6
Security Architecture and Design Principles

Security management, including its goals, security assurance program, and security requirements, were explained in previous chapters. This chapter will discuss security architecture and design principles. For security architects and developers, building software on a mature security framework will greatly reduce not only security risks with industry best practices but also implementation efforts. Therefore, this chapter introduces the key security elements of a cloud service architecture and some mature security frameworks, which can be applied based on the scenario. We will also discuss GDPR and data protection techniques in this chapter.

We will cover the following topics in this chapter:

- Security architecture design principles
- Cloud service security architecture reference (ESAPI)
- Security framework (Shiro, encryption, validation, data masking)
- GDPR and data governance

Security architecture design principles

In this section, we would like to discuss two key concepts, which are security by design and privacy by design. When we discuss security, it's more about the security controls of the whole system such as authentication, authorization, availability, accountability, integrity, and confidentiality. For privacy, it focuses specifically on privacy data or PII (personal identifiable information). Privacy protection is focused on the authorized data handling life cycle and governance.

If we categorize some security controls in general terms, you may find some differences, although there are some overlapping areas in terms of security and privacy:

	Security by design	Privacy by design
Primary concerns	Unauthorized access to the system.	Authorized process of privacy data.
Principles	According to OWASP, security by design principles are the following: • Minimize attack surface area • Establish secure defaults • Principle of least privilege • Principle of defense in depth • Fail securely • Don't trust services • Separation of duties • Avoid security by obscurity • Keep security simple • Fix security issues correctly	Referring to OECD Privacy Principles, the term privacy by design is defined by eight principles: • Collection Limitation Principle • Data Quality Principle • Purpose Specification Principle • Use Limitation Principle • Security Safeguards Principle • Openness Principle • Individual Participation Principle • Accountability Principle
Examples of controls	• Access control • Unsuccessful login attempts • Session control • Timestamps • Non-repudiation • Configuration change control • Audit security events • Cryptographic module • Incident monitoring • Error handling	• Cookie • Anonymity • Consent • Obfuscation • Restrict • Notify and inform • Authentication • Minimization • Separation • Encryption • Data masking

The following industry references may help you to build a secure architecture:

- **Open Security Architecture (OSA) Patterns**: `http://www.opensecurityarchitecture.org/`
- **CSA CAIQ (Consensus Assessment Initiative Questionnaire)**: `https://cloudsecurityalliance.org/group/consensus-assessments`
- **Google VSAQ (Vendor Security Assessment Questionnaires)**: `https://github.com/google/vsaq`
- **PCI Self-Assessment Questionnaire (SAQ)**: `https://www.pcisecuritystandards.org/pci_security/completing_self_assessment`
- **NIST 1500-4 v4 Big Data Interoperability Framework Security and Privacy**: `https://www.nist.gov/publications/nist-big-data-interoperability-framework-volume-4-security-and-privacy`
- **NIST 800-122 Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)**: `https://csrc.nist.gov/publications/detail/sp/800-122/final`

We have understood the concepts and principles of security and privacy. However, the challenges for most organizations are how to build these into applications or services. Therefore, we will discuss some design patterns and also open source framework implementations in upcoming sections.

Cloud service security architecture reference

The **Open Security Architecture** (**OSA**) Patterns **SP-011: Cloud Computing Pattern** and **SP-008: Public Web Server Pattern** provide an overview diagram of the whole system. In addition, **SP-001: client module** and **SP-002 Server module** are also a good reference. Take a look at the components of the cloud computing pattern in the following link: `http://www.opensecurityarchitecture.org/cms/library/patternlandscape/251-pattern-cloud-computing`

In addition, if you are looking for a questionnaire or checklist for self-assessment or for partner security evaluation, here are some suggested references. CSA CAIQ consolidated most security standards (including ISO 27001, FedRAMP, NIST 800-53 R3, and PCI DSS) into a self-assessment questionnaire. VSAQ is mainly for external vendor assessment with the aspects of web application security, security and privacy programs, infrastructure security, and physical and data center security.

- **CSA CAIQ (Consensus Assessment Initiative Questionnaire)**: `https://cloudsecurityalliance.org/group/consensus-assessments/`
- **Google VSAQ (Vendor Security Assessment Questionnaires)**: `https://vsaq-demo.withgoogle.com/`
- **PCI Data Security Standard Self-Assessment Questionnaire (SQA)**: `https://www.pcisecuritystandards.org/documents/SAQ-InstrGuidelines-v3_2.pdf`

Security framework

Architecture principles may still be too high-level for most developers. Therefore, in this section, we will discuss some key open source security frameworks. Depending on the purposes of the security objective and programming languages, there are various kinds of open source security framework. We will only discuss some major or widely used security frameworks.

Adoption of a security framework is the best approach to achieve *secure by design*. A mature security framework provides security controls such as authentication, access control, session management, HTTP security, cryptography, and logging. It also enables a junior developer who has little knowledge of security to build secure software.

Just remember that the security frameworks we will introduce are third-party security components built with our applications. Security applications such as anti-virus software, web application firewalls, and intrusion detection will not be discussed in this section but will be discussed in a later chapter.

Java web security framework

As discussed earlier, the adoption of a web security framework will help us to handle lots of security controls. Take Spring Security as an example—a few edits of the XML configuration will not only provide login/logout form authentication but also CSRF attack, session, and HTTP security header (HSTS, X-content-type, XSS, X-Frame-Options) protection:

Java security framework	Key characteristics
Spring Security	• The Spring Security framework is only for Java- and Spring-based applications. It provides lots of out-of-box security controls such as user authentication, CSRF attack protection, session fixation protection, a HTTP security header, and URL access control. Also, it supports various kinds of authentication such as Oauth2.0, CAS, and OpenID.
Shiro	• Shiro is a more lightweight and simple framework compared to Spring Security. The key difference between Shiro and Spring Security is that Shiro doesn't require a Spring-based application, and it can run standalone without tying into any web framework or a non-web environment.
Object Access Control (OACC)	• OACC primarily provides authentication and authorization. The key characteristic of OACC is that it provides a security relationship with application resources while Spring Security defines authorization by URL, methods, and roles. • A security relationship example definition in OACC may be: (Sara) has (READ, EDIT) permissions on (`TimeSheet.xls`). Being able to establish the application resource (`TimeSheet.xls`) in a security relationship is a unique authorization model in OACC.

For a Java development team, which one is recommended? If the web is built purely on Java Spring, Spring Security is still the best choice due to its powerful security features and complete technical documents. However, if your web applications are running with non-web or non-Spring applications, Shiro is recommended. If your application may need resource access control models, try the OACC.

Non-Java web security frameworks

For non-Java programming, here are some recommendations:

Programming language	Authentication framework
Node.JS	• **Passport framework** is an authentication module for Node.JS.
Ruby on Rails	• **Devise Security:** This is a security module for Ruby. It provides security features such as password complexity, CAPTCHA, user account inactivity checks, verification code, and session control for the web.
ASP.NET	• **ASP.NET Core** provides security features such as authentication, authorization, anti-XSS, SSL enforcement, anti-request forgery, encryption, and also APIs to support GDPR.
Python	• **Yosai** is a security framework for Python applications • **Flask Security:** It provides common security controls to Flask applications such as authentication, password hashing, and role management.

Web readiness for privacy protection

To evaluate the privacy protection readiness for a website, include not only general web security controls but also the following major areas:

- **TLS for secure data transmission**: The misconfiguration of TLS may result in insecure data transmission or man-in-the-middle attacks.
- **Referrer Policy**: The Referrer Policy defines how the browser should handle Referrer information, which reveals the user's original visiting web site. The website visiting history is also considered to be personal privacy information.
- **Cookie Consent Disclaimer**: To comply with the GDPR, the collection of cookie information and the use of any third-party cookies will require explicit cookie consent.
- **HTTP Security Headers**: The HTTP protocol itself provides web security controls. Please also refer to the following table for the suggested HTTP security header configurations.

The following table summarizes the technical parts of privacy security requirements and suggested tools to assess and build the web:

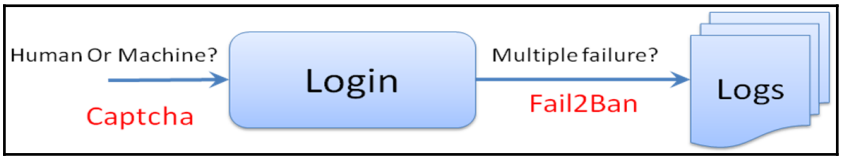
Privacy technical requirements	Tools
Secure Communication: HTTPS by default and secure configuration of TLS.	• SSLyze, SSLScan, and TestSSLServer included in Pentest Box or Kali Linux
The origins of a visiting website source should not be leaked to other websites by the referrer header.	• Referrer Policy defines how the referrer can be used. The configuration of the Referrer Policy depends on the requirements. • no-referrer will ensure the browser never sends the referer header. • If the information is needed, it's suggested to configure sending information over HTTPS by using 'strict-origin'.
If Google Analytics is used, enable privacy extension to anonymize IPs.	• Enable IP masking for Google Analytics
Third-party cookies or embeds services (such as Google Analytics), with user consent.	• Cookie Consent • Cookie Consent JavaScript plug-in: `https://github.com/insites/cookieconsent`
HTTP Security Headers	The following are the suggested mandatory examples of secure http headers. • Content-Security Policy (CSP) "default-src 'self' " • Referrer-Policy "no-referrer" • Strict-Transport-Security "max-age=31536000" • X-content-Type-options "nosniff" • X-Frame-Options "SAMEORGIN" • X-Xss-Protection "1;mode=block" • Cookie "Secure" Refer to OWASP Secure Headers Project for details of each security headers definition.

It's also suggested to build in-house privacy scanning tools for your websites. The following resources provide online scanning services for the privacy requirements mentioned:

Privacy Score Assessment: `https://privacyscore.org`.

Login protection

Login protection can be seen as the first defense layer of the application. Hackers may use tools or APIs to do brute-force login attacks. CAPTCHA is one of the approaches to distinguishing human from machine input. A CAPTCHA requires the client to complete visual-perception tasks. However, the CAPTCHA may be defeated by OCR or unwitting human labor. In addition to CAPTCHA, we can also have another layer of security defense to monitor the number of login failures. If the number of login failures reaches a certain threshold level, the system should take action, such as banning the IP source:



Tools/modules for login protection are summarized in the table:

Login protection techniques	Tools/Modules
Detect the number of login failures in logs and take action	• Fail2Ban
CAPTCHA solution to prevent machine brute-force login attacks	• VisualCaptcha to build your own CAPTCHA service • Google reCAPTCHA

Cryptographic modules

Typical use case cases for cryptographic modules are not only data encryption/decryption, but also SSL/TLS secure communication, key exchange, X509 certificate handling, one-way hashing for message integrity, and random number generation. The recommended encryption modules that the development team may need are shown here:

Encryption module	Adoption scenario
OpenSSL	• Full-featured and most widely used cryptography and SSL/TLS toolkit
Bouncy Castle Crypto APIs	• Lightweight cryptography Java API
mbed TLS	• OpenSSL alternative • Cryptographic and SSL/TLS in embedded products • Cryptography C API
SSLyze	• Verify the secure TLS configuration of the web server

In addition, an operation team may care more about the configuration of encryption on servers such as web servers, SSH, Mail, VPNs, database, proxy, and Kerberos.

Refer to Applied Crypto Hardening: `https://betterCrypto.org/static/applied-crypto-hardening.pdf`.

Input validation and sanitization

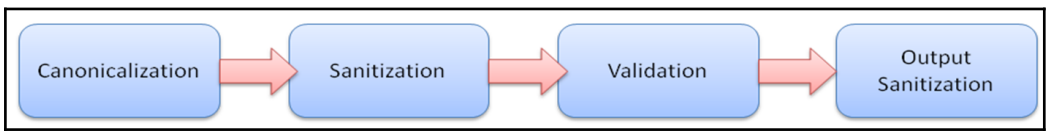
Input validation is like the perimeter security control of the whole application. The input not only includes data input from users but also covers the parameters passing between function calls, methods, APIs, or systems. The concept of validation covers various kinds of technical approaches:

Techniques	Purpose	Example
Canonicalization Normalization	Process input data into known or expected form.	• URL decode/encode • File path or names handling
Sanitization	Sanitization is to remove illegal characters or make potentially risky data safe. Always sanitize an output to avoid XSS.	• Escape: replace < > ' " & with HTML entities.
Validation	To check if the input is valid or within the constraint data type, length, and so on.	• IsAlpha, isCreditCard, isDecimal, isIP

The right order of implementation also matters and reduces the chances of malicious data bypassing the validation. Secure coding requires the following:

- Normalize strings before validating them
- Canonicalize path names before validating them
- Perform any string modifications before validation
- Canonicalize a URL before it is used

When the data is received, the data should be canonicalized first to transform it into expected forms, then the data will be sanitized to remove illegal characters, and the validation may check if it's acceptable based on business rules. Finally, if the data requires output, it always needs to do output sanitization to prevent XSS:



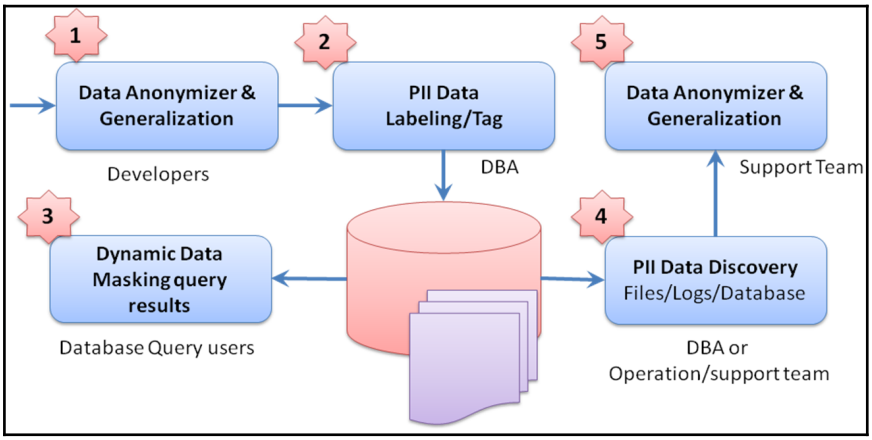
For general canonicalization, sanitization, and validation, we can apply the APIs provided by the mature security framework, while the development team can focus more on business logic validation:

Programming languages	Validation and Sanitization Framework
Java	• OWASP Java HTML Sanitizer
Ruby on Rails	• Active Record Validations
Node.js/JavaScript	• Validators
JavaScript	• DOMPurity to sanitize HTML and prevents XSS attacks
Python	• Cerberus

Data masking

Data masking is the process of obfuscating original/sensitive data to protect it. There are five typical key scenarios that require data masking. Different tools are required based on different roles or usage scenarios:

Scenario	Involved roles	Required tools/modules
• The application receives data and will do data masking based on defined policies	Developer	• Data masking modules • Data masking policies
• Define the PII data tag and access policies	DBA	• PII metadata definition • PII access policies
• Query results with data masking based on defined PII tags and access policies	Data query users	• Dynamic data masking
• The operation team may monitor and check if there is any PII in data, files, configuration, or any unstructured data	Operation team	• PII data discovery
• Any PII in the logs or files must be masked before further processing.	Support team	• Data Anonymizer tools



For data masking techniques, anonymization and pseudonymization are two common categories.

	Anonymization	**Pseudonymization**
Key Difference	Anonymous data cannot be re-identified.	Pseudonymous data is a data substitution which allows for some form of re-identification. Encryption or hashing are the most common techniques in this category.
Data	Anonymization is mainly used for sensitive personal information such as: • Names • IDs (Credit Cards, Social Security numbers and so on.) • Postal addresses • Telephones • Postal codes + cities	• Any data

The table lists common techniques used for data masking, anonymization, and pseudonymization:

Category	**Sub-Category**	**Techniques**	**Application Scenario**
Anonymisation	Randomization	Noise Addition	Numeric data
		Permutation	Numeric data needs to be reversible
		Differential privacy	Big Data statistics
	Generalization	Aggregation	Big Data statistics
		K-anonymity	
		L-diversity	
		T-closeness	
Pseudonymisation		Encryption (AES256)	data needs to be reversible
		Hash (HMAC-SHA256)	Fixed length value
		Tokenization	Keep data format such as ID.

Let's take a telephone number as an example to tell the key differences between anonymization and pseudonymization. If the original value of a telephone number is 12345678, then the anonymization of the number will be 123***** and the pseudonymization of the phone number will be the hash or encrypted value ADF231DADEF. It also means it's impossible for users to know the original value if its anonymization is similar to 123*****. However, it is still likely that the hash or encrypted value can be reversed to the original value if the algorithm is known or enough data samples are collected.

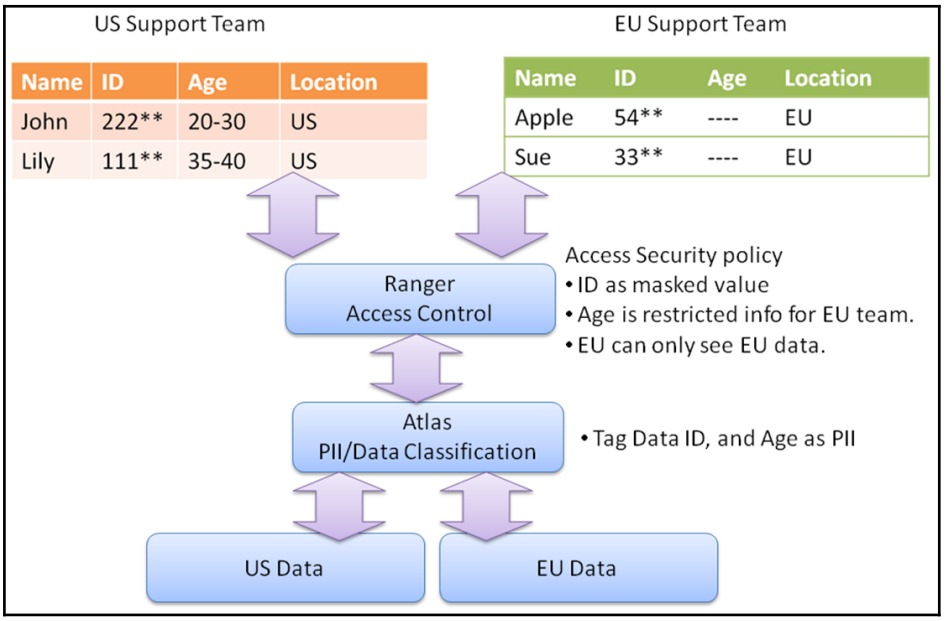
- For the implementation of anonymization and pseudonymization, please also refer to the ARX Data Anonymization Tool: `http://arx.deidentifier.org/`.
- Reference to data masking techniques: `https://www.pdpjournals.com/docs/88197.pdf`

Data governance – Apache Ranger and Atlas

When it comes to data privacy governance, we will need more than just **role-based access control** (**RBAC**) or **attribute-based access control** (**ABAC**) which are common in securing access control. Data governance requires additional metadata or tags to define the data classification, and also row-level attribute-based access control for data masking or row filtering. Take data centers in both the EU and US as an example—we would like to have granular access control policies, as follows:

- US support team can only query US data, and cannot view EU data
- EU support team can only query EU data, and cannot view US data
- The age is considered PII and can only be displayed as a range for the US support team
- The age cannot be displayed to the EU support team
- The ID is PII and will be applied with data masking

This example shows privacy by data is more about the authorized access control of privacy data. The need for techniques such as data governance, data masking, encryption, data classification, and granular ABAC is on the rise due to the usage of big data with cloud services, GDPR compliance, and also the awareness of personal privacy:



You may consider building data privacy governance based on the Apache Ranger and Atlas frameworks. Apache Ranger is mainly for ABAC while Atlas is for data classification.

Third-party open source management

An organization should set up its own internal open source and third-party software database and selection criteria. The database keeps records of open source or in-house developed components adopted in projects. It will provide a good framework selection reference for similar projects such as the web security framework we discussed earlier. If you are looking for an open source component search database, try Open Hub. You may search open source projects and find what you need for the project: `https://www.openhub.net/`. Furthermore, the open source selection criteria help to reduce legal and quality risks.

A typical criteria checklist is listed in the following table:

Selection criteria	Example and description
Does the open source community fix the security issue in a timely manner?	• High-security risks fixed within 1 month.
Adoption of latest and stable releases	• Official and stable release by the community.
Availability of technical support?	• The open source community provides official technical support and issues feedback.
Software licenses with GPL and LGPL are less preferred.	• Licenses with GPL and LGPL are not preferred. If any GPL software components are used, custom-developed source code may also need to be available as open source. • The **binary analysis tool** (**BAT**) is suggested for license scanning based on binary files: `http://www.binaryanalysis.org/`.
Vulnerability status and fixes	• Search for the vulnerability status of the components. For more details, please visit `https://nvd.nist.gov/vuln/search`.
Software release or update frequency	• Was the latest version released within 6 months or several years ago?
Software architecture	• Is it using the latest software technologies or legacy framework?

For the security of open source components, the recommended security practices and tools during the DevOps stages are summarized in the table:

Stage	Activities	Recommended Tools/Practices
Design and Selection	Selection of Open Source.	• `www.openHub.Net` • Open Source selection checklist • In-house Open source database
Package Delivery	Identify all the dependencies in the project and check known vulnerabilities.	• OWASP dependency check • OWASP dependency Track
Package Deployment	On-line services monitoring and scanning of CVE.	• CVE database (`https://nvd.nist.gov/vuln/search`) • NMAP or OpenVAS scanning

Also, refer to SAFECode Managing Security Risks Inherent in the Use of Third-party Components: `https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf`.

Summary

We discussed security architecture design principles including the clarification of security by design and privacy by design. Security by design is focused on **confidentiality, integrity, and availability** (**CIA**) and design by privacy is more about the protection of privacy data. The industry-standard CSA, Google, PCI, or NIST provide good references. We can also refer to the OSA cloud computing pattern to understand the whole security architecture of a cloud service.

To build a security framework, we list some open source security frameworks to achieve some security controls instead of reinventing the wheel. For example, there is Spring Security and Shiro for web security frameworks in Java, and the Password Framework for NodeJS.

When it comes to website privacy protection, we discussed what is required legally, such as copyright notices, cookies, disclaimers, and data protection notices. We listed key security technical controls for web privacy.

We also discussed login protection modules such as Fail2Ban and reCAPTCHA, and cryptographic modules (OpenSSL, SSLyze). We explained the concept of input validation including normalization, sanitization, and validation. To protect sensitive data, the scenario of data masking and techniques (anonymization, pseudonymization) were explained. Data governance with Apache Ranger and Atlas frameworks was explained to classify and mask sensitive data. With lots of third-party components and security framework components introduced, we also suggested how an organization should manage third-party open source software.

In the next chapter, we will discuss threat modeling and secure design security practices in more detail.

Questions

1. Which of the following is one of the security by design principles?
    1. Establish secure defaults
    2. Least Privilege
    3. Fail securely
    4. All of the above

2. Which one of the following references consolidated most security standards such as ISO, FedRAMP, and NIST?
    1. CSA CAIQ
    2. Google VSAQ
    3. PCI DSS
    4. (OSA) Open Security Architecture Patterns
3. What security protection does a Spring Security framework provide?
    1. Authentication
    2. CSRF attack protection
    3. HTTP security headers
    4. All of the above
4. What's the key difference between Shiro and Spring Security?
    1. Shiro doesn't require a Java Spring framework
    2. Logging
    3. Encryption
    4. Intrusion defense
5. Which one of the followings may apply to the Passport Framework?
    1. ASP .NET
    2. Node.JS
    3. Ruby on Rails
    4. Python
6. Which one of the following cryptographic modules is specially used for embedded applications?
    1. OpenSSL
    2. Mbed TLS
    3. SSLyze
    4. Fail2Ban
7. Which one of these is an example of sanitization?
    1. Process input data into known or expected form
    2. Check if the input is valid
    3. Remove illegal characters
    4. Check the data type

Further reading

- **Privacy by Design, the 7 Foundational Principles**: https://ipc.on.ca/wp-content/uploads/Resources/7foundationalprinciples.pdf
- **NIST 800-53 Rev.4 Security and Privacy Controls for Federal Information Systems and Organizations**: https://www.nist.gov/publications/security-and-privacy-controls-federal-information-systems-and-organizations-including-0
- **NIST SP800-30 Rev 1 Guide for Conducting Risk Assessments**: https://csrc.nist.gov/publications/detail/sp/800-30/rev-1/final
- **NIST SP 800-12 Rev 1 An introduction to Information Security**: https://csrc.nist.gov/publications/detail/sp/800-12/rev-1/final
- **SP 800-39 Managing Information Security Risk: Organization, Mission and Information System View**: https://csrc.nist.gov/publications/detail/sp/800-39/final
- **SP 800-37 Rev 1 Guide for Applying the Risk Management Framework to Federal Information Systems: a Security life Cycle Approach**: https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final
- **Privacy Pattern**: https://privacypatterns.org/patterns
- **Open Reference Architecture for Security and Privacy**: https://media.readthedocs.org/pdf/security-and-privacy-reference-architecture/latest/security-and-privacy-reference-architecture.pdf
- **OSA (Open Security Architecture) Patterns**: www.opensecurityarchitecture.org/cms/library/patternlandscape
- **Google VSAQ Vendor Security Assessment Questionnaires**: https://github.com/google/vsaq
- **Hadoop Data security**: https://docs.hortonworks.com/HDPDocuments/HDP2/HDP-2.6.4/bk_security/content/ch_hdp-security-guide-overview.html
- **Cryptographic Key Length Recommendation**: www.keylength.com/
- **SAFECode Managing Security Risks Inherent in the Use of Third-party Components**: https://www.safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf
- **Practices for Secure Development of Cloud Applications**: https://safecode.org/wp-content/uploads/2018/01/SAFECode_CSA_Cloud_Final1213.pdf
- **OECD Privacy Principles** http://oecdprivacy.org/