Cryptographic Failures

Group No: 05

24369 | 24394 | 24429

Contents

Introduction

Cryptography is a method of securing communication so only authorized parties can access the information. It involves converting readable data (plaintext) into an unreadable format (ciphertext) using encryption algorithms. Simply it's the science of protecting information by transforming into a particular unreadable format that unauthorized users will be unable to clearly understand or use the information.

Cryptography includes essential points such as:

- **Encryption** - Converts readable data into unreadable ciphertext
- **Decryption** – Restores the unreadable text back to the readable data by the authorized user using secret key
- **Key** - A secret piece of information used to encrypt and decrypt data.
- **Algorithms** – Mathematical rules or methods used for encryption and decryption.

Cryptographic failures occur when the mechanisms which was designed to secure data and communications through encryption break down, become compromised, or fail to perform as expected.

Cryptographic failures can arise from various reasons such as flawed or insecure algorithms, poor implementation, weak / improper key management, weak encryption, misconfigured protocols and insider threats and inadequate security awareness leading to inadequate protection of sensitive information.

When a cryptographic mechanism fails, it allows attackers to exploit vulnerabilities to breach systems, decrypt sensitive personal data, manipulate information, or impersonate users etc. Therefore these cryptographic failures may cripple the basis of data security, namely the confidentiality, integrity, and authenticity.

Thereby a cryptographic failure is not simply a technical glitch it represents a severe breakdown in the processes meant to protect information.

Reasons for the Cryptographic Failures

1. Weak or Outdated Encryption Algorithms

Encryption algorithms are the foundation of cryptographic security. However, algorithms that were once considered secure eventually become obsolete due to advances in computing power, cryptanalysis techniques, and new attack methods.

For example:

- **DES (Data Encryption Standard) -** Once widely used, but now broken by brute force since its 56-bit key space is too small for modern computers.
- **MD5 & SHA-1 -** Commonly used for hashing, but proven vulnerable to collision attacks (two different inputs producing the same hash). Attackers can forge certificates or files by exploiting this weakness.
- **RC4 -** Once popular in SSL/TLS, but vulnerable to bias attacks that reveal parts of plaintext.
- **SSL 2.0/3.0 and TLS 1.0/1.1 -** Outdated transport protocols susceptible to downgrade and man-in-the-middle attacks.

Continuing to use these outdated standards creates a false sense of security, as encrypted data can be cracked with affordable cloud resources today.

Identification Methods

- **Static Analysis Tools (SAST) -** Detect hardcoded use of weak ciphers (e.g., MD5, SHA1, DES).
- **Penetration Testing -** Check server configurations for insecure TLS versions or weak cipher suites.
- **Vulnerability Scanners (e.g., Nessus, Qualys) -** Automatically flag deprecated protocols and algorithms.
- **Code Review -** Look for insecure default libraries or cryptographic calls.

2. Poor Key Management

Keys are the most sensitive component of cryptographic systems. Even with strong algorithms, improper key handling can nullify security. Common issues include:

- **Hardcoded Keys -** Developers embed keys directly in source code or configuration files, making them easy to extract.

- **Weak Keys -** Using predictable or short keys makes brute-force attacks feasible.
- **Unprotected Storage -** Keys stored in plaintext on disk, in environment variables, or in Git repositories can be stolen.
- **Improper Distribution -** Sending keys over insecure channels (e.g., email, FTP) increases exposure risk.

Real-world example: In 2013, Adobe's breach exposed 150M records partly because poorly secured cryptographic keys were stolen.

Identification Methods

- **Automated Scanning (SonarQube, TruffleHog) -** Detect hardcoded secrets in repositories.
- **Security Audits -** Evaluate policies for key rotation, revocation, and use of secure storage (HSM, KMS).
- **Code Review -** Verify keys are retrieved from secure vaults instead of being hardcoded.
- **Penetration Testing -** Attempt to extract keys from memory dumps, logs, or files.

3. Implementation Errors in Cryptographic Functions

Even when secure algorithms are chosen, cryptography can fail due to developer mistakes or poor integration. Common pitfalls include:

- **Insecure Modes of Operation -** Using ECB (Electronic Codebook) with AES instead of secure modes like CBC or GCM can leak patterns in the encrypted data.
- **Weak Random Number Generation -** Using non-cryptographic RNGs (e.g., rand()) can make keys predictable.
- **Improper Certificate Validation -** Accepting self-signed or expired certificates makes TLS vulnerable to MITM attacks.

These mistakes often stem from developers treating cryptography as a "black box" without understanding nuances, which is why security libraries and proper reviews are critical.

Identification Methods

- **Manual Code Review -** Verify cryptographic libraries are used correctly (AES-GCM instead of AES-ECB, proper IV handling).
- **DAST Tools -** Detect TLS/SSL misconfigurations (missing certificate validation, weak ciphers).

- **Unit & Integration Tests -** Test random number generators and cryptographic flows under varied conditions.
- **Penetration Testing -** Attempt padding oracle, timing, and replay attacks.

4. Lack of Encryption or Misconfiguration

One of the simplest but most dangerous failures is not encrypting sensitive data at all or incorrectly configuring encryption. Examples include:

- **Data in Transit -** Sending passwords, credit card numbers, or medical records over HTTP, FTP, or SMTP without TLS exposes them to interception.
- **Data at Rest -** Storing database records, backups, or log files in plaintext makes breaches devastating.
- **Certificate Misconfiguration -** Using self-signed, expired, or mismatched certificates makes encrypted traffic unreliable.
- **Insecure Defaults -** Systems often ship with weak settings (e.g., TLS disabled by default).

This is especially risky under compliance laws (e.g., GDPR, HIPAA, PCI DSS) that require encryption of sensitive data both at rest and in transit.

Identification Methods

- **Network Scanning (Wireshark, Burp Suite) -** Check if sensitive data is transmitted in cleartext.
- **Config Audits -** Verify HTTPS enforcement, TLS certificates, and HSTS headers.
- **Database Review -** Check if stored data (passwords, financial info) is encrypted with secure algorithms.
- **Automated Security Scanners -** Flag missing HTTPS or misconfigured certificates.

5.Insider Threats and Inadequate Security Awareness

Even with strong algorithms and configurations, human factors can cause cryptographic failures.

- **Insider Threats -** Employees or contractors with privileged access might intentionally leak or misuse cryptographic keys.
- **Poor Security Training -** Developers might unintentionally misconfigure cryptography, e.g., using weak defaults or skipping certificate validation.

- **Improper Handling of Secrets -** Admins might store passwords in spreadsheets or email keys insecurely.
- **Social Engineering -** Attackers can trick insiders into revealing sensitive cryptographic material.

Identification Methods

- **Insider Threat Monitoring -** Track abnormal key access behavior in logs.
- **Security Awareness Programs -** Train staff on secure handling of cryptographic material.
- **Regular Audits & Compliance Checks -** Ensure developers follow best practices in key management.
- **Access Control Policies -** Enforce least privilege for cryptographic systems.

How to prevent Cryptographic Failures

1. Using Strong and Modern Cryptographic Algorithms

The security of any cryptographic system depends on the strength and reliability of the algorithms it uses. Weak, outdated, or homegrown algorithms often contain vulnerabilities that attackers can exploit. By contrast, well-established modern algorithms have been thoroughly analyzed and tested by the security community, providing strong guarantees of confidentiality, integrity, and authenticity. To prevent cryptographic failures, organizations must carefully choose algorithms that are still considered secure by current industry standards.

- **Encryption** - Use AES-256 for symmetric encryption.
- **Hashing** - Use SHA-256 or stronger for ensuring integrity.
- **Asymmetric Encryption** - Use RSA (2048/4096-bit with secure padding) or Elliptic Curve Cryptography (ECC) for key exchange and digital signatures.
- **Password Hashing** - Use algorithms like bcrypt, scrypt, or Argon2 with proper salting and iteration counts. Avoid MD5, SHA-1, or unsalted hashes.
- **Random Numbers** - Use cryptographically secure pseudorandom number generators (CSPRNGs) for keys, tokens, and session IDs. Avoid weak random functions such as rand() or math.random().

2. Securing Transport and Communication

Data must be protected not only while stored but also while moving between systems. Transport security ensures that information transmitted over networks cannot be intercepted, modified, or stolen by attackers. This requires the use of secure communication protocols and hardened configurations that enforce confidentiality and integrity for all data in transit.

- **Protocols** - Always use HTTPS with TLS 1.2 or higher.
- **Weak Protocols** - Disable outdated versions such as SSL, TLS 1.0, and TLS 1.1.
- **Certificates** - Use valid and up-to-date digital certificates from trusted certificate authorities (CAs).
- **Cipher Suites** - Enforce strong cipher suites and disable insecure ones (e.g., RC4, DES, 3DES).
- **HSTS** - Enable HTTP Strict Transport Security (HSTS) headers to enforce secure connections and prevent protocol downgrades.

3. Proper Key Management

Strong encryption is useless without proper management of cryptographic keys. Keys must remain confidential, well-protected, and rotated regularly to limit the damage in case of compromise. Poor practices such as hardcoding keys in application code or storing them in plaintext files expose sensitive systems to serious risks. Secure key management ensures that keys are created, stored, used, and retired in a way that upholds the overall security of the cryptographic system.

- **Key Storage** - Store keys securely in a Key Management System (KMS) or Hardware Security Module (HSM).
- **Rotation** - Rotate keys periodically to reduce exposure in case of compromise.
- **Access Control** - Limit key access to authorized users and services only.
- **Generation** - Generate keys using strong entropy sources to prevent predictability.
- **Revocation** - Implement clear revocation procedures for compromised or expired keys.

4. Encrypting Sensitive Data

Sensitive information such as personal details, financial records, or healthcare data must be protected at all times. Encryption safeguards data both when it is stored and when it is transmitted, ensuring that even if unauthorized individuals gain access to it, the information

remains unreadable. This dual-layer approach is critical for meeting regulatory requirements and maintaining user trust.

- **Data at Rest** - Encrypt databases, file systems, and backups using AES-256.
- **Data in Transit** - Secure all traffic with TLS, IPsec, or SSH.
- **Field-Level Encryption** - Apply encryption at the field level for highly sensitive values such as passwords, credit card numbers, and medical records.
- **Backups** - Ensure that backup files are encrypted and stored securely.
- **Key Use** - Always pair encryption with proper key management to maintain effectiveness.

5. Robust Implementation Practices

Even when the correct algorithms and protocols are chosen, cryptographic security can fail if the implementation is flawed. Many breaches have occurred not because of weak algorithms, but due to mistakes in how they were applied. Developers must avoid writing their own cryptographic code, since even minor errors in padding, random number generation, or protocol handling can create severe vulnerabilities. Instead, organizations should rely on mature, well-tested cryptographic libraries and follow secure coding standards to ensure correctness and reliability.

- **Avoid Custom Code** - Do not implement encryption or hashing functions from scratch.
- **Libraries** - Use established libraries such as OpenSSL, Bouncy Castle, or Java's javax.crypto.
- **Secure Defaults** - Configure libraries with secure defaults instead of weak backward-compatible options.
- **Coding Standards** - Follow secure development guidelines to avoid introducing flaws.
- **Side-Channel Resistance** - Implement protections against timing attacks and side-channel leaks.

6. Thorough Security Testing

Testing is a critical step in ensuring cryptographic mechanisms function as intended and are not vulnerable to exploitation. Functional testing verifies that encryption, decryption, and hashing processes behave correctly. Security testing goes further by examining potential flaws such as weak implementations, side-channel vulnerabilities, and misconfigurations. By conducting

penetration testing, fuzzing, and cryptographic audits, organizations can identify and fix weaknesses before attackers exploit them.

- **Functional Testing** - Verify that cryptographic operations (encryption, decryption, signing, verification) perform correctly.
- **Penetration Testing** - Assess the system for weaknesses that attackers could exploit.
- **Fuzz Testing** - Use fuzzing tools to uncover unexpected errors in cryptographic code.
- **Side-Channel Analysis** - Test for vulnerabilities such as timing, power, or cache-based attacks.
- **Audits** - Conduct regular cryptographic audits and code reviews.

7. Ongoing Security Monitoring

Cryptographic security is not a one-time effort; it requires continuous monitoring to remain effective. Threats evolve quickly, and newly discovered vulnerabilities in algorithms or libraries can make systems insecure. Organizations must monitor for expired or misconfigured certificates, apply updates and patches promptly, and track cryptographic usage across their systems. Security monitoring also includes logging and auditing cryptographic events to detect unauthorized activity.

- **Certificate Monitoring** - Track SSL/TLS certificates to prevent expiration or misconfiguration.
- **Patch Management** - Apply security patches and updates to cryptographic libraries immediately.
- **System Monitoring** - Monitor systems for signs of breach or unusual cryptographic usage.
- **Auditing** - Maintain logs of cryptographic operations for security audits.
- **Threat Intelligence** - Stay informed about new cryptographic vulnerabilities and act quickly to mitigate risks.

8. Defense in Depth

Cryptography is only one layer of a secure system, and relying on it alone is not sufficient. A defense-in-depth strategy ensures that even if one layer is compromised, additional protections remain in place to safeguard sensitive data. This approach combines cryptographic protection with broader security practices such as network segmentation, access control, intrusion detection,

and multi-factor authentication. The goal is to create a resilient security posture where multiple overlapping controls reduce the overall risk of failure.

- **Layered Controls** - Combine cryptography with firewalls, IDS/IPS, and access controls.
- **Network Segmentation** - Separate critical systems from general networks.
- **Access Restrictions** - Enforce least privilege of access for users and services.
- **Monitoring Tools** - Deploy intrusion detection and prevention systems (IDS/IPS).

Multi-Factor Authentication (MFA) - Strengthen identity verification beyond passwords.

How Organization use these prevention techniques in practice

1. Strong Cryptographic Algorithms

Use AES-256 for encryption, RSA/ECC for key exchange, and SHA-256 or stronger for hashing.

- Google Drive & Dropbox use AES-256 to encrypt files stored on their servers. This ensures that even if someone breaks into their storage backend, files are unreadable without decryption keys.
- Microsoft Windows Updates are signed using RSA 2048/4096-bit digital signatures. This prevents tampered or malicious updates from being installed on user devices.
- Bitcoin and Ethereum rely on SHA-256 and Keccak-256 hashing for transaction integrity and proof-of-work security.

2. Secure Transport and Communication

Always use HTTPS with TLS 1.2+ and configure SSL/TLS properly with strong certificates.

- Banks like HSBC and PayPal enforce TLS 1.3 across all web traffic. If a user tries to connect via HTTP, they are immediately redirected to HTTPS with HSTS (HTTP Strict Transport Security), which blocks downgrade attacks.
- WhatsApp uses the Signal Protocol for end-to-end encryption in messaging. Even WhatsApp's own servers cannot read the communication between two users, since transport security is layered with cryptographic session keys.
- Google Chrome & Mozilla Firefox no longer trust SHA-1 certificates, forcing websites to adopt modern TLS certificates to maintain secure browsing.

3. Proper Key Management

Keys must be stored securely (not hardcoded), rotated, and managed using specialized systems.

- Amazon Web Services (AWS) uses AWS Key Management Service (KMS) to generate, store, and rotate encryption keys automatically. Applications using AWS services never directly handle raw keys; instead, they call the KMS API to encrypt or decrypt data.
- Apple iCloud employs a Hardware Security Module (HSM) for encryption keys. These keys are never exposed outside Apple's secure environment, making it extremely difficult for attackers to steal them.
- Netflix uses HashiCorp Vault to manage its API keys, database passwords, and encryption keys. This prevents secrets from being leaked into code repositories or logs.

4. Password Protection with Modern Hashing

Use algorithms like bcrypt, scrypt, or Argon2 for hashing, with proper salting and iteration counts.

- Facebook stores passwords hashed with scrypt, ensuring computational difficulty for attackers. Even if a password database leak occurs, brute-forcing is impractical.
- GitHub migrated to bcrypt for password storage to protect user accounts after large-scale password leaks from other services.
- ProtonMail uses Argon2 hashing for passwords. Even ProtonMail itself cannot read user passwords due to client-side hashing before transmission.

5. Robust Implementation Practices

Avoid custom cryptographic implementations; use well-tested libraries and secure coding practices.

- Google Chrome relies on BoringSSL (a fork of OpenSSL maintained by Google) to implement TLS securely, avoiding the pitfalls of custom cryptography.
- OpenSSL is widely used in web servers like Apache and Nginx; organizations rely on its vetted implementation rather than writing custom encryption code.
- Apple uses its CommonCrypto library across macOS and iOS, ensuring developers use secure, tested cryptographic functions instead of risky custom algorithms.

6. Thorough Security Testing

Perform functional and security testing, penetration testing, fuzzing, and audits to detect vulnerabilities in cryptographic systems.

- Microsoft conducts extensive internal cryptography audits and penetration tests before releasing Windows updates to ensure encryption and authentication mechanisms are secure.
- Zoom hired external security firms to conduct penetration testing and code reviews after privacy concerns in 2020, which led to the adoption of AES-256 GCM encryption for meetings.
- Facebook employs a "bug bounty" program encouraging external researchers to test their cryptographic systems and report weaknesses.

7. Ongoing Security Monitoring

Continuously monitor cryptographic systems, track certificates, update libraries, and audit cryptographic operations.

- Let's Encrypt automates certificate monitoring and renewal, ensuring websites using their certificates maintain valid TLS encryption.
- AWS CloudTrail monitors key usage in AWS KMS, logging all encryption and decryption operations for auditing.
- Google monitors certificate transparency logs to detect rogue or misissued certificates affecting their domains, enabling fast mitigation.

8. Defense in Depth

Combine cryptography with other security layers such as access control, network segmentation, intrusion detection, and multi-factor authentication.

- Bank of America uses layered security controls: all transactions are encrypted (AES/TLS), accounts require multi-factor authentication, and network segmentation protects internal systems.
- Netflix employs defense-in-depth for streaming content: encrypted video streams, secure API keys, firewall rules, and continuous monitoring prevent data leaks or content piracy.

- Signal combines end-to-end encryption with secure transport protocols, app sandboxing, and device-level protections to ensure messages remain private even if one layer is compromised.

Real World Scenarios

1. WhatsApp Backdoor Controversy (2017)

WhatsApp uses end-to-end encryption but in 2017 a researcher Tobias Boelter (University of California, Berkeley) revealed the cryptographic vulnerability in the end-to-end encryption implementation of the WhatsApp messaging application.

In End-to-end encryption if a user have changed their phone or reinstalls WhatsApp, a new encryption key pair is generated. If there were any undelivered messages, WhatsApp would automatically re-encrypt and resend them using the new key, without immediately alerting the sender.

This behavior explains that WhatsApp servers could force a key change intentionally or compromised and request messages to be re-sent. As the sender wasn't warned immediately, attackers controlling the server could potentially intercept and read those messages.

WhatsApp responded that this behavior was a deliberate design decision, they denied the existence of a backdoor and emphasized that users can enable Security Notifications. Many security experts concluded that this was not a backdoor, but rather a usability trade-off.

Even though WhatsApp did not suffer a direct financial loss from this flaw,they faced a reputational damage as many news outlets initially reported it as a "backdoor," which made users worry about their privacy and many users misunderstood the issue and thought WhatsApp's encryption was "broken," even though it wasn't and switched into different platforms such as Telegram.

2. Heartbleed Vulnerability (2014)

Heartbleed is a vulnerability in OpenSSL that came to light in April 2014, which was discovered by Neel Mehta, an engineer working at Google. It affected thousands of web servers, including major sites like Yahoo. OpenSSL is an open-source software library that implements the Transport Layer Security (TLS) and Secure Sockets Layer (SSL**)** protocols to secure data transmitted over the internet.

The name "Heartbleed" comes from the heartbeat**,** a feature of TLS/SSL used to check that two computers communicating with each other are still connected, even when no data is being actively sent. During a heartbeat, one computer sends a small, encrypted message (heartbeat

request) to the other, which is supposed to send back the exact same data, confirming the connection is alive.

Due to a programming error, OpenSSL did not verify that the length of the message was accurate. This flaw allowed an attacker to trick the server into sending more data than it should, potentially revealing sensitive information from the server's memory, such as passwords, private keys, and personal data.

As a remedy to this issue Heartbleed fix was rolled out in version 1.0.1g of the OpenSSL library, released on April 8, 2014, and was also included in all subsequent versions of the software.Due to Heartbleed vulnerability Millions of websites and services, including Yahoo, GitHub, and banking sites, were affected.

As consequence for the Heartbleed vulnerability it explains the importance of rigorous code review and timely patching in cryptographic implementations.

3. Dual EC DRBG (2007)

Random numbers are critical for cryptography: for encryption keys, random authentication challenges, initialization vectors, key-agreement schemes etc. Break the random-number generator, and most of the time you break the entire security system. Dual Elliptic Curve Deterministic Random Bit Generator (Dual EC DRBG) was a random number generator (RNG) which was standardized by NIST (National Institute of Standards and Technology) in 2007. However at the CRYPTO 2007 conference in August, Dan Shumow and Niels Ferguson showed that the algorithm of Dual EC DRBG contains a weakness that can only be described as a backdoor.

Dual EC DRBG was based on elliptic curves. There are a bunch of constants (fixed numbers) in the standard used to define the algorithm's elliptic curve. These constants are listed in Appendix A of the NIST publication, but nowhere is it explained where they came from. Shumow and Ferguson showed that these fixed numbers have a relationship with a second, secret set of numbers that can act as a kind of skeleton key. If you know the secret numbers, you can predict the output of the random-number generator after collecting just 32 bytes of its output. The researchers stated that they don't know what the secret numbers(Fixed numbers) are. But because of the way the algorithm works, the person who produced the constants might know and he has the mathematical opportunity to produce the constants and the secret numbers in tandem.

This incident was suspected as this was a deliberate backdoor by the NSA (National Security Agency) and it was founded that RSA Security allegedly accepted $10 million from the NSA to make Dual EC DRBG default in its products. This severely damaged RSA's credibility, leading to boycotts and lawsuits.

Due to this incident the companies (including RSA) who used Dual EC DRBG removed from their products and In 2014, NIST formally withdrew Dual EC DRBG from its recommendations after public backlash.

4. Debian OpenSSL RNG Bug (2006–2008)

The Debian OpenSSL RNG Bug occurred between 2006 and 2008 when a developer modified the OpenSSL package for Debian Linux. The change inadvertently removed a line of code that added extra entropy to the random number generator (RNG).This was discovered by Luciano Bello in 2007**.**

Because of this modification, the RNG produced predictable outputs, making it possible for attackers to guess or brute-force cryptographic keys. This affected a wide range of cryptographic operations, including SSL/TLS certificates, SSH keys, VPN keys, and other secure communications on Debian-based systems. Millions of keys generated during this period were vulnerable to compromise.

The weakness was not due to a flaw in the OpenSSL algorithms themselves, but rather a cryptographic implementation failure caused by poor configuration. Once discovered, Debian promptly fixed the RNG, and administrators were required to regenerate all cryptographic keys created during the vulnerable period. The incident highlighted the critical importance of proper RNG implementation and careful code review in cryptographic systems.

Conclusion

Cryptography plays a vital role in safeguarding modern digital systems by ensuring confidentiality, integrity, and authenticity of data. However, cryptographic failures can undermine these protections, often due to weak algorithms, poor key management, flawed implementations, misconfigurations, or human factors such as insider threats. Real-world incidents like Heartbleed, the Debian OpenSSL RNG bug, and the Dual EC DRBG controversy demonstrate that even small oversights can have global security consequences.

Preventing such failures requires not only adopting strong and modern algorithms but also implementing secure key management, robust coding practices, continuous testing, and proactive monitoring. Equally important is the recognition that cryptography alone is not sufficient organizations must apply defense-in-depth strategies, combining encryption with layered security controls and staff awareness.

Ultimately, cryptographic security is an ongoing process rather than a one-time implementation. By continuously updating practices in line with evolving threats and standards, organizations can reduce the risk of cryptographic failures and maintain the trust and safety of their users.

References

The Hacker News. (2017, January 14). Explained — What's up with the WhatsApp "backdoor" story? *The Hacker News*. https://thehackernews.com/2017/01/whatsapp-encryption-backdoor.html

Lomas, N. (2017, January 13). Encrypted messaging platform WhatsApp denies "backdoor" claim. *TechCrunch*. https://techcrunch.com/2017/01/13/encrypted-messaging-platform-whatsapp-denies-backdoor-claim/

Finnie, R. (2024, May 13). I discovered the Debian OpenSSL bug. https://www.finnie.org/2024/05/13/i-discovered-the-debian-openssl-bug/

Schneier, B. (2007, November 15). Did NSA put a secret backdoor in new encryption standard? *Schneier on Security.* https://www.schneier.com/essays/archives/2007/11/did_nsa_put_a_secret.html

Cybersecurity News. (2024, December 11). What is cryptographic failures? *Cybersecurity News.* https://cybersecuritynews.com/cryptographic-failures/

Fruhlinger, J. (2022, September 6). The Heartbleed bug: How a flaw in OpenSSL caused a security crisis. *CSO Online*. https://www.csoonline.com/article/562859/the-heartbleed-bug-how-a-flaw-in-openssl-caused-a-security-crisis.html

OWASP. (2021). A02: Cryptographic failures – OWASP Top 10. *OWASP.* https://owasp.org/Top10/A02_2021-Cryptographic_Failures/

Security Journey. (n.d.). OWASP Top 10 cryptographic failures explained. *Security Journey.* https://www.securityjourney.com/post/owasp-top-10-cryptographic-failures-explained

Invicti. (n.d.). Cryptographic failures. *Invicti.* https://www.invicti.com/blog/web-security/cryptographic-failures/

Monga, A. (n.d.). Cryptographic failures: Understanding and preventing vulnerabilities. *Medium.* https://medium.com/@ajay.monga73/cryptographic-failures-understanding-and-preventing-vulnerabilities-91c8b2c56854

Certera. (n.d.). What is cryptographic failure? Real-life examples, prevention & mitigation. *Certera.* https://certera.com/blog/what-is-cryptographic-failure-real-life-examples-prevention-mitigation/