process of verifying a claimed identity.
- Establishes **accountability**—actions can be traced to a specific entity.
- Fundamental to all access control: You must know *who* is making the request.

Authentication is the first gate. Without it, we cannot implement least privilege or complete mediation. It answers the question: "To whom am I speaking?" Everything else in security depends on a reliable answer.

Authentication Factors

- **Something you know:** Password, PIN, passphrase, security question.
- **Something you have:** Smart card, security token, mobile device, key.
- **Something you are:** Biometric characteristic (fingerprint, iris, face, voice).
- **Somewhere you are:** Location/network context (IP address, GPS).
- **Something you do:** Behavioral biometrics (typing rhythm, signature dynamics).

The strength of authentication often depends on the number and type of factors used. Single-factor (e.g., password only) is weak. Multi-factor authentication (MFA) combines factors from different categories, dramatically increasing security.

Single-Factor Authentication: The Password

- Most common authentication mechanism.
- **Vulnerabilities:** Weak/predictable choices, sharing, writing down, phishing, shoulder surfing, offline cracking of stored hashes.
- **Defenses:** Complexity policies, password managers, secure hashing with salt, rate-limiting login attempts.
- Psychological Acceptability vs. Security trade-off is pronounced here.

Passwords suffer from fundamental human limitations. Users choose memorable passwords, which are often guessable. The "Crackability" of a password is a function of length, complexity, and unpredictability. No password policy is perfect.

Storing Passwords Securely

- **NEVER store passwords in plaintext.**
- Use cryptographically strong, **salted hash functions**.
- **Salt:** A random value unique to each user, concatenated with the password before hashing.
- Purpose: Prevents rainbow table attacks and ensures identical passwords yield different hashes.

When a database is breached, the attacker gets the password table. If passwords are hashed with a unique salt, they must crack each password individually. Algorithms like bcrypt, scrypt, or Argon2 are designed to be computationally expensive (slow) to resist brute-force attacks.

Multi-Factor Authentication (MFA)

- Requires two or more *different* factors.
- Significantly reduces risk because compromising multiple factors is much harder.
- Common implementation: Password (something you know) + One-Time Code from phone (something you have).
- **Strong Authentication** typically implies the use of MFA.

MFA is the single most effective control to prevent account takeover. Even if a password is phished or stolen, the attacker likely doesn't have the user's physical token or biometric. This implements the *Separation of Privilege* principle.

Biometric Authentication

- Uses unique physiological or behavioral characteristics.
- **Advantages:** Difficult to steal or share, convenient (always with you).
- **Disadvantages:** Not secret (you leave fingerprints everywhere), not revocable (can't change your iris), false acceptance/rejection rates, privacy concerns.
- Used for identification ("Who is this?") or verification ("Is this John?").

Biometrics are probabilistic, not absolute. Systems must be tuned between False Acceptance Rate (FAR) and False Rejection Rate (FRR). They are best used as one factor in MFA, not as a standalone secret.

Authentication Protocols and Challenges

- **Challenge-Response:** Verifier sends a challenge (e.g., a random number), claimant responds with a transformation using their secret. Proves knowledge without transmitting the secret.
- **Zero-Knowledge Proof:** Prover convinces verifier they know a secret without revealing any information about the secret itself.
- **Mutual Authentication:** Both parties authenticate each other (client & server).

Good protocols prevent eavesdropping and replay attacks. For example, in a simple challenge-response, if the challenge is fresh (non-repeating), a captured response cannot be reused. This is critical for network authentication.

Single Sign-On (SSO)

- User authenticates once and gains access to multiple related systems.
- **Benefits:** User convenience, reduced password fatigue, centralized authentication management.
- **Risks:** Single point of failure; compromise of SSO credentials grants access to all connected systems.
- Examples: Kerberos, OAuth, SAML-based enterprise logins.

SSO represents a trade-off. It improves usability (Psychological Acceptability) but violates *Least Common Mechanism*—many systems depend on one authentication service. Its security depends entirely on the strength of that central service.

Part 2: Access Control

- **Definition:** The process of mediating every request to data and resources to determine if the request should be granted or denied.
- Operates after authentication.
- Implements the security policy: "Who can do what to which resource and under what conditions?"

Access control is *Complete Mediation* in action. It is the gatekeeper that enforces *Least Privilege*. Every file open, every database query, every network connection should be subject to an access control check.

Access Control Triad

- **Subject:** An active entity (user, process) that requests access.
- **Object:** A passive entity (file, database record, printer) being accessed.
- **Access Right:** The type of access being requested (read, write, execute, delete).
- The access control system decides if a specific **Subject** has a specific **Access Right** to a particular **Object**.

This is the fundamental model. A subject (e.g., a word processor process running as user Alice) requests an access right (e.g., WRITE) on an object (e.g., the file "report.doc"). The reference monitor makes the decision.

Reference Monitor Concept

The Reference Monitor is a security concept in operating systems that controls all access to system resources (files, memory, devices, processes).

It acts like a gatekeeper between a subject (user, process) and an object (file, database, device).

- The abstract model of the mediator that controls all access.
- **Three Key Properties:**
  - **Tamperproof:** Cannot be bypassed.
  - **Always invoked:** Complete mediation.
  - **Small and simple enough to be verified:** Economy of mechanism.

The reference monitor is the security kernel. In an operating system, it's part of the kernel that checks every system call. Its correct implementation is the core of system security. It must be trusted (high assurance).

For a system to truly implement a reference monitor, it must satisfy:

1. Complete Mediation: Every access request must be checked. No bypass allowed.
2. Tamperproof: It cannot be modified by unauthorized users.
3. Verifiable: Its correctness must be provable and testable. Simple and small enough to be verified.

Access Control Lists (ACLs)

An Access Control List (ACL) is a list attached to a resource (object) that specifies which users or groups are allowed or denied specific types of access.

In simple words: ACL = "Who can do what" on a particular resource.

- **Model:** Access control matrix stored by column (per object).
- Each object has a list (the ACL) of subjects and their permitted rights.
- **Example:** File permissions in UNIX/Linux (`rwxr-xr--`) are a compact form of an ACL.
- Easy to answer: "Who has access to this object?"

ACLs are object-centric. When you check file properties to see who can access it, you are looking at an ACL. They are intuitive for managing permissions on shared resources like network fileshares.

Capabilities (or Tickets)

A capability (also called a ticket) is a special, unforgeable token that grants a subject the right to access a specific object in a specific way.

ACL → stored with object (who can access me?)
Capability → stored with user (what can I access?)

- **Model:** Access control matrix stored by row (per subject).
- Each subject holds a set of unforgeable tokens (capabilities) that specify their access rights to objects.
- Possession of the capability is proof of access right.
- Easy to answer: "What objects can this subject access?"

Capabilities are like keys. If you have the key to a room, you can enter.

You don't need to check a central list. They are efficient for distributed systems but revocation is difficult—how do you take back a key?

Discretionary Access Control (DAC)

Discretionary Access Control (DAC) is an access control model where the owner of a resource decides who is allowed to access it.

The word discretionary means the control is at the discretion of the owner.

- The *owner* of the object decides who can access it and what rights they have.
- Control is at the discretion of the user.
- Common in personal and commercial operating systems (Windows, Linux).
- **Weakness:** The "confused deputy" problem; users can accidentally or maliciously grant excessive rights.

DAC embodies a decentralized, user-centric model. It's flexible but risky from an organizational security perspective. If a user owns sensitive data, they can share it with anyone, potentially violating organizational policy.

Mandatory Access Control (MAC)

- Access is controlled by a central authority based on fixed security *labels*.
- Users cannot change permissions, even on objects they "own."
- Used in military and high-security environments (e.g., Bell-LaPadula model).
- Implements a system-wide security policy.

MAC is policy-centric, not user-centric. If a file is labeled "Top Secret," a user with only "Secret" clearance cannot access it, no matter what the file's owner wants. This prevents users from overriding organizational security rules.

Role-Based Access Control (RBAC)

Access Rights are given on the basis of roles and responsibilities, not based on individual user.

- Access rights are assigned to **roles**, not individual users.
- Users are assigned to roles.
- **Core principle:** Separation of duty—users can be assigned conflicting roles to prevent fraud.
- Efficient for large organizations: changing a user's job means changing their role assignment, not hundreds of individual permissions.

RBAC is the dominant model in enterprise systems. A "Doctor" role has access to patient records. A "Pharmacist" role can dispense medication. Users (Dr. Smith) are assigned these roles. This simplifies administration and enforces least privilege by role.

Rule-Based Access Control

- Access is granted or denied based on a set of rules defined by a system administrator.
- Rules are typically triggered by conditions (time of day, location, system status).
- Often used in network devices (firewalls): "IF source IP = X AND destination port = 80 THEN ALLOW."

Rule-based access control is context-aware. It can enforce policies like "Employees can access the payroll system only from the corporate network during business hours." It's flexible but rule sets can become complex and contradictory.

Attribute-Based Access Control (ABAC)

- A generalized model where access decisions are based on **attributes** of the subject, object, action, and environment.
- **Example:** "A subject with attribute `Department=Finance` can perform action `View` on an object with attribute `Classification=Internal` if the environment attribute `Time` is between 9 AM and 5 PM."
- Very flexible and powerful, used in cloud services and complex policies.

ABAC is like a sophisticated rule engine. It can express highly nuanced policies. XACML (eXtensible Access Control Markup Language) is a standard for ABAC. It's powerful but can be complex to manage and has performance implications.

Part 3: Cryptography

- **Definition:** The practice and study of techniques for secure communication in the presence of adversaries.
- Provides the mathematical foundations for confidentiality, integrity, authentication, and non-repudiation.
- Not a security product, but a set of tools for building security.

Cryptography turns security problems into key management problems. If you can keep a small secret key safe, you can protect vast amounts of data. It is the ultimate implementation of the *Open Design* principle.

Cryptographic Terminology

- **Plaintext/Cleartext:** The original, readable message.
- **Ciphertext:** The scrambled, unreadable message.
- **Encryption:** The process of converting plaintext to ciphertext.
- **Decryption:** The process of converting ciphertext back to plaintext.
- **Cipher/Cryptosystem:** The algorithm (and its implementation) for encryption/decryption.
- **Key:** A secret value that controls the encryption/decryption process.

Remember Kerckhoffs's Principle: The security of a cryptosystem must depend solely on the secrecy of the key, not the secrecy of the algorithm. This allows for public scrutiny and standardization.

Symmetric (Secret-Key) Cryptography

- **Same key** used for both encryption and decryption.
- Fast and efficient for bulk data encryption.
- **Problem:** Key Distribution. How do you securely share the secret key with the intended recipient?
- Examples: AES (Advanced Encryption Standard), 3DES, ChaCha20.

Imagine a physical padlock where the same key locks and unlocks it. To send a locked box to someone, they need a copy of your key. Getting that key to them securely is the fundamental challenge. This is also called private-key cryptography.

Asymmetric (Public-Key) Cryptography

- Uses a mathematically related **key pair:** a public key and a private key.
- **Public Key:** Can be shared openly with everyone. Used for encryption or signature verification.
- **Private Key:** Kept secret by the owner. Used for decryption or signature creation.
- Solves the key distribution problem.
- Examples: RSA, Elliptic Curve Cryptography (ECC), Diffie-Hellman.

This is a revolutionary concept. You can give everyone a padlock (public key) that only you have the key to open (private key). Anyone can lock a box for you, but only you can open it. Or, you can prove your identity by creating a signature only your private key can make.

Comparing Symmetric and Asymmetric Crypto

- **Speed:** Symmetric is much faster (100-1000x).
- **Key Management:** Asymmetric solves distribution but private keys must still be kept secret.
- **Use Cases:**
  - Symmetric: Encrypting files, database fields, VPN traffic.
  - Asymmetric: Key exchange, digital signatures, encrypting small amounts of data (like a symmetric key).

In practice, systems use a hybrid approach. Asymmetric crypto is used to securely exchange a symmetric key (a session key), which is then used to encrypt the actual data. This combines the strengths of both.

Cryptographic Hash Functions

- Takes an input (message) of any size and produces a fixed-size output (hash, digest).
- **Properties:**
  - **Deterministic:** Same input = same hash.
  - **Pre-image Resistance:** Given a hash, it's infeasible to find the original input.
  - **Collision Resistance:** Infeasible to find two different inputs that produce the same hash.
  - **Avalanche Effect:** A small change in input completely changes the hash.

Hash functions are one-way streets. You can't get the original data back from the hash. They are crucial for password storage (as discussed), data integrity checks (checksums), and digital signatures. Examples: SHA-256, SHA-3.

Message Authentication Codes (MACs)

- Used to provide **integrity** and **authenticity** of a message.
- A function that takes a message and a secret key, producing a MAC tag.
- The recipient, who also knows the key, can recompute the MAC and verify it matches.
- Proves the message was not altered and came from someone with the same secret key.

Think of it as a "keyed hash." Unlike a regular hash, an attacker cannot forge a valid MAC without knowing the secret key. HMAC (Hash-based MAC) is a common construction. Used in network protocols and file integrity checking.

Digital Signatures

- Provide **integrity, authenticity, and non-repudiation**.
- Uses asymmetric cryptography.
- **Process:** Sender creates a hash of the message, then encrypts the hash with their **private key**. This encrypted hash is the signature, appended to the message.
- **Verification:** Recipient decrypts the signature with the sender's **public key** to get the hash, compares it to their own hash of the received message.

Because only the sender has the private key, a valid signature proves the message came from them (authenticity) and hasn't changed (integrity). And they cannot later deny sending it (non-repudiation). This is foundational for e-commerce and legal documents.

Public Key Infrastructure (PKI)

- The system needed to support the distribution and trust of public keys.
- Solves the problem: "Is this public key really Bob's key, or is it an attacker's?"
- Central components: Digital Certificates, Certificate Authorities (CAs), Registration Authorities (RAs).
- The "trust fabric" of the internet (SSL/TLS).

A PKI is like a digital notary public. A trusted third party (the CA) vouches for the binding between an identity (e.g., "google.com") and a public key by issuing a digitally signed certificate. Your browser trusts a list of well-known CAs.

Digital Certificates (X.509)

- A digital document that binds a public key to an identity.
- Contains: Owner's identity, owner's public key, issuer's name (CA), issuer's digital signature, validity period.
- The CA's signature on the certificate is what you trust.
- Forms a chain of trust: Root CA -¿ Intermediate CA -¿ End-entity certificate.

When you visit https://bank.com, your browser receives bank.com's certificate. It checks that the certificate is signed by a CA it trusts, that the domain matches, and that it's not expired or revoked. This is how you know you're really connected to your bank.

Cryptographic Protocols: SSL/TLS

- Secure Sockets Layer / Transport Layer Security.
- The protocol for secure web browsing (HTTPS).
- Uses a hybrid approach:
  1. Client and server negotiate cipher suites.
  2. Server authenticates with its certificate (PKI).
  3. They use asymmetric crypto (e.g., Diffie-Hellman) to establish a shared secret.
  4. That secret is used to generate symmetric session keys for bulk encryption.

TLS provides a secure channel with confidentiality, integrity, and server authentication (and optionally client authentication). It's a masterpiece of applied cryptography, combining nearly all the tools we've discussed into one protocol.

Key Management: The Hardest Part

- Cryptography's strength depends entirely on key secrecy.
- **Lifecycle:** Generation, distribution, storage, use, archival, destruction.
- **Storage:** Hardware Security Modules (HSMs), Trusted Platform Modules (TPMs), key vaults.
- **Distribution:** Out-of-band, PKI, key exchange protocols (Diffie-Hellman).

You can have the world's strongest encryption algorithm, but if the key is on a sticky note on the monitor, the system is insecure. Key management is an *administrative* and *technical* challenge. Poor key management is a common cause of cryptographic failure.

Trusted Platform Module (TPM): Hardware Security Foundation

- **Definition:** A dedicated microcontroller providing hardware-based security functions
- **Standardization:** International standard (ISO/IEC 11889) maintained by Trusted Computing Group (TCG)
- **Integration:** Typically embedded in motherboard or integrated into CPU (fTPM)
- **Purpose:** Securely generate, store, and manage cryptographic keys and measurements

The TPM is a hardware root of trust—a foundational security component that other security mechanisms can rely upon. It provides protection even if the operating system is compromised, as keys never leave the secure hardware boundary.

TPM Key Capabilities and Functions

- **Random Number Generation:** True hardware random number generator for cryptographic operations
- **Secure Key Storage:** Keys stored in TPM's protected memory, never exposed in plaintext
- **Remote Attestation:** Cryptographic proof of system state/configuration to remote parties
- **Platform Configuration Registers (PCRs):** Store cryptographic measurements of boot components
- **Sealing/Unsealing:** Binding data to specific system state (e.g., decrypt only if system is uncompromised)

These capabilities enable advanced security features like BitLocker drive encryption, Windows Hello for Business, and measured boot. The TPM's isolation from the main CPU provides tamper resistance against software attacks.

TPM Versions and Evolution

- **TPM 1.2 (2003):** First widespread adoption, used SHA-1, limited functionality
- **TPM 2.0 (2014):** Major redesign, supports multiple cryptographic algorithms (SHA-256, ECC)
- **Discrete vs. Integrated vs. Firmware TPM:**
  - Discrete: Separate physical chip (highest security)
  - Integrated: Part of another chip (e.g., chipset)
  - Firmware (fTPM): Implemented in CPU firmware (AMD PSP, Intel PTT)
- **Platform Trust Technology (PTT):** Intel's firmware-based TPM 2.0 implementation

TPM 2.0 represents a significant advancement with support for modern cryptography and more flexible authorization mechanisms. While discrete TPMs offer strongest physical security, firmware TPMs provide broader adoption in consumer devices.

TPM Applications in Modern Security

- **Full Disk Encryption:** Stores encryption keys for BitLocker, FileVault, LUKS
- **Windows Hello:** Enables password-less authentication using biometrics/PIN
- **Secure Boot + Measured Boot:** Chain of trust from firmware to operating system
- **Device Health Attestation:** Remote verification of device security posture
- **Digital Rights Management (DRM):** Protected media playback
- **Credential Guard (Windows):** Isolates credentials using virtualization-based security

Beyond basic key storage, TPM enables sophisticated security architectures. It's foundational for Zero Trust implementations, providing hardware-based device identity and integrity verification that software alone cannot provide.

Cryptographic Attacks

- **Ciphertext-only:** Attacker has only ciphertext. Goal: find key or plaintext.
- **Known-plaintext:** Attacker has some plaintext-ciphertext pairs.
- **Chosen-plaintext:** Attacker can obtain ciphertext for plaintexts of their choosing.
- **Man-in-the-Middle (MITM):** Intercepts and potentially alters communications.
- **Side-channel:** Attacks based on implementation (timing, power consumption, sound).

Cryptanalysis is the science of breaking cryptosystems. Modern algorithms like AES and RSA are designed to resist all known cryptanalytic attacks when used correctly with sufficient key lengths. Implementation flaws are often the real weakness.

Putting It Together: A Secure Login Example

- **Step 1 (AuthN):** User enters username/password. System verifies against salted hash.
- **Step 2 (Session):** Server generates a random session ID, stores it server-side linked to user identity.
- **Step 3 (Access Control):** For each request, server checks session ID == user identity, then uses RBAC to verify user's role has permission for the requested action.
- **Step 4 (Crypto):** All communication occurs over TLS. Sensitive data (like passwords) is hashed before storage.

This is a typical web application flow. Notice how all three components work in concert. Authentication establishes identity at login. Access control checks that identity for every action. Cryptography protects data in transit and at rest.

Common Pitfalls and Misconceptions

- "We use 256-bit encryption, so we're secure." (Neglecting key management, implementation, and other controls).
- "Our passwords are encrypted." (They should be *salted and hashed*, not encrypted).
- "We use biometrics, so we don't need passwords." (Biometrics are usernames, not secrets).
- "Our firewall is our access control." (Firewalls are a type of network access control; application-layer access control is still needed).

Security is holistic. A strong tool used incorrectly provides no security. Understanding these concepts helps you spot these dangerous misconceptions in the real world.

Summary: The Toolbox in Review

- **Authentication:** Proves identity using factors (knowledge, possession, biometrics).
- **Access Control:** Mediates requests using models (DAC, MAC, RBAC, ABAC) and mechanisms (ACLs, capabilities).
- **Cryptography:** Protects data using symmetric/asymmetric algorithms, hashes, and digital signatures, enabled by PKI.

These are not isolated topics. RBAC defines *who* can do *what*. Cryptography ensures the *who* is authentic and the *what* is protected. Together, they form the technical core of information security.

Looking Ahead: Where Do We Go From Here?

- **Chapter 3:** Programs and Programming – Applying these tools to software security.
- **Chapter 4:** The Web – User-side and server-side security on the internet.
- With our toolbox in hand, we can now examine how to build and break secure systems.

Now that you understand the basic tools, we can explore the complex environments where they are deployed. The next chapters will show you what happens when these tools are used correctly—and incorrectly—in real software and networks.

Critical Thinking Questions (1)

- Why is multi-factor authentication based on "something you know" and "something you have" more secure than two factors of the same type (e.g., two passwords)?
- Explain how a role-based access control (RBAC) system enforces the principle of least privilege more effectively than a simple discretionary access control (DAC) system in a large company.

The first question tests understanding of factor independence. The second asks you to apply the principle from Chapter 1 to a specific access control model.

Critical Thinking Questions (2)

- Describe the steps of a TLS handshake. Identify where symmetric crypto, asymmetric crypto, and PKI (certificates) are used and why each is necessary.
- A developer stores passwords using MD5 hashes without a salt. List the specific attacks this enables and explain how adding a salt prevents them.

These questions force synthesis. The TLS question connects all three toolbox components. The password question moves from abstract concept (hashing) to concrete vulnerability and mitigation.

Hands-On Exercise Suggestions

- Use a command-line tool (like `openssl`) to generate a key pair, create a self-signed certificate, encrypt and decrypt a file, and compute hashes.
- Examine the permission structure (ACLs) on files in your operating system. Create a user and experiment with granting/denying access.
- Set up a simple web app with login and observe the session cookies. Try implementing a simple RBAC system for different user types (admin, user, guest).

Theory is essential, but practice cements understanding. These exercises bridge the gap between concept and implementation. They reveal the complexity and nuance that simple descriptions cannot.

Chapter 2: Key Takeaways

- Authentication establishes identity; MFA is critical for strong authentication.
- Access control enforces policy; choose the model (DAC, MAC, RBAC, ABAC) that fits your security requirements.
- Cryptography provides core security services; understand the difference between symmetric/asymmetric, hashes, and digital signatures.
- These three components are interdependent and form the foundation of all technical security controls.

You now possess the vocabulary and conceptual understanding of the security practitioner's primary tools. In subsequent chapters, we will see attackers relentlessly probing for weaknesses in these very mechanisms.

Questions?

Next: Chapter 3 - Programs and Programming

This was a dense but foundational chapter. Mastery of these concepts is non-negotiable for cybersecurity professionals. Review the terms, understand the interactions, and prepare to apply this knowledge to software security.

Thank you!