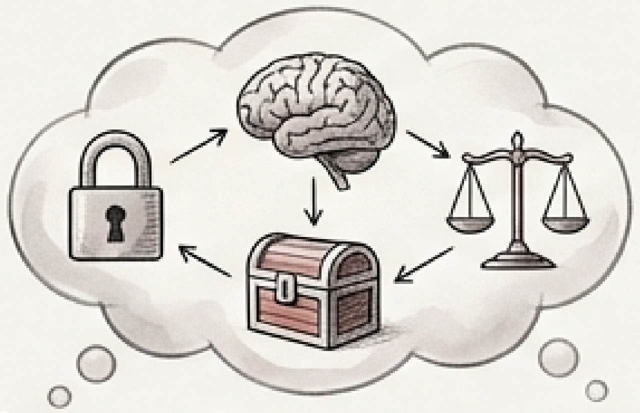
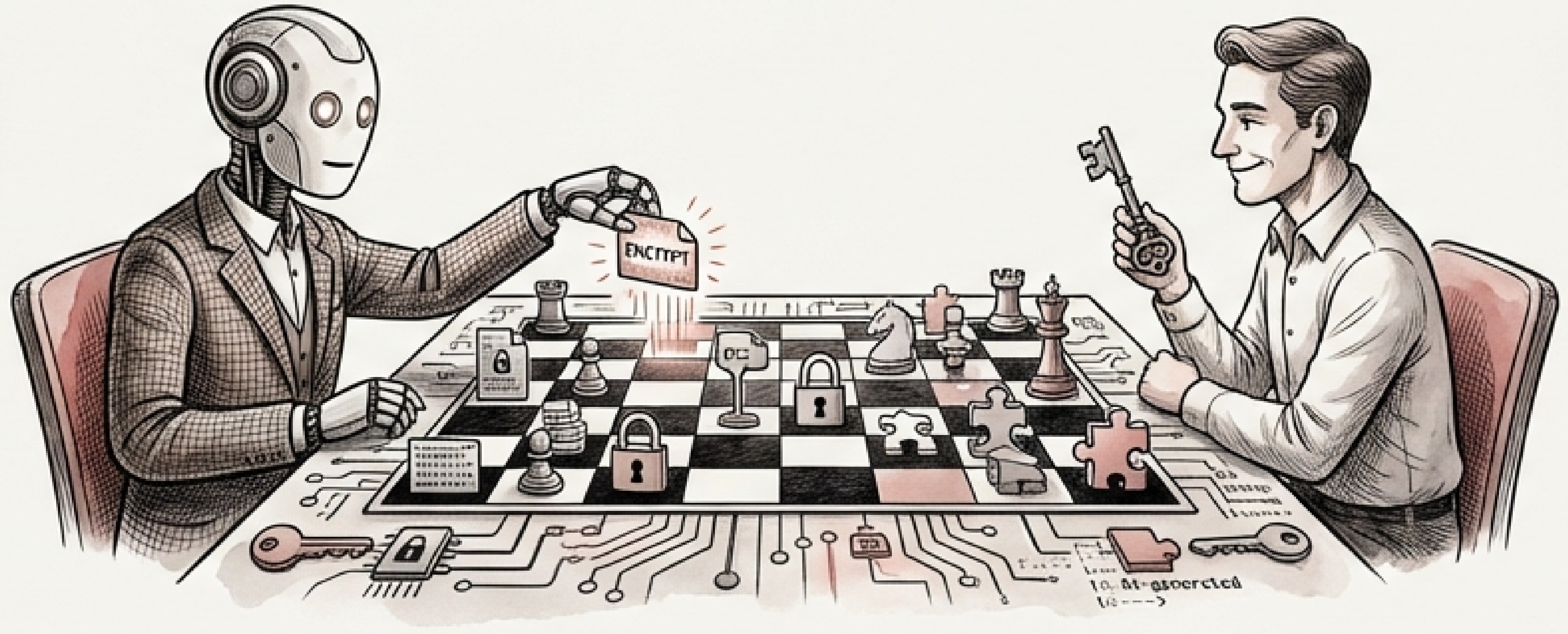


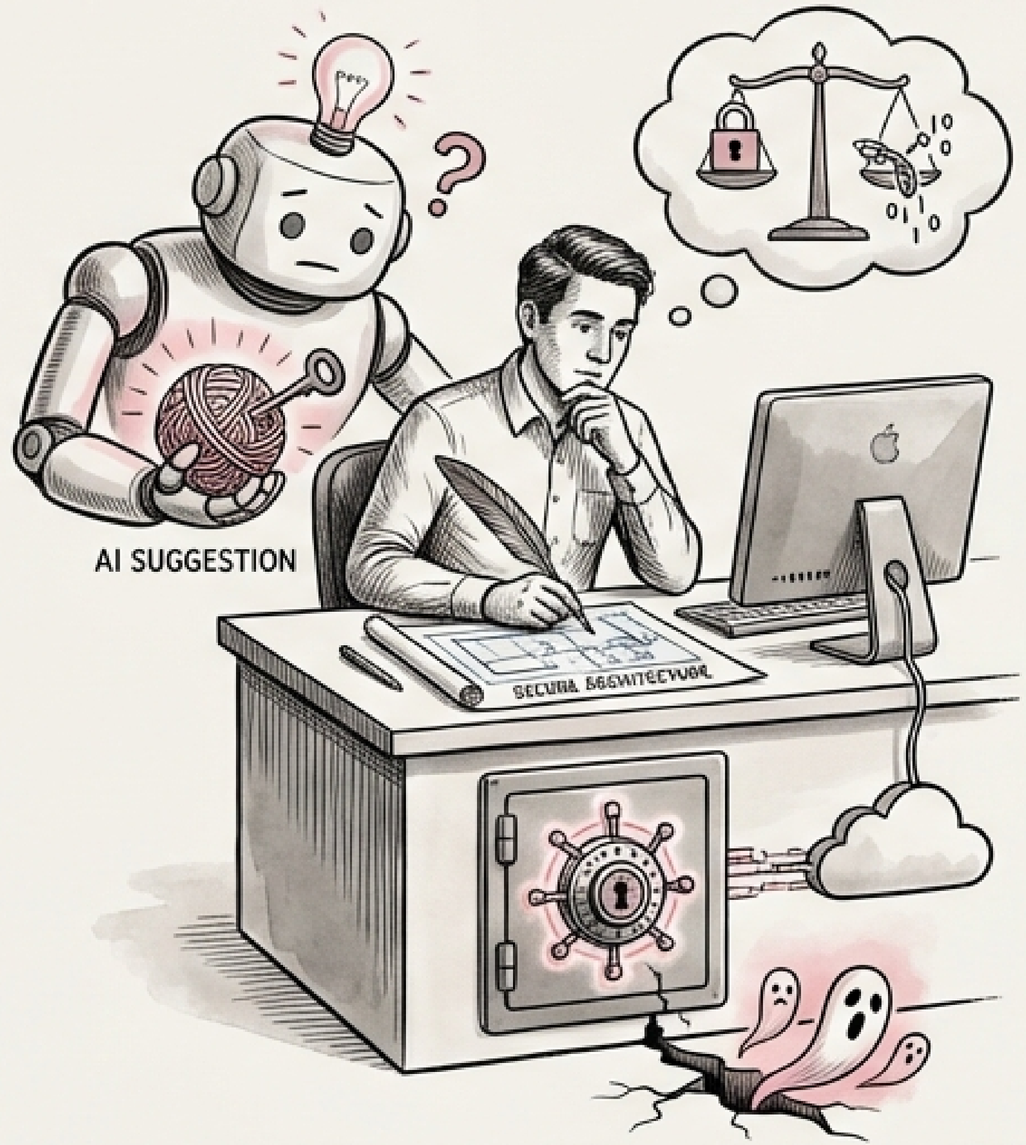
Cryptography in AI-Augmented Development: A High-Stakes Game



A HIGH-STAKES GAME

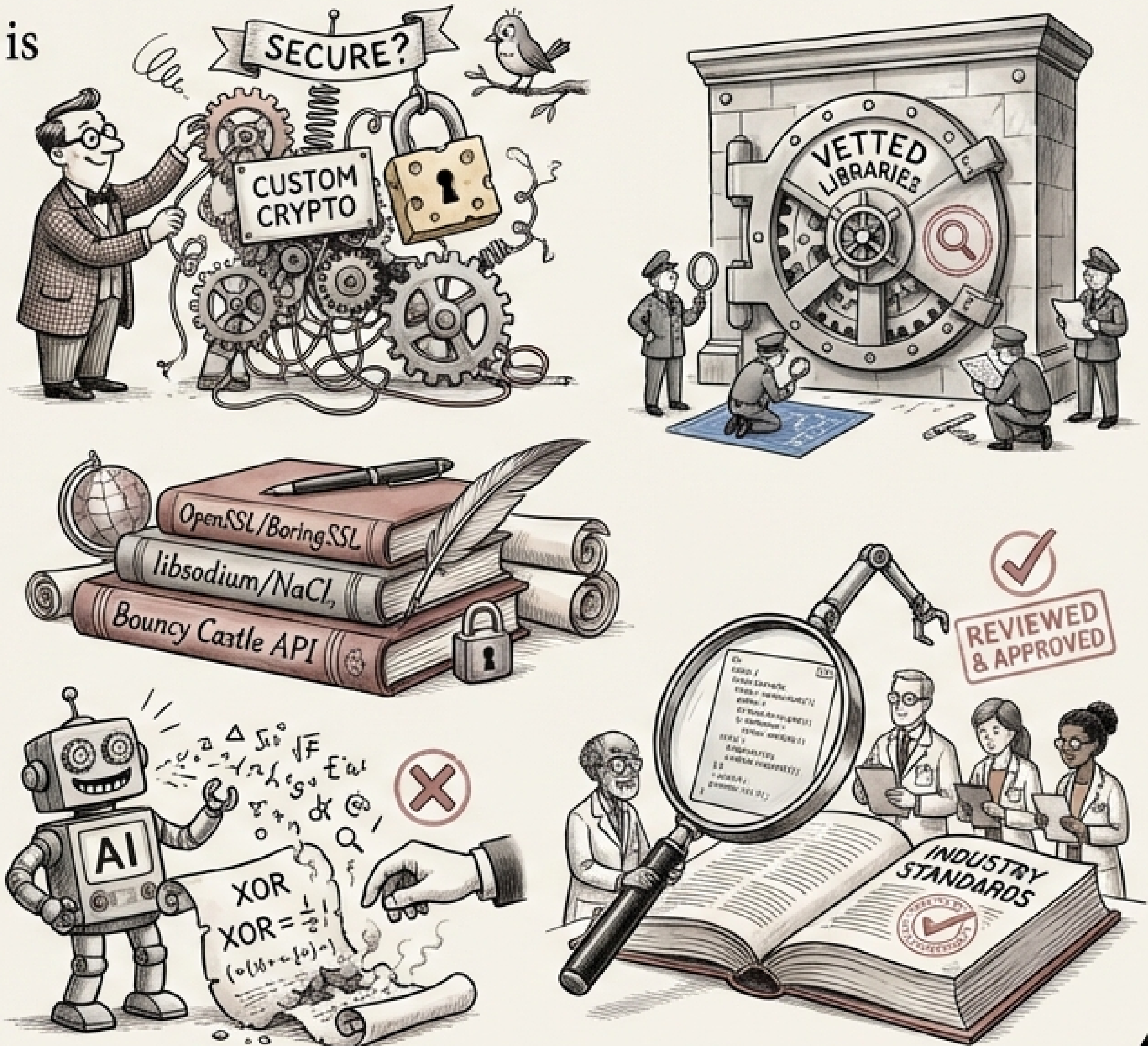
CRYPTOGRAPHY IN AI-AUGMENTED DEVELOPMENT: A HIGH-STAKES GAME

- Cryptography is essential for data protection but dangerous if implemented incorrectly, leading to false security assumptions.
- AI coding tools, while helpful, frequently suggest deprecated or insecure cryptographic patterns.
- Developers must critically review AI-generated code to avoid introducing cryptographic vulnerabilities into applications.
- Ignoring cryptographic best practices can result in severe data breaches and compromise sensitive information.
- This module will equip developers with the knowledge to identify and avoid common AI-induced cryptographic errors.



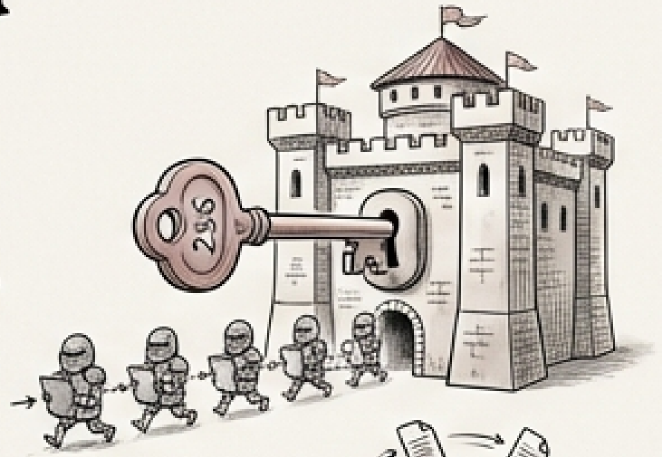
The Cardinal Rule: Never “Roll Your Own” Crypto

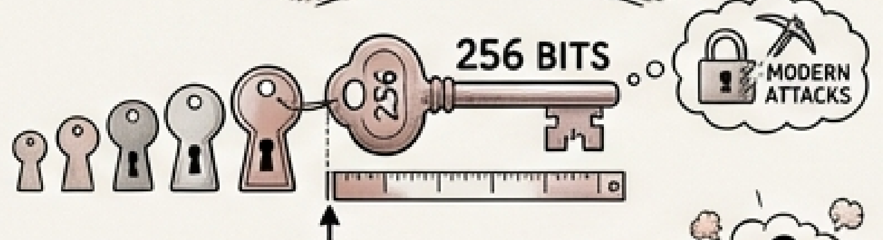
- Implementing custom cryptographic algorithms is strongly discouraged due to the complexity and likelihood of introducing vulnerabilities.
- Reliance on vetted, peer-reviewed libraries ensures that algorithms and implementations have undergone rigorous security analysis.
- Recommended libraries include OpenSSL/BoringSSL, libsodium/NaCl, Bouncy Castle, and the Web Crypto API.
- AI tools often violate this rule by generating custom encryption functions, such as XOR-based schemes.
- All AI-generated cryptographic code MUST be meticulously reviewed against industry-approved algorithms and best practices.

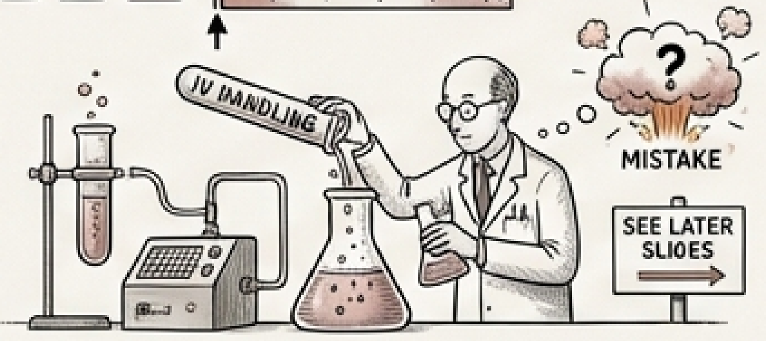


Symmetric Encryption: Approved Algorithms and Use Cases

- AES-256-GCM is the preferred symmetric encryption algorithm for general-purpose applications due to its robust security.
- ChaCha20-Poly1305 is recommended for mobile and resource-constrained environments because of its efficient performance.
- Symmetric algorithms encrypt and decrypt data using the same key, making them suitable for bulk data encryption.
- Key sizes of at least 256 bits are necessary for AES to provide sufficient security against modern attacks.
- Ensure proper initialization vector (IV) handling when using these algorithms *(see later slides on common mistakes)*.

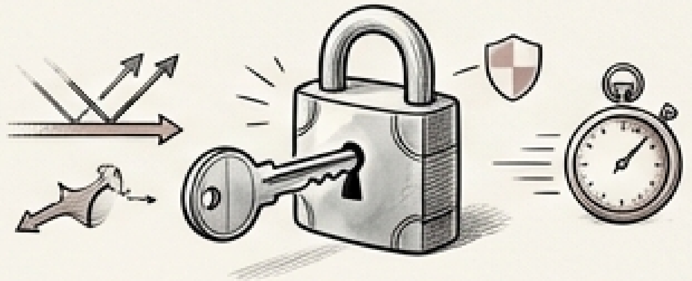


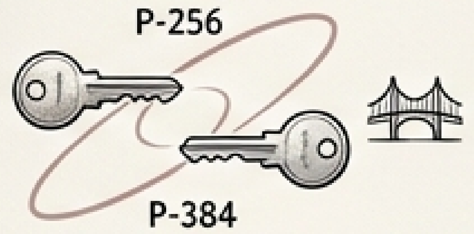




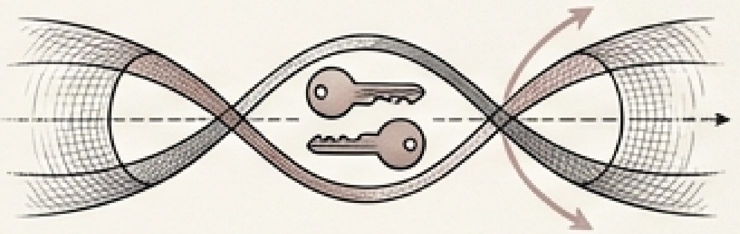
Asymmetric Encryption: Algorithm Selection and Legacy Considerations

- Ed25519 is the preferred asymmetric algorithm for digital signatures due to its security, performance, and resistance to side-channel attacks.

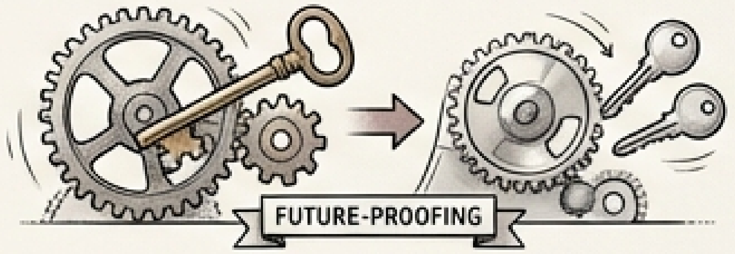




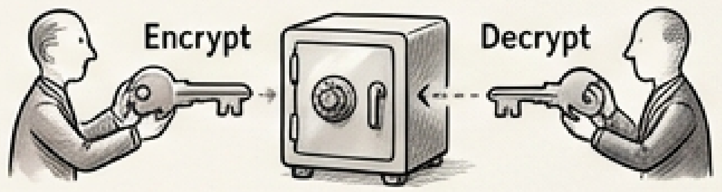
- ECC (Elliptic Curve Cryptography) using P-256/P-384 curves is also approved for key exchange and digital signatures.



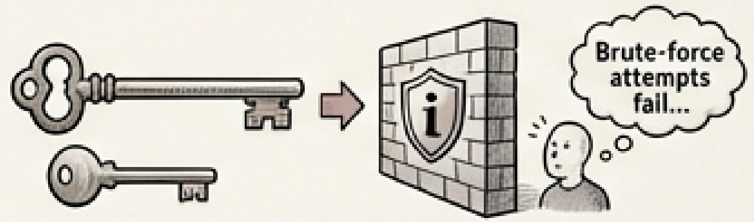
- RSA-2048+ is acceptable for legacy systems but should be phased out in favor of ECC or Ed25519 where possible.



- Asymmetric algorithms use separate keys for encryption and decryption, enabling secure key exchange and digital signatures.

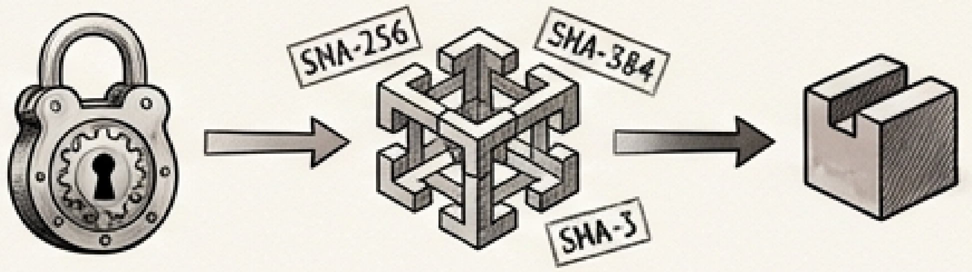


- RSA keys should be at least 2048 bits in length to provide adequate security.



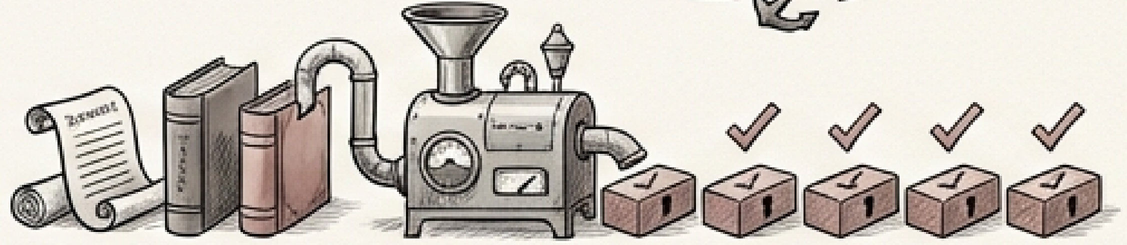
Hashing: Securely Transforming Data

- SHA-256, SHA-384, and SHA-3 are approved hashing algorithms for generating one-way cryptographic hashes.



- BLAKE2 and BLAKE3 offer performance advantages over SHA-2 while maintaining strong security properties.

- Hashing algorithms produce fixed-size outputs from arbitrary-length inputs, making them useful for data integrity checks and password storage.

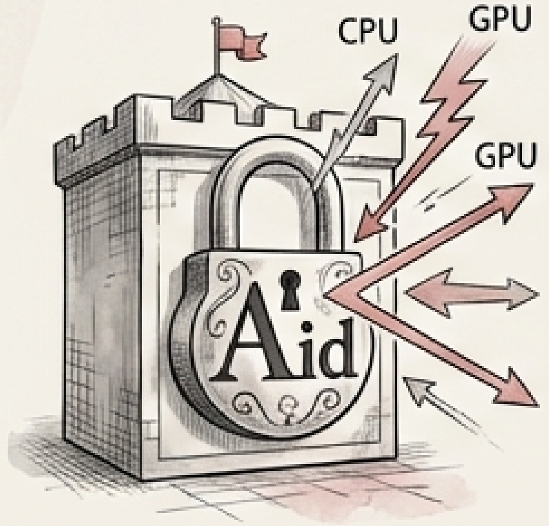


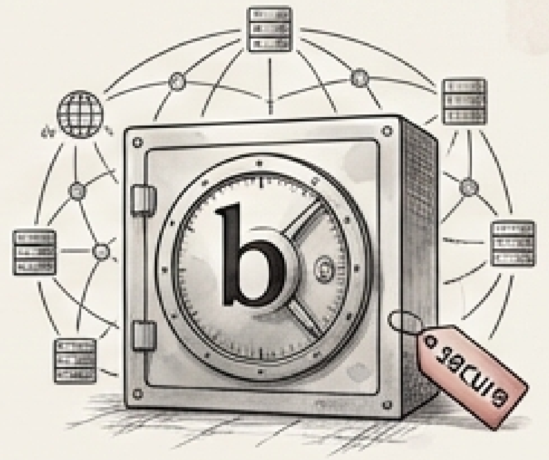
- Hashing is NOT encryption; it cannot be reversed to recover the original data.

ENCRYPTION
HASHING
NOT REVERSIBLE

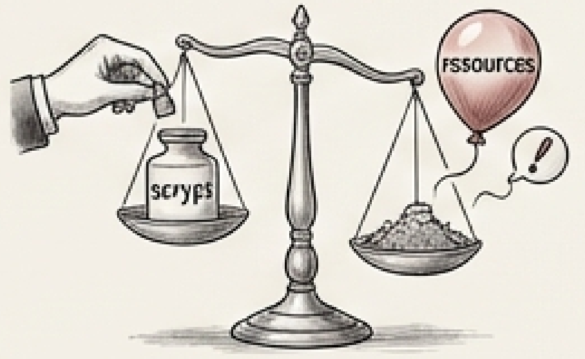
- Never use MD5 or SHA-1 for security purposes due to their known vulnerabilities (collision attacks).

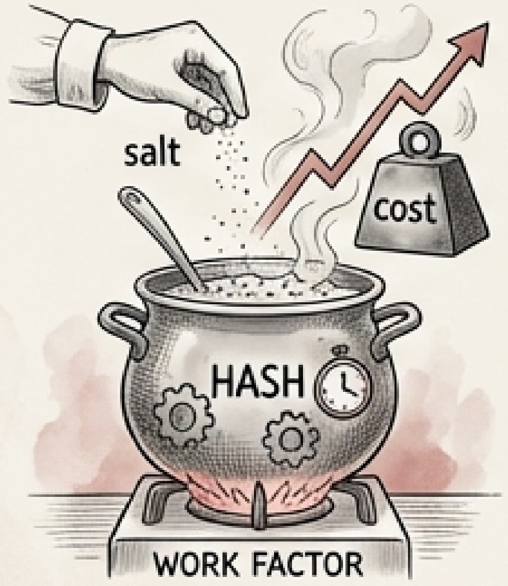
Password Hashing: Protecting User Credentials



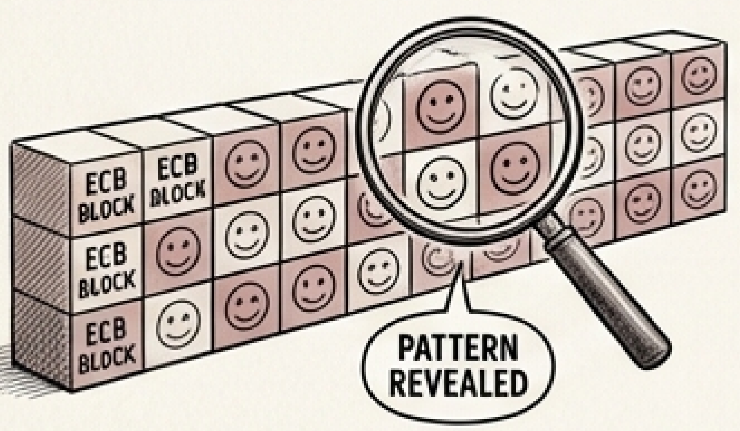


- **Argon2id** is the preferred password hashing algorithm due to its resistance to both CPU and GPU-based attacks.
- **bcrypt** is an acceptable alternative to Argon2id, offering strong security and widespread availability.
- **scrypt** is also acceptable, but requires careful parameter tuning to avoid resource exhaustion attacks.
- Password hashing algorithms should be configured with appropriate salt and work factor parameters to increase computational cost.
- Never store passwords in plaintext or use simple hashing algorithms like MD5 or SHA-1.

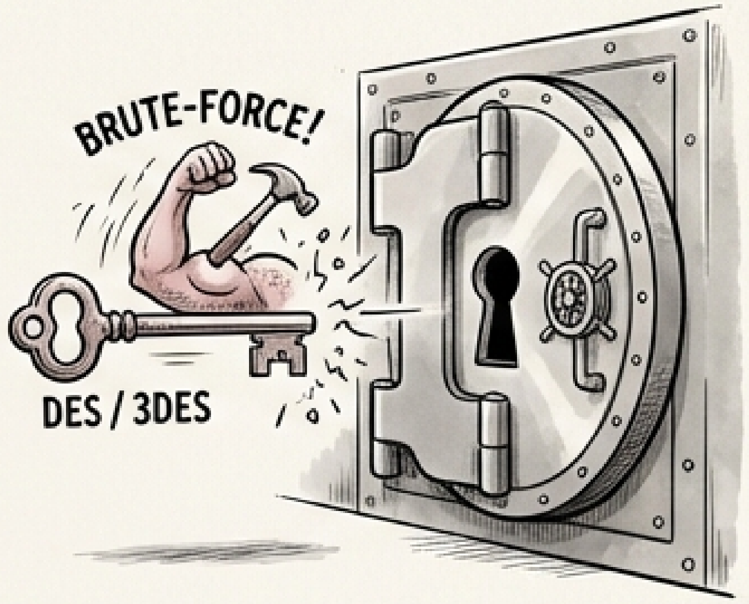




Banned Cryptographic Algorithms: AVOID THESE AT ALL COSTS



- MD5 and SHA-1 are vulnerable to collision attacks and should not be used for security purposes.
- DES and 3DES are outdated symmetric encryption algorithms with small key sizes that are susceptible to brute-force attacks.
- RC4 is a stream cipher with known vulnerabilities and should not be used.
- ECB (Electronic Codebook) mode should never be used with block ciphers because it reveals patterns in the ciphertext.
- Using banned algorithms introduces significant security risks and potential compliance violations.

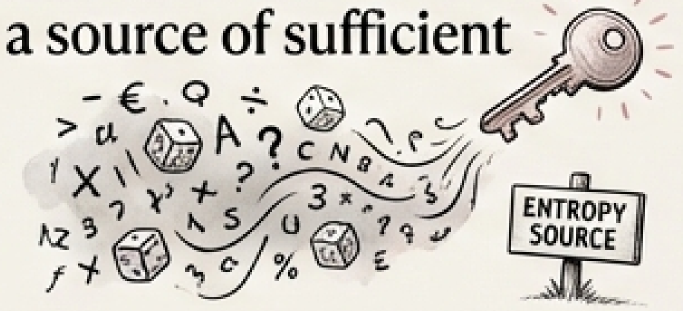


TLS Configuration: Securing Network Communications

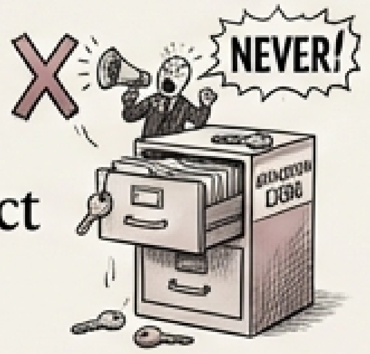
- TLS 1.3 is the preferred TLS version due to its improved security and performance.
- TLS 1.2 is the minimum acceptable version, but migration to TLS 1.3 is strongly recommended.
- Only approved cipher suites should be enabled to prevent the use of weak or vulnerable algorithms.
- HSTS (HTTP Strict Transport Security) should be enabled with a long max-age to enforce HTTPS connections.
- Certificate pinning can be used in mobile apps to prevent man-in-the-middle attacks, but requires careful management.

Key Management Lifecycle: A Cradle-to-Grave Approach

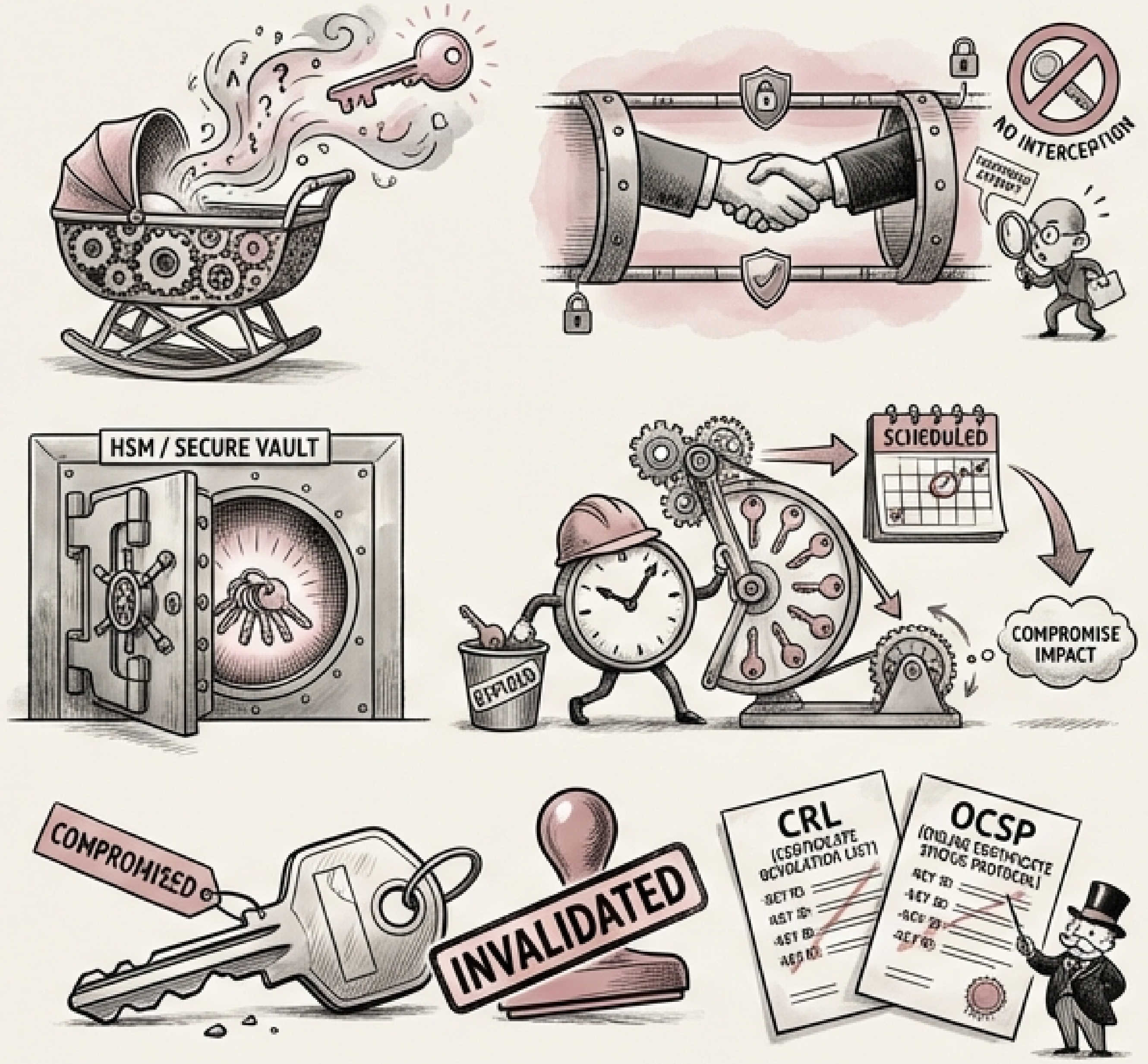
- Key generation requires a source of sufficient entropy to ensure the unpredictability of the generated keys.



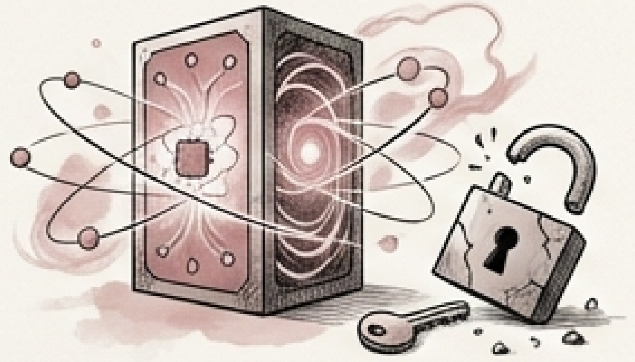
- Key distribution must be performed over secure channels to prevent interception or tampering.
- Key storage should be in Hardware Security Modules (HSMs) or secure vaults, never hardcoded in application code.



- Key rotation should be automated and scheduled to minimize the impact of key compromise.
- Key revocation mechanisms (CRL/OCSP) should be in place to invalidate compromised keys.

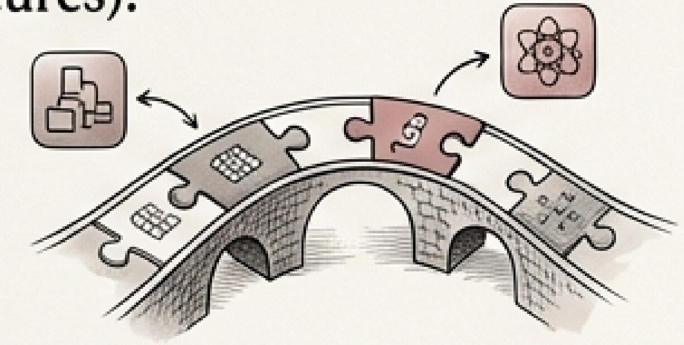


PREPARING FOR THE FUTURE: POST-QUANTUM CRYPTOGRAPHY

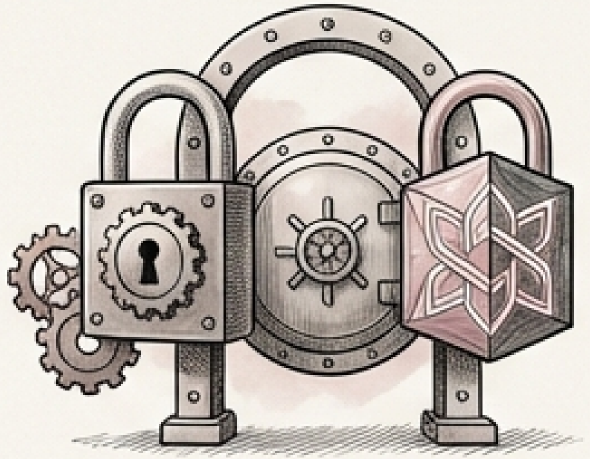


- Quantum computers pose a threat to current cryptographic algorithms, necessitating the development of post-quantum cryptography (PQC).

- NIST (National Institute of Standards and Technology) is developing PQC standards, including ML-KEM (key encapsulation) and ML-DSA (digital signatures).



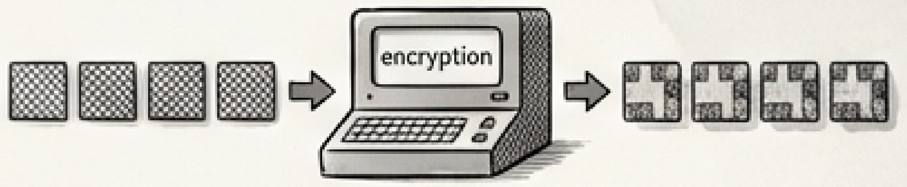
- Crypto agility is essential: design systems to easily swap cryptographic algorithms without requiring major architectural changes.

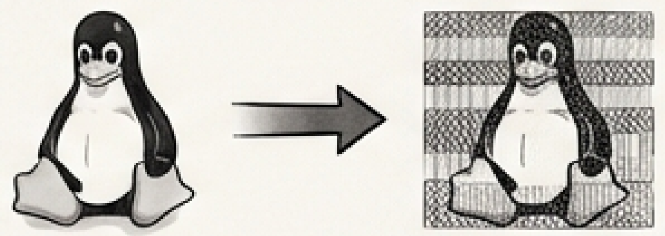


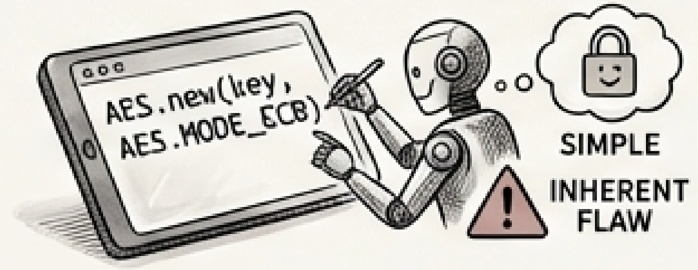
- Start planning for hybrid approaches now, combining classical and post-quantum algorithms for increased security.
- Aim for full migration to PQC by 2030 to mitigate the risk of quantum attacks.

AI Crypto Mistake #1: The Dangers of ECB Mode

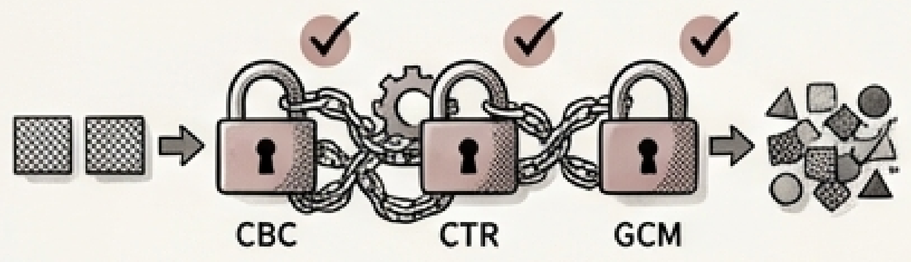
- ECB (Electronic Codebook) mode encrypts each block of data independently, leading to visible patterns in the ciphertext.
- This pattern visibility can reveal sensitive information about the encrypted data, compromising confidentiality.
- ECB mode is particularly vulnerable when encrypting images or other structured data with repeating patterns.
- AI-generated code frequently uses ECB mode due to its simplicity, despite its inherent security flaws.
- Always use a secure mode of operation, such as CBC, CTR, or GCM, instead of ECB.





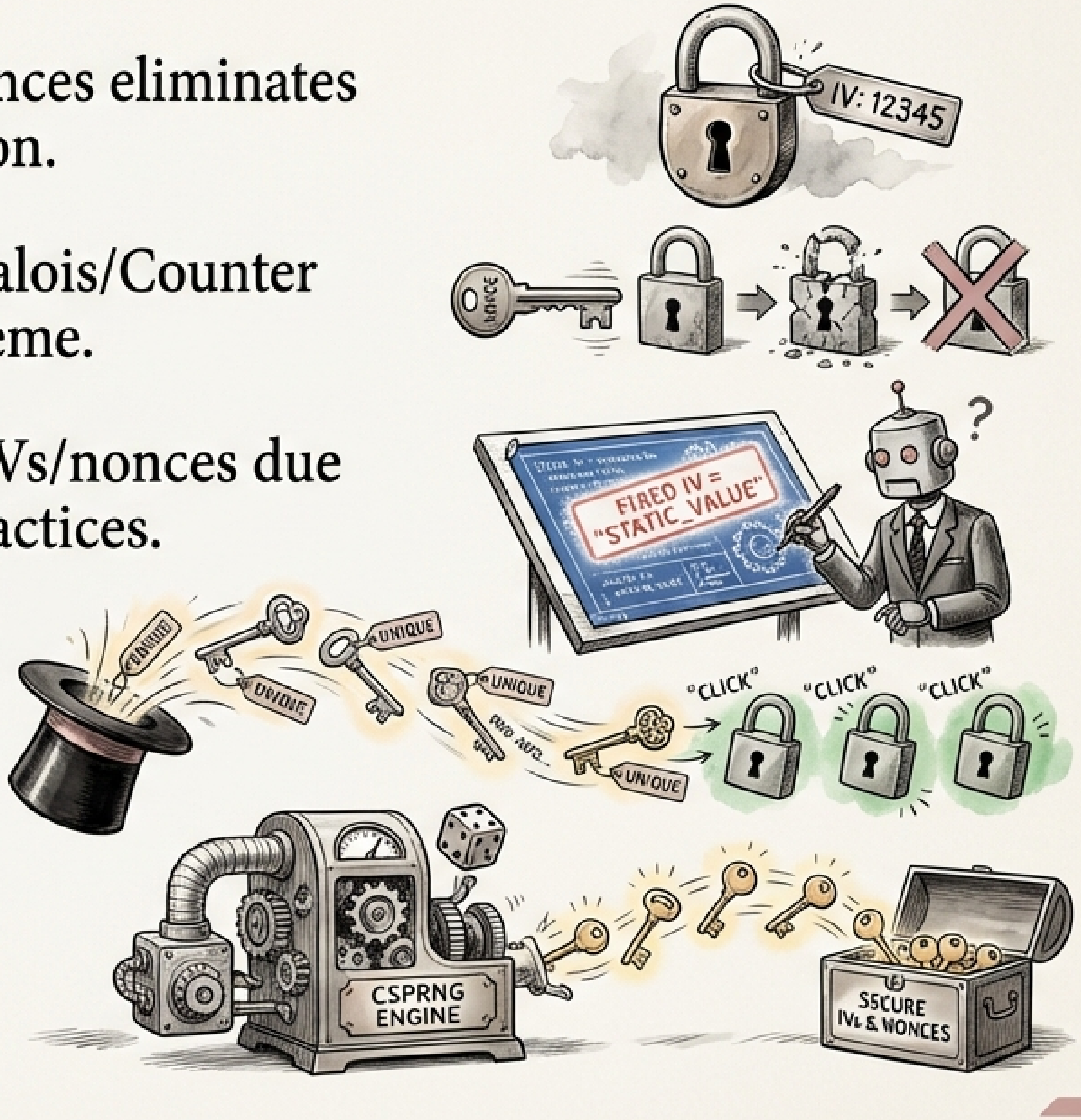


```
AES.new(key,
AES.MODE_ECB)
```

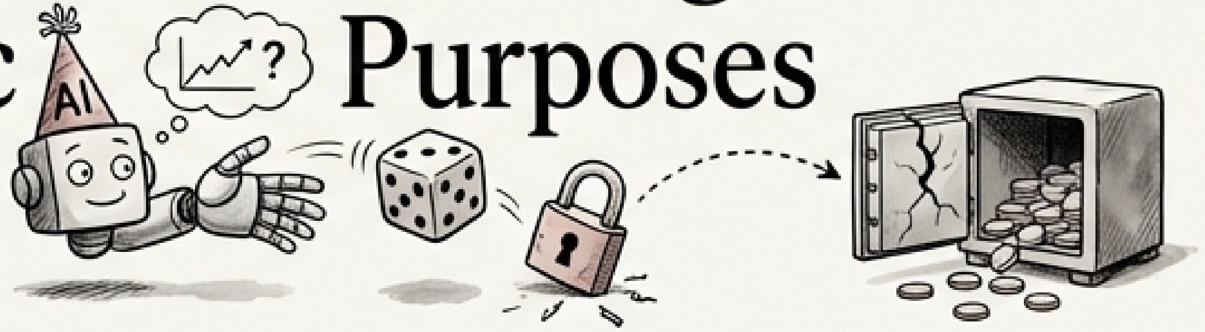


AI CRYPTO MISTAKE #2: HARDCODED IVs AND (vCCL) AND NONCES REUSE

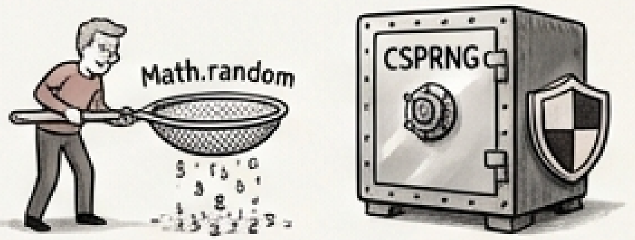
- Hardcoding Initialization Vectors (IVs) or nonces eliminates the randomness required for secure encryption.
- Reusing a nonce with a given key in GCM (Galois/Counter Mode) completely breaks the encryption scheme.
- AI tools may generate code with hardcoded IVs/nonces due to lack of awareness of cryptographic best practices.
- Always generate IVs/nonces randomly and uniquely for each encryption operation.
- Use a Cryptographically Secure Pseudo-Random Number Generator (CSPRNG) for IV/nonce generation.

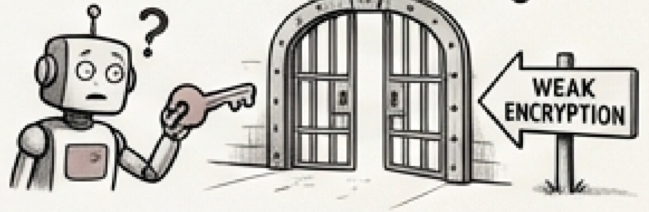


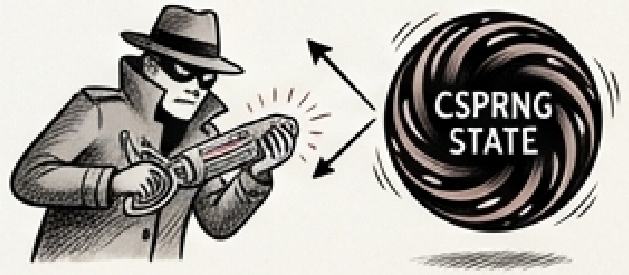
AI Crypto Mistake #3: Using `Math.random()` for Cryptographic Purposes



- `Math.random()` and similar functions are **not** Cryptographically Secure Pseudo-Random Number Generators (CSPRNGs).
- These functions lack the statistical properties required for cryptographic applications and are predictable.
- AI tools may naively use `Math.random()` for key generation or IV generation, leading to weak cryptography.
- Always use a CSPRNG specifically designed for cryptographic purposes, such as those provided by the operating system or cryptographic libraries.
- CSPRNGs are designed to be unpredictable even to attackers with knowledge of the algorithm's internal state.

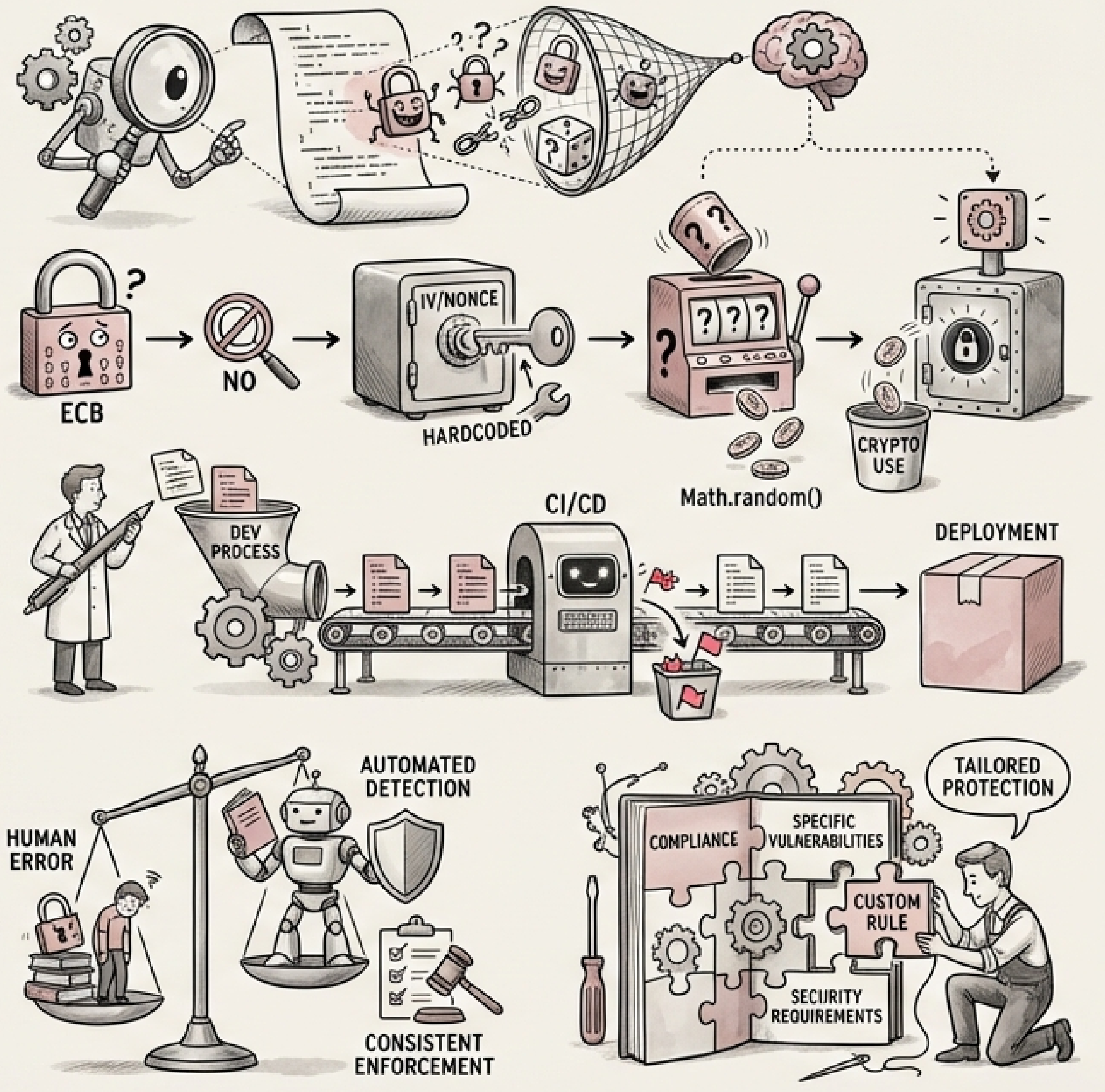




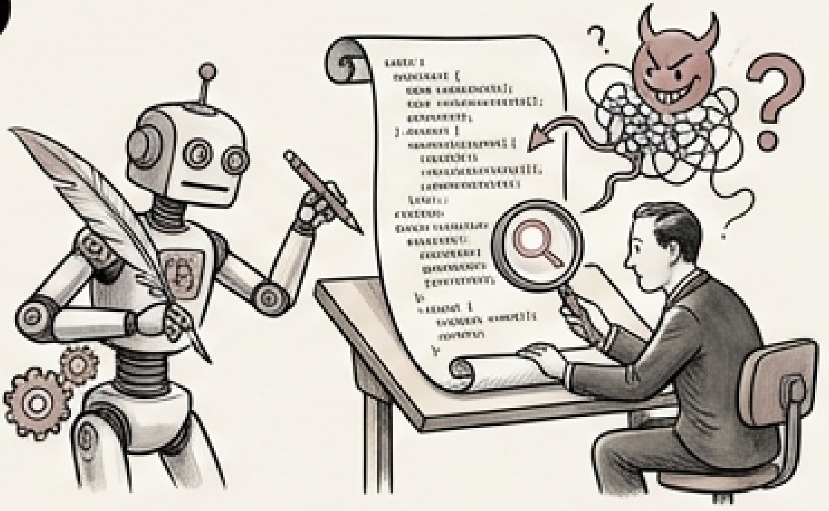


Automated Detection: Semgrep Rules for Crypto Mistakes

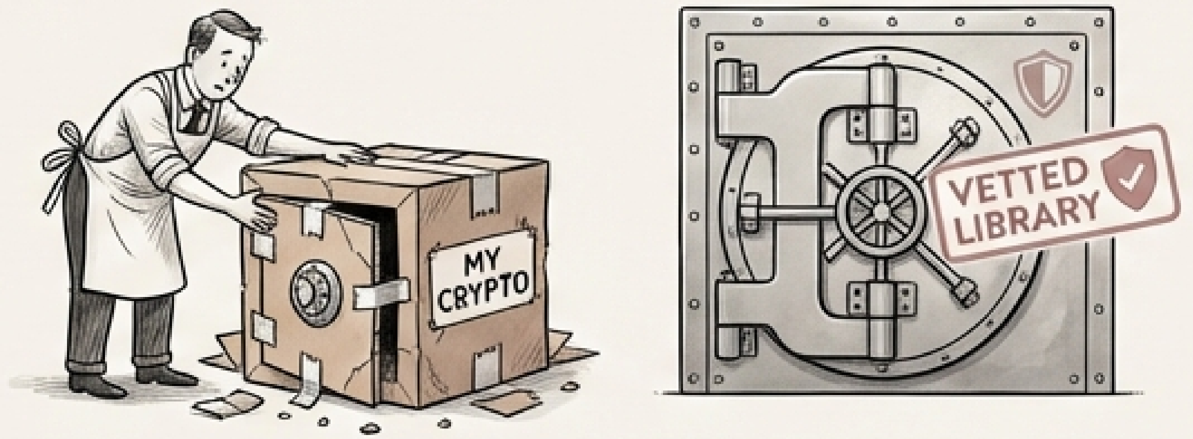
- Semgrep is a static analysis tool that can be used to detect common cryptographic mistakes in code.
- Create Semgrep rules to automatically identify instances of ECB mode usage, hardcoded IVs/nonces, and use of Math.random() for crypto.
- Integrate Semgrep into your CI/CD pipeline to catch cryptographic errors early in the development process.
- Automated detection reduces the risk of human error and ensures consistent enforcement of cryptographic best practices.
- Semgrep rules can be customized to address specific cryptographic requirements and vulnerabilities.



SECURE CRYPTO IN AI-AUGMENTED DEVELOPMENT: VIGILANCE IS KEY

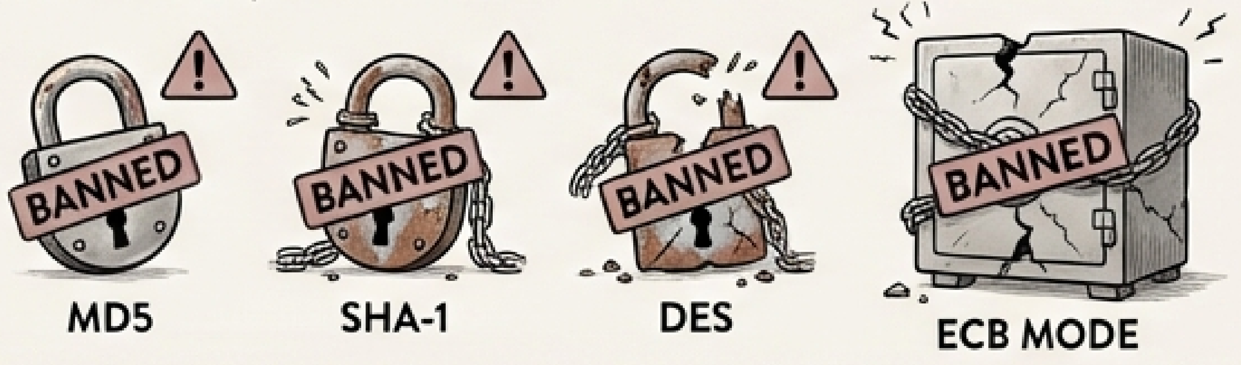


- **AI coding tools** can assist with development, but require careful review to avoid introducing cryptographic vulnerabilities.
- **Adhere to the cardinal rule:** never "roll your own" crypto; use vetted cryptographic libraries.

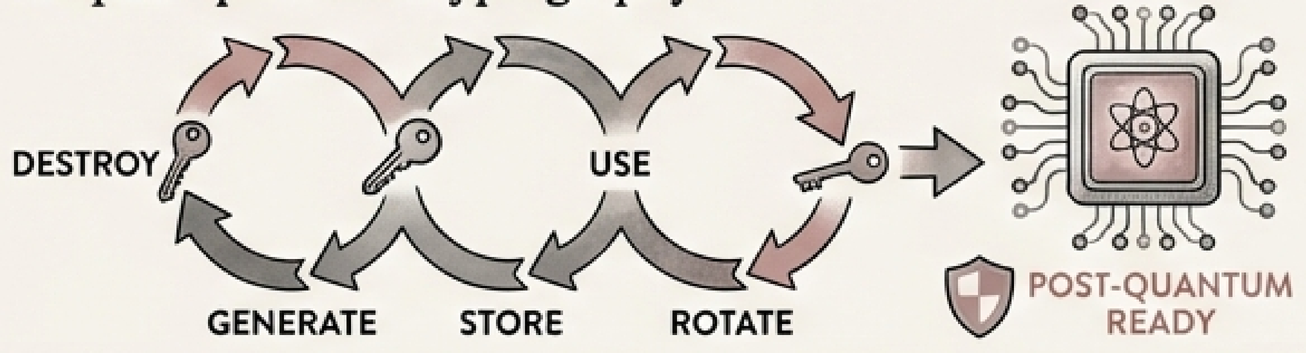


- Select approved algorithms based on security requirements and performance constraints (e.g., AES-256-GCM, Ed25519, Argon2id).

- **Avoid banned algorithms** (e.g., MD5, SHA-1, DES, ECB mode) at all costs.



- Implement a robust key management lifecycle and prepare for post-quantum cryptography.



Thank You

- Questions?