CSSLP®

Certified Secure Software Lifecycle Professional

ISC2 Certification

Certification **Exam Outline**

Effective Date: September 15, 2023

ISC2™

About CSSLP

The Certified Secure Software Lifecycle Professional (CSSLP) validates that software professionals have the expertise to incorporate security practices – authentication, authorization and auditing – into each phase of the software development lifecycle (SDLC), from software design and implementation to testing and deployment.

The broad spectrum of topics included in the CSSLP Common Body of Knowledge (CBK®) ensure its relevancy across all disciplines in the field of cybersecurity. Successful candidates are competent in the following eight domains:

- Secure Software Concepts
- Secure Software Lifecycle Management
- Secure Software Requirements
- Secure Software Architecture and Design
- Secure Software Implementation
- Secure Software Testing
- Secure Software Deployment, Operations, Maintenance
- Secure Software Supply Chain

Experience Requirements

A candidate is required to have a minimum of four years of cumulative paid SDLC professional work experience in one or more of the eight domains of the ISC2 CSSLP CBK, or three years of cumulative paid SDLC professional work experience in one or more of the eight domains of the CSSLP CBK with a four-year degree leading to a Baccalaureate, or regional equivalent in Computer Science, Information Technology (IT) or related fields.

If you don't have the required experience to become a CSSLP, you may become an Associate of ISC2 by successfully passing the CSSLP examination. You will then have five years to earn the four years required experience. You can learn more about CSSLP experience requirements and how to account for part-time work and internships at www.isc2.org/Certifications/CSSLP/experience-requirements.

Accreditation

CSSLP is in compliance with the stringent requirements of ANSI/ISO/IEC Standard 17024.

Job Task Analysis (JTA)

ISC2 has an obligation to its membership to maintain the relevancy of the CSSLP. Conducted at regular intervals, the Job Task Analysis (JTA) is a methodical and critical process of determining the tasks that are performed by CSSLP credential holders. The results of the JTA are used to update the examination. This process ensures that candidates are tested on the topic areas relevant to the roles and responsibilities of today's practicing information security professionals.

CSSLP Examination Information

Length of exam	3 hours
Number of items	125
Item format	Multiple choice
Passing grade	700 out of 1000 points
Exam availability	English
Testing center	Pearson VUE Testing Center

CSSLP Examination Weights

Domains	**Weight**
1. Secure Software Concepts	12%
2. Secure Software Lifecycle Management	11%
3. Secure Software Requirements	13%
4. Secure Software Architecture and Design	15%
5. Secure Software Implementation	14%
6. Secure Software Testing	14%
7. Secure Software Deployment, Operations, Maintenance	11%
8. Secure Software Supply Chain	10%
Total:	**100%**

Domain 1:
Secure Software Concepts

1.1 Understand core concepts

- Confidentiality (e.g., encryption)
- Integrity (e.g., hashing, digital signatures, code signing, reliability, modifications, authenticity)
- Availability (e.g., redundancy, replication, clustering, scalability, resiliency)
- Authentication (e.g., multi-factor authentication (MFA), identity & access management (IAM), single sign-on (SSO), federated identity, biometrics)
- Authorization (e.g., access controls, permissions, entitlements)
- Accountability (e.g., auditing, logging)
- Nonrepudiation (e.g., digital signatures, block chain)
- Governance, risk and compliance (GRC) standards (e.g., regulatory authority, legal, industry)

1.2 Understand security design principles

- Least privilege (e.g., access control, need-to-know, run-time privileges, Zero Trust)
- Segregation of Duties (SoD) (e.g., multi-party control, secret sharing, split knowledge)
- Defense in depth (e.g., layered controls, geographical diversity, technical diversity, distributed systems)
- Resiliency (e.g., fail safe, fail secure, no single point of failure, failover)
- Economy of mechanism (e.g., single sign-on (SSO), password vaults, resource efficiency)
- Complete mediation (e.g., cookie management, session management, caching of credentials)
- Open design (e.g., Kerckhoffs's principle, peer review, open source, crowd source)
- Least common mechanism (e.g., compartmentalization/isolation, allow/accept list)
- Psychological acceptability (e.g., password complexity, passwordless authentication, screen layouts, Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA))
- Component reuse (e.g., common controls, libraries)

Domain 2:
Secure Software Lifecycle Management

2.1 Manage security within a software development methodology (e.g., Agile, waterfall)

2.2 Identify and adopt security standards (e.g., implementing security frameworks, promoting security awareness)

2.3 Outline strategy and roadmap

- Security milestones and checkpoints (e.g., control gate, break/build criteria)

2.4 Define and develop security documentation

2.5 Define security metrics (e.g., criticality level, average remediation time, complexity, Key Performance Indicators (KPI), objectives and key results)

2.6 Decommission applications

- End of Life (EOL) policies (e.g., credential removal, configuration removal, license cancellation, archiving, service-level agreements (SLA))
- Data disposition (e.g., retention, destruction, dependencies)

2.7 Create security reporting mechanisms (e.g., reports, dashboards, feedback loops)

2.8 Incorporate integrated risk management methods

- Regulations, standards and guidelines (e.g., International Organization for Standardization (ISO), Payment Card Industry (PCI), National Institute of Standards and Technology (NIST), Open Web Application Security Project (OWASP), Software Assurance Forum for Excellence in Code (SAFECode), Software Assurance Maturity Model (SAMM), Building Security in Maturity Model (BSIMM))
- Legal (e.g., intellectual property, breach notification)
- Risk management (e.g., risk assessment, risk analysis)
- Technical risk vs. business risk

2.9 Implement secure operation practices

- Change management process
- Incident response plan
- Verification and validation
- Assessment and Authorization (A&A) process

Domain 3:
Secure Software Requirements

3.1 Define software security requirements

- Functional (e.g., business requirements, use cases, stories)
- Non-functional (e.g., security, operational, continuity, deployment)

3.2 Identify compliance requirements

- Regulatory authority
- Legal
- Industry-specific (e.g., defense, healthcare, commercial, financial, Payment Card Industry (PCI))
- Company-wide (e.g., development tools, standards, frameworks, protocols)

3.3 Identify data classification requirements

- Data ownership (e.g., data dictionary, data owner, data custodian)
- Data labeling (e.g., sensitivity, impact)
- Data types (e.g., structured, unstructured)
- Data lifecycle (e.g., generation, storage, retention, disposal)
- Data handling (e.g., personally identifiable information (PII), publicly available information)

3.4 Identify privacy requirements

- Data collection scope
- Data anonymization (e.g., pseudo-anonymous, fully anonymous)
- User rights (legal) and preferences (e.g., data disposal, right to be forgotten, marketing preferences, sharing and using third parties, terms of service)
- Data retention (e.g., how long, where, what)
- Cross-border requirements (e.g., data residency, jurisdiction, multi-national data processing boundaries)

3.5 Define data access provisioning

- User provisioning
- Service accounts
- Reapproval process

3.6 Develop misuse and abuse cases

- Mitigating control identification

3.7 Develop security requirement traceability matrix

3.8 Define third-party vendor security requirements

Domain 4: Secure Software Architecture and Design

4.1 Define the security architecture

- Secure architecture and design patterns (e.g., Sherwood Applied Business Security Architecture (SABSA), security chain of responsibility, federated identity)
- Security controls identification and prioritization
- Distributed computing (e.g., client server, peer-to-peer (P2P), message queuing, N-tier)
- Service-oriented architecture (SOA) (e.g., enterprise service bus, web services, microservices)
- Rich internet applications (e.g., client-side exploits or threats, remote code execution, constant connectivity)
- Pervasive/ubiquitous computing (e.g., Internet of Things (IoT), wireless, location-based, Radio-Frequency Identification (RFID), Near Field Communication (NFC), sensor networks, mesh)
- Embedded software (e.g., secure boot, secure memory, secure update)
- Cloud architectures (e.g., Software as a Service (SaaS), Platform as a Service (PaaS), Infrastructure as a Service (IaaS))
- Mobile applications (e.g., implicit data collection privacy)
- Hardware platform concerns (e.g., side-channel mitigation, speculative execution mitigation, secure element, firmware, drivers)
- Cognitive computing (e.g., artificial intelligence (AI), virtual reality, augmented reality)
- Industrial Internet of Things (IoT) (e.g., facility-related, automotive, robotics, medical devices, software-defined production processes)

4.2 Perform secure interface design

- Security management interfaces, out-of-band management, log interfaces
- Upstream/downstream dependencies (e.g., key and data sharing between apps)
- Protocol design choices (e.g., application programming interfaces (API), weaknesses, state, models)

4.3 Evaluate and select reusable technologies

- Credential management (e.g., X.509, single sign-on (SSO))
- Flow control (e.g., proxies, firewalls, protocols, queuing)
- Data loss prevention (DLP)
- Virtualization (e.g., Infrastructure as code (IaC), hypervisor, containers)
- Trusted computing (e.g., Trusted Platform Module (TPM), Trusted Computing Base (TCB))
- Database security (e.g., encryption, triggers, views, privilege management, secure connections)
- Programming language environment (e.g., common language runtime, Java virtual machine (VM), Python, PowerShell)
- Operating system (OS) controls and services
- Secure backup and restoration planning
- Secure data retention, retrieval, and destruction

4.4 Perform threat modeling

- Threat modeling methodologies (e.g., Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege (STRIDE), Process for Attack Simulation and Threat Analysis (PASTA), Hybrid Threat Modeling Method, Common Vulnerability Scoring System (CVSS))
- Common threats (e.g., advanced persistent threat (APT), insider threat, common malware, third-party suppliers)
- Attack surface evaluation
- Threat analysis
- Threat intelligence (e.g., identify credible relevant threats, predict)

4.5 Perform architectural risk assessment and design reviews

4.6 Model (non-functional) security properties and constraints

4.7 Define secure operational architecture (e.g., deployment topology, operational interfaces, Continuous Integration and Continuous Delivery (CI/CD))

Domain 5: Secure Software Implementation

5.1 Adhere to relevant secure coding practices (e.g., standards, guidelines, regulations)

- Declarative versus imperative (programmatic) security
- Concurrency (e.g., thread safety, database concurrency controls)
- Input validation and sanitization
- Error and exception handling
- Output sanitization (e.g., encoding, obfuscation)
- Secure logging & auditing (e.g., confidentiality, privacy)
- Session management
- Trusted/untrusted application programming interfaces (API), and libraries
- Resource management (e.g., compute, storage, network, memory management)
- Secure configuration management (e.g., baseline security configuration, credentials management)
- Tokenization
- Isolation (e.g., sandboxing, virtualization, containerization, Separation Kernel Protection Profiles)
- Cryptography (e.g., payload, field level, transport, storage, agility, encryption, algorithm selection)
- Access control (e.g., trust zones, function permissions, role-based access control (RBAC), discretionary access control (DAC), mandatory access control (MAC))
- Processor microarchitecture security extensions

5.2 Analyze code for security risks

- Secure code reuse
- Vulnerability databases/lists (e.g., Open Web Application Security Project (OWASP) Top 10, Common Weakness Enumerations (CWE), SANS Top 25 Most Dangerous Software Errors)
- Static application security testing (SAST) (e.g., automated code coverage, linting)
- Manual code review (e.g., peer review)
- Inspect for malicious code (e.g., backdoors, logic bombs, high entropy)

5.3 Implement security controls (e.g., watchdogs, file integrity monitoring, anti-malware)

5.4 Address the identified security risks (e.g., risk strategy)

5.5 Evaluate and integrate components

- Systems-of-systems integration (e.g., trust contracts, security testing, analysis)
- Reusing third-party code or open-source libraries in a secure manner (e.g., software composition analysis)

5.6 Apply security during the build process

- Anti-tampering techniques (e.g., code signing, obfuscation)
- Compiler switches
- Address compiler warnings

Domain 6: Secure Software Testing

6.1 Develop security testing strategy & plan

- Standards (e.g., International Organization for Standardization (ISO), Open Source Security Testing Methodology Manual, Software Engineering Institute)
- Functional security testing (e.g., logic)
- Nonfunctional security testing (e.g., reliability, performance, scalability)
- Testing techniques (e.g., known environment testing, unknown environment testing, functional testing, acceptance testing)
- Testing environment (e.g., interoperability, test harness)
- Security researcher outreach (e.g., bug bounties)

6.2 Develop security test cases

- Attack surface validation
- Automated vulnerability testing (e.g., dynamic application security testing (DAST), interactive application security testing (IAST))
- Penetration tests (e.g., security controls, known vulnerabilities, known malware)
- Fuzzing (e.g., generated, mutated)
- Simulation (e.g., simulating production environment and production data, synthetic transactions)
- Failure (e.g., fault injection, stress testing, break testing))
- Cryptographic validation (e.g., pseudorandom number generators, entropy)
- Unit testing and code coverage
- Regression tests
- Integration tests
- Continuous testing
- Misuse and abuse test cases

6.3 Verify and validate documentation (e.g., installation and setup instructions, error messages, user guides, release notes)

6.4 Identify undocumented functionality

6.5 Analyze security implications of test results (e.g., impact on product management, prioritization, break/build criteria)

6.6 Classify and track security errors

- Bug tracking (e.g., defects, errors and vulnerabilities)
- Risk scoring (e.g., Common Vulnerability Scoring System (CVSS))

6.7 Secure test data

- Generate test data (e.g., referential integrity, statistical quality, production representative)
- Reuse of production data (e.g., obfuscation, sanitization, anonymization, tokenization, data aggregation mitigation)

6.8 Perform verification and validation testing (e.g., independent/internal verification and validation, acceptance test)

Domain 7:
Secure Software Deployment, Operations, Maintenance

7.1 Perform operational risk analysis

- Deployment environment (e.g., staging, production, quality assurance (QA))
- Personnel training (e.g., administrators vs. users)
- Legal compliance (e.g., adherence to guidelines, regulations, privacy laws, copyright, etc.)
- System integration

7.2 Secure configuration and version control

- Hardware
- Baseline configuration
- Version control/patching
- Documentation practices

7.3 Release software securely

- Secure Continuous Integration and Continuous Delivery (CI/CD) pipeline (e.g., DevSecOps)
- Application security toolchain
- Build artifact verification (e.g., code signing, hashes)

7.4 Store and manage security data

- Credentials
- Secrets
- Keys/certificates
- Configurations

7.5 Ensure secure installation

- Secure boot (e.g., key generation, access, management)
- Least privilege
- Environment hardening (e.g., configuration hardening, secure patch/updates, firewall)
- Secure provisioning (e.g., credentials, configuration, licensing, Infrastructure as code (IaC))
- Security policy implementation

7.6 Obtain security approval to operate (e.g., risk acceptance, sign-off at appropriate level)

7.7 Perform information security continuous monitoring

- Observable data (e.g., logs, events, telemetry, trace data, metrics)
- Threat intelligence
- Intrusion detection/response
- Regulation and privacy changes
- Integration analysis (e.g., security information and event management (SIEM))

7.8 Execute the incident response plan

- Incident triage
- Forensics
- Remediation
- Root cause analysis

7.9 Perform patch management (e.g. secure release, testing)

7.10 Perform vulnerability management (e.g., tracking, triaging, Common Vulnerabilities and Exposures (CVE))

7.11 Incorporate runtime protection (e.g., Runtime Application Self Protection (RASP), web application firewall (WAF), Address Space Layout Randomization (ASLR), dynamic execution prevention)

7.12 Support continuity of operations

- Backup, archiving, retention
- Disaster recovery plan (DRP)
- Resiliency (e.g., operational redundancy, erasure code, survivability, denial-of-service (DoS))
- Business continuity plan (BCP)

7.13 Integrate service level objectives and service-level agreements (SLA) (e.g., maintenance, performance, availability, qualified personnel)

Domain 8: Secure Software Supply Chain

8.1 Implement software supply chain risk management (e.g., International Organization for Standardization (ISO), National Institute of Standards and Technology (NIST))

» Identification and selection of the components
» Risk assessment of the components (e.g., mitigate, accept)
» Maintaining third-party components list (e.g., software bill of materials)
» Monitoring for changes and vulnerabilities

8.2 Analyze security of third-party software

» Certifications
» Assessment reports (e.g., cloud controls matrix)
» Origin and support

8.3 Verify pedigree and provenance

» Secure transfer (e.g., chain of custody, authenticity, integrity)
» System sharing/interconnections
» Code repository security
» Build environment security
» Cryptographically-hashed, digitally-signed components
» Right to audit

8.4 Ensure and verify supplier security requirements in the acquisition process

» Audit of security policy compliance (e.g., secure software development practices)
» Vulnerability/incident notification, response, coordination, and reporting
» Maintenance and support structure (e.g., community versus commercial, licensing)
» Security track record
» Scope of testing (e.g., shared responsibility model)
» Log integration into security information and event management (SIEM)

8.5 Support contractual requirements (e.g., intellectual property ownership, code escrow, liability, warranty, End-User License Agreement (EULA), service-level agreements (SLA))

Additional Examination Information

Supplementary References

Candidates are encouraged to supplement their education and experience by reviewing relevant resources that pertain to the CBK and identifying areas of study that may need additional attention.

View the full list of supplementary references at www.isc2.org/certifications/References.

Examination Policies and Procedures

ISC2 recommends that CSSLP candidates review exam policies and procedures prior to registering for the examination. Read the comprehensive breakdown of this important information at www.isc2.org/Register-for-Exam.

Legal Info

For any questions related to ISC2's legal policies, please contact the ISC2 Legal Department at legal@isc2.org.

Any Questions?

Contact ISC2 Candidate Services in your region:

Americas

Tel: +1-866-331-4722
Email: membersupport@isc2.org

Asia-Pacific

Tel: +852-5803-5662
China: +86-10-5873-2896
Japan: +81-3-5322-2837
Email: membersupportapac@isc2.org

Europe, Middle East and Africa

Tel: +44 (0)203-960-7800
Email: membersupportemea@isc2.org