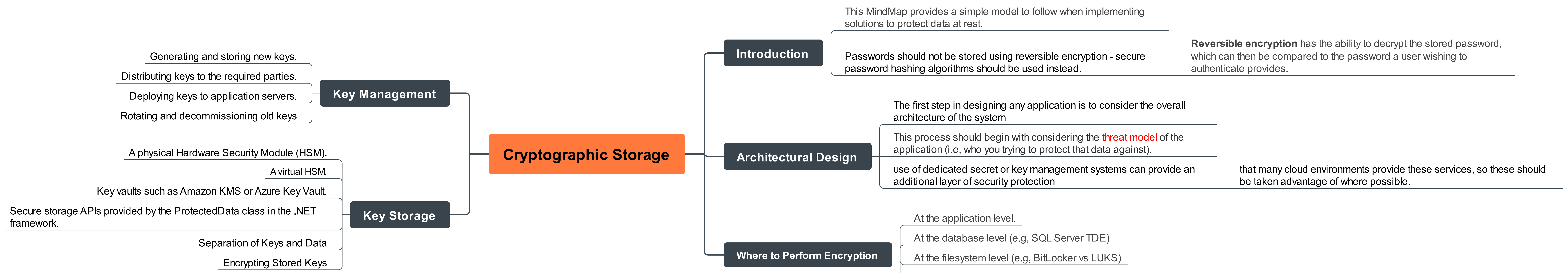
# 2 - Cryptographic Failures

- **Overview**
  - **Cryptographic Failures = Sensitive Data Exposure**
    - The focus is on failures related to cryptography (or lack thereof), Which often lead to exposure of sensitive data
  - **Common Weakness**
    - **CWE-259: Use of Hard-coded Password**
      - Hardcoded Passwords: referred to as Embedded Credentials, are plain text passwords or other secrets in source code.
    - **CWE-327: Broken or Risky Crypto Algorithm**
      - is an unnecessary risk that may result in the exposure of sensitive information.
      - attacker may be able to break the algorithm and compromise whatever data has been protected.
      - Example
        - DES
          - `$encryptedPassword = mcrypt_encrypt(MCRYPT_DES, $key, $password, MCRYPT_MODE_ECB, $iv);`
    - **CWE-331: Insufficient Entropy .**
      - **Description**
        - The software uses an algorithm or scheme that produces insufficient entropy, leaving patterns or clusters of values that are more likely to occur than others.
        - Insufficient Entropy : refers to the initial internal state or seed of a PRNG being so limited that it or the PRNG's actual output is restricted to a more easily brute forcible range of possible values.
          - **pseudorandom number generator (PRNG)**
            - for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers.
        - Example
          - This code generates a unique random identifier for a user's session.

```
function generateSessionID($userID){
  srand($userID);□
  return rand();
}
```

- **Description**
  - **EU's General Data Protection Regulation (GDPR), or regulations**
    - **e.g**
      - PCI Data Security Standard (PCI DSS).
    - **For all such data**
      - Is any data transmitted in clear text?
        - as HTTP, SMTP, FTP also using TLS upgrades like STARTTLS.
      - Are any old or weak cryptographic algorithms or protocols used either by default or in older code?
      - Are default crypto keys in use, weak crypto keys generated or re-used, or is proper key management or rotation missing?
      - Is encryption not enforced, e.g., are any HTTP headers (browser) security directives or headers missing?
      - Is the received server certificate and the trust chain properly validated?
      - Are passwords being used as cryptographic keys in absence of a password base key derivation function?
      - Are deprecated hash functions such as MD5 or SHA1 in use, or are non-cryptographic hash functions used when cryptographic hash functions are needed?
      - Are deprecated cryptographic padding methods such as PKCS number 1 v1.5 in use?
      - re cryptographic error messages or side channel information exploitable, for example in the form of padding oracle attacks?
- **Examples**
  - Cryptographic Storage
  - Transport Layer Protection
  - HTTP Strict Transport Security
  - TLS Cipher String
  - Key Management
  - Pinning
- **Attack Scenarios**
  - **Scenario 1**
    - An application encrypts credit card numbers in a database using automatic database encryption.
    - However, this data is automatically decrypted when retrieved
    - allowing a SQL injection flaw to retrieve credit card numbers in clear text.
  - **Scenario 2**
    - A site doesn't use or enforce TLS for all pages or supports weak encryption.
    - An attacker monitors network traffic
  - **Scenario 3**
    - The password database uses unsalted or simple hashes to store everyone's passwords.
    - A file upload flaw allows an attacker to retrieve the password database.
    - Hashes generated by simple or fast hash functions may be cracked by GPUs, even if they were salted.
- **How to Prevent**
  - Don't store sensitive data unnecessarily. Discard it as soon as possible or use PCI DSS compliant tokenization or even truncation. Data that is not retained cannot be stolen.
  - Make sure to encrypt all sensitive data at rest.
  - Ensure up-to-date and strong standard algorithms, protocols, and keys are in place; use proper key management.
  - Disable caching for response that contain sensitive data.
  - Do not use legacy protocols such as FTP and SMTP for transporting sensitive data.
  - Store passwords using strong adaptive and salted hashing functions with a work factor (delay factor), such as Argon2, scrypt, bcrypt or PBKDF2.



# Cryptographic Storage

- Key Management
  - Generating and storing new keys.
  - Distributing keys to the required parties.
  - Deploying keys to application servers.
  - Rotating and decommissioning old keys
- Key Storage
  - A physical Hardware Security Module (HSM).
  - A virtual HSM.
  - Key vaults such as Amazon KMS or Azure Key Vault.
  - Secure storage APIs provided by the ProtectedData class in the .NET framework.
  - Separation of Keys and Data
  - Encrypting Stored Keys
- Introduction
  - This MindMap provides a simple model to follow when implementing solutions to protect data at rest.
  - Passwords should not be stored using reversible encryption - secure password hashing algorithms should be used instead.
    - **Reversible encryption** has the ability to decrypt the stored password, which can then be compared to the password a user wishing to authenticate provides.
- Architectural Design
  - The first step in designing any application is to consider the overall architecture of the system
  - This process should begin with considering the threat model of the application (i.e, who you trying to protect that data against).
  - use of dedicated secret or key management systems can provide an additional layer of security protection
    - that many cloud environments provide these services, so these should be taken advantage of where possible.
- Where to Perform Encryption
  - At the application level.
  - At the database level (e.g, SQL Server TDE)
  - At the filesystem level (e.g, BitLocker vs LUKS)
  - At the hardware level (e.g, encrypted RAID cards or SSDs)

# Transport Layer Protection

- Introduction
  - This MindMap provides guidance on how to implement transport layer protection for an application using Transport Layer Security (TLS).
    - Confidentiality
    - Integrity
    - Replay prevention
      - protection against an attacker replaying requests against the server.
    - Authentication
  - SSL vs TLS
    - There were two publicly released versions of SSL - versions 2 and 3
      - Both of these have serious cryptographic weaknesses and should no longer be used.
    - For various reasons the next version of the protocol (effectively SSL 3.1) was named Transport Layer Security (TLS) version 1.0.
    - The terms "SSL", "SSL/TLS" and "TLS" are frequently used interchangeably, and in many cases "SSL" is used when referring to the more modern TLS protocol.
    - Secure Socket Layer (SSL) was the original protocol that was used to provide encryption for HTTP traffic
- Server Configuration
  - Only Support Strong Protocols
    - General purpose web applications should only support TLS 1.2 and TLS 1.3, with all other protocols disabled.
    - Where it is known that a web server must support legacy clients with unsupported an insecure browsers
    - it may be necessary to enable TLS 1.0 to provide support.
    - that PCI DSS forbids the use of legacy protocols such as TLS 1.0.
  - Only Support Strong Ciphers
    - There are a large number of different ciphers (or cipher suites) that are supported by TLS, that provide varying levels of security.
    - Where possible, only GCM ciphers should be enabled.
    - types of ciphers should always be disabled:
      - Null ciphers
      - Anonymous ciphers
      - EXPORT ciphers
  - Use Strong Diffie-Hellman Parameters
    - generate 2048 bit parameters:
      - openssl dhparam 2048 -out dhparam2048.pem
    - The Weak DH website provides guidance on how various web servers can be configured to use these generated parameters.
  - Disable Compression
    - TLS compression should be disabled in order to protect against a vulnerability (nicknamed CRIME)
      - which could potentially allow sensitive information such as session cookies to be recovered by an attacker.
  - Patch Cryptographic Libraries
  - Test the Server Configuration
    - SSL Labs Server Test
    - CryptCheck
    - CipherCraft
    - Hardenize
    - ImmuniWeb
    - Observatory by Mozilla Scanigma
- Certificates
  - Use Strong Keys and Protect Them
  - Use Strong Cryptographic Hashing Algorithms
  - Use Correct Domain Names
  - Carefully Consider the use of Wildcard Certificates
  - Use an Appropriate Certification Authority for the Application's User Base
  - Use CAA Records to Restrict Which CAs can Issue Certificates
  - Always Provide All Needed Certificates
  - Consider the use of Extended Validation Certificates
- Application
  - Use TLS For All Pages
  - Do Not Mix TLS and Non-TLS Content
    - A page that is available over TLS should not include any resources (such as JavaScript or CSS) files which are loaded over unencrypted HTTP.
    - These unencrypted resources could allow an attacker to sniff session cookies or inject malicious code into the page.
    - Modern browsers will also block attempts to load active content over unencrypted HTTP into secure pages.
  - Use the "Secure" Cookie Flag
  - Prevent Caching of Sensitive Data
    - HTTP headers
      - Cache-Control: no-cache, no-store, must-revalidate<br>Pragma: no-cache<br>Expires: 0
  - Use HTTP Strict Transport Security
    - Introduction
      - HTTP Strict Transport Security (also named HSTS) is an opt-in security enhancement that is specified by a web application through the use of a special response header.
      - Once a supported browser receives this header that browser will prevent any communications from being sent over HTTP to the specified domain and will instead send all communications over HTTPS.
    - Threats
      - User bookmarks or manually types http://example.com and is subject to a man-in-the-middle attacker
        - HSTS automatically redirects HTTP requests to HTTPS for the target domain
    - Examples
      - Strict-Transport-Security: max-age=31536000
      - This example is useful if all present and future subdomains will be HTTPS. This is a more secure option but will block access to certain pages that can only be served over HTTP:
        - Strict-Transport-Security: max-age=31536000; includeSubDomains
      - This example is useful if all present and future subdomains will be HTTPS. In this example we set a very short max-age in case of mistakes during initial rollout:
        - Strict-Transport-Security: max-age=86400; includeSubDomains
    - HTTP Strict Transport Security (HSTS) instructs the user's browser to always request the site over HTTPS, and also prevents the user from bypassing certificate warnings.
  - Consider the use of Client-Side Certificates
  - Consider Using Public Key Pinning
    - What Is Pinning
      - Pinning is the process of associating a host with their expected X509 certificate or public key.
    - Public key pinning can be used to provides assurance that the server's certificate is not only valid and trusted, but also that it matches the certificate expected for the server.
    - This provides protection against an attacker who is able to obtain a valid certificate
    - Public key pinning was added to browsers in the HTTP Public Key Pinning (HPKP) standard.