### **Scenario 1: The Mysterious Death of Gerald Cotten and the Frozen Wallets**

**Objective:** To prevent loss of access to cryptocurrency funds in the event of the death or disappearance of a key custodian by enabling threshold-based, distributed key management and recovery mechanisms.

**Cryptocurrency Custody Resilience Protocol**

**Entities Involved:**

* CEO/Wallet Manager
* Cold Storage Backup Administrator
* Multi-Signature Wallet System
* Legal Authority or Trustee

**Steps:**

1. **Multi-Signature Setup**: Deploy a wallet that requires at least 3 out of 5 signatures (3-of-5 threshold scheme). This is implemented using m-of-n cryptographic multi-sig schemes, typically supported in wallet software like Bitcoin’s P2SH or Ethereum’s smart contracts.
2. **Key Sharing with Shamir's Secret Sharing**: Use Shamir’s Secret Sharing Scheme (SSSS), where the master private key K is divided into 5 parts using modular polynomial arithmetic:  
   * f(x)=a0+a1x+a2x2+...+at−1xt−1mod  pf(x) = a\_0 + a\_1x + a\_2x^2 + ... + a\_{t-1}x^{t-1} \mod p
   * Each key share is a point (x\_i, f(x\_i)) on the polynomial. Any 3 shares can reconstruct the master key using Lagrange interpolation.
3. **Role-Based Distribution**: Assign the shares to the CEO, a legal trustee, an independent auditor, and two senior technical officers.
4. **Hardware Security Module (HSM) Storage**: Encrypt each share using AES-256 and store them in HSMs that require biometric authentication and PIN verification.
5. **Rotation and Auditing**: Shares are rotated every quarter using new random polynomials. All access attempts and key usage are logged and hashed using SHA-256 into a Merkle tree for tamper-proof audit trails.
6. **Recovery Protocol**: In case of unavailability of any custodian, the legal trustee verifies death using a government-issued certificate and triggers the key reconstruction mechanism.
7. **Transaction Authorization**: A transaction is considered valid only when it contains at least 3 out of 5 digital signatures. Verification is done using standard ECDSA signature aggregation.

**Security Features:**

* Threshold access with Shamir’s Secret Sharing
* Hardware-enforced key protection
* Legal and cryptographic recovery assurance
* Tamper-proof audit trail

### **Scenario 2: Uber vs. Waymo Trade-Secret Theft**

**Objective:** To secure sensitive intellectual property and prevent unauthorized access, duplication, or exfiltration by insiders, particularly in transitions between competing firms.

**Insider Threat Prevention and IP Protection Protocol**

**Entities Involved:**

* Employee Workstation
* Secure File Server
* DLP System
* HR and Legal Oversight

**Steps:**

1. **Access Control via RBAC**: Implement role-based access control using an access matrix. For example, matrix entries define permissions: (UID123, Read, ProjectA\_DesignFiles).
2. **Document Fingerprinting and DRM**: Every confidential document is embedded with a unique digital watermark using spread-spectrum techniques. Accessed copies have session-specific IDs.
3. **Data Loss Prevention (DLP)**: Deploy endpoint agents that monitor file transfers. Use hash-based detection to block known fingerprinted documents from leaving the system.
4. **Transmission Encryption**: Files are transmitted over TLS 1.3. Additionally, large files are encrypted using hybrid encryption: AES-256 for content and RSA-2048 for key exchange.
5. **Audit Logging and Alerting**: Each file access triggers logging with HMAC-SHA256 and timestamp. Logs are stored in a tamper-evident log chain:  
   * H\_1 = SHA256(event\_1)
   * H\_n = SHA256(H\_{n-1} + event\_n)
6. **Credential Revocation on Exit**: Employee offboarding initiates revocation of system credentials, rotation of shared secrets, and audit of recent activities.
7. **Periodic Security Audits**: Conduct periodic audits and enforce data access policies through automated rule engines.

**Security Features:**

* Document traceability via watermarking
* Modular access control and session-bound file tagging
* Log integrity via hash chaining

### **Scenario 3: Edward Snowden NSA Leak**

**Objective:** To restrict and monitor access to sensitive national security data, thereby mitigating risks posed by insiders with elevated privileges.

**Privileged Access Monitoring Protocol**

**Entities Involved:**

* Contractor/Analyst
* Intelligence Data Server
* Privileged Access Gateway
* Forensic Audit System

**Steps:**

1. **Zero Trust Authentication**: Enforce MFA using biometric + smart card. Require session context binding (e.g., device ID, geo-location).
2. **Just-In-Time Access**: Access to confidential files is granted temporarily using expirable tokens:  
   * ( token = HMAC\_{K\_s}(userID | timestamp | resourceID)
3. **Session Monitoring**: Every session is screen-recorded and input-logged. Logs are encrypted using session keys and stored in immutable WORM (Write Once Read Many) storage.
4. **Behavioral Baselines**: Use unsupervised ML models (e.g., Isolation Forest) to detect deviations such as sudden large exports.
5. **File Encryption with Key Wrapping**: Files are AES-encrypted using a key K\_f which is itself encrypted using a user-specific RSA public key:  
   * ( C = AES\_{K\_f}(file), E = RSA\_{Pub}(K\_f)
6. **Log Chain for Integrity**: Maintain a Merkle Tree of access events for efficient and verifiable audit.
7. **Incident Response Trigger**: On anomaly, trigger auto-disablement of session, send alert to SOC, and initiate forensic capture.

**Security Features:**

* Zero-trust model and session binding
* Forensic traceability via session capture
* Key-wrapped encryption with public-key cryptography

### **Scenario 4: Netflix Ransom Leak (TDO Attack)**

**Objective:** To enforce strict content security and traceability during third-party post-production, ensuring media cannot be leaked without detection.

**Secure Media Distribution and Monitoring Protocol**

**Entities Involved:**

* Netflix Content Server
* Post-production Vendor Workstation
* Secure Access Gateway

**Steps:**

1. **Encrypted Transfer**: All media files are encrypted using AES-GCM and transferred over a TLS 1.3 VPN.
2. **Session-Specific Forensic Watermarking**: Each media frame is embedded with a watermark encoding the session ID using a Discrete Cosine Transform (DCT) based technique.
3. **Virtual Desktop Infrastructure (VDI)**: Vendors access files in a sandboxed VDI environment with no local storage or clipboard access.
4. **Biometric + Token MFA**: Access requires biometric login and a time-bound token issued via TOTP (Time-based OTP).
5. **Access Monitoring**: View durations, file requests, and playback speed are logged. Any offline access attempt is blocked.
6. **Real-time Watermark Verification**: Screenshots are continuously sampled and compared with original watermarks to detect leaks.
7. **Response Mechanism**: On detection of breach (via watermark fingerprint), the content is disabled, keys are revoked, and law enforcement is notified.

**Security Features:**

* Media traceability via robust watermarking
* Secure playback environments
* Continuous monitoring and automated lockouts

### **Scenario 5: University AES Dictionary Encryption Scheme**

**Objective:** To maintain the confidentiality of research reports containing dictionary-based content while securing novel terms against pattern analysis and replay.

**Dynamic Dictionary AES Encryption Protocol**

**Entities Involved:**

* Research Author
* Encryption Engine
* Server-Side Decryption Processor

**Steps:**

1. **Known Word Encryption**: Each known word in the dictionary is mapped to a unique index. The index is encrypted using AES-CBC:  
   * C=AESKd(Index∥IV)C = AES\_{K\_d}(Index \| IV)
2. **Random IVs**: Each word is encrypted with a unique 128-bit IV to prevent identical ciphertexts from revealing word frequency.
3. **Unknown Word Handling**: If a word is not in the dictionary, it is encrypted using RSA with OAEP padding:  
   * C=RSAPub(word)C = RSA\_{Pub}(word)
4. **Message Padding**: Each message is padded to a uniform length using a padding scheme such as PKCS#7.
5. **MAC Calculation**: The full report is hashed using SHA-256 and HMAC’d with a session key:  
   * MAC=HMACKs(SHA256(report))MAC = HMAC\_{K\_s}(SHA256(report))
6. **Replay Prevention**: A session nonce and timestamp are appended to prevent identical reports being reused maliciously.
7. **Server Decryption**: Server verifies the MAC, checks timestamp freshness (±30 seconds), and decrypts using corresponding keys.

**Security Features:**

* Hybrid encryption for dynamic content
* Frequency obfuscation via IV variation
* Integrity and replay protection

### **Scenario 6: RFID Badge Replay Attack**

**Objective:** To protect physical access systems from replay attacks by ensuring challenge-response authentication for each scan attempt using nonces and HMAC.

**Challenge-Response RFID Authentication Protocol**

**Entities Involved:**

* RFID Badge
* Door Access Reader
* Central Authentication Server

**Steps:**

1. **Nonce Generation**: Each time a badge is scanned, the door reader generates a 64-bit cryptographic nonce using a secure random number generator.
2. **Challenge Transmission**: The nonce and Reader ID are sent to the RFID badge.
3. **HMAC Response**: The badge contains a shared secret key . It computes an HMAC over the nonce using SHA-256:
4. **Response Submission**: The badge sends back its ID and the computed response to the reader.
5. **Server Verification**: The reader forwards to the Authentication Server. The server retrieves the stored key for that ID and recalculates:
   * Access is granted if
6. **Replay Protection**: The server checks if the nonce was used recently. Nonces are valid only once within a 10-second window. Used nonces are stored temporarily in a Bloom filter for fast lookup.
7. **Logging and Alerts**: Every access attempt, valid or invalid, is logged with timestamp and IP. Repeated failures from a badge trigger an alert for potential cloning.

**Security Features:**

* Nonce-based challenge-response
* Symmetric-key HMAC authentication
* Replay prevention with Bloom filters

### **Scenario 7: Web Portal Brute Force Login**

**Objective:** To secure web portals against brute-force and dictionary attacks using login attempt throttling, multi-factor authentication, and session binding.

**Secure Web Authentication and Rate Limiting Protocol**

**Entities Involved:**

* End User
* Web Server
* Authentication Service
* MFA Token Service

**Steps:**

1. **Login Attempt Monitoring**: The server tracks the number of failed login attempts per user and per IP. Threshold set at 5 attempts per 10 minutes.
2. **Hashing and Verification**: Passwords are stored using Argon2id hashing with a random salt. At login:
3. **Account Lockout**: After 5 failures, the account is locked for 15 minutes. CAPTCHA and device re-verification are enforced.
4. **MFA Enforcement**: On successful password verification, a TOTP-based second factor is requested.
   * Token is verified using:
5. **Session Token Issuance**: A short-lived JWT is issued with a unique session ID, device fingerprint, and IP address:
6. **Anomaly Detection**: If the same account logs in from 3 locations within 1 hour, session re-authentication is triggered.
7. **Audit and Alerts**: All login events are logged with timestamps and geolocation. Repeated brute-force attempts from a source trigger a temporary IP block.

**Security Features:**

* Rate limiting and lockout
* Strong password hashing
* MFA with HMAC-based tokens
* Session anomaly detection

### **Scenario 9: IoT Smart Lock Data Modification**

**Objective:** To prevent interception or tampering of commands sent from a mobile application to a smart lock by enforcing encrypted and integrity-protected communication.

**Secure Smart Lock Command Protocol**

**Entities Involved:**

* Mobile Application
* Smart Lock Device
* IoT Cloud Broker

**Steps:**

1. **Session Initialization**: Perform ECDH key exchange between the mobile app and smart lock to establish a shared session key .
2. **Command Encryption**: Lock/unlock commands are encrypted using AES-GCM, including nonce and timestamp:
3. **HMAC Binding**: Each packet is signed using an HMAC to ensure integrity:
4. **Verification at Lock**: Lock decrypts the message, verifies MAC and ensures that timestamp is within ±5s of system clock.
5. **Replay Protection**: Nonces are stored in a sliding window. Duplicate nonces trigger denial and alerts.
6. **Acknowledge Response**: Lock sends back encrypted ACK with a fresh nonce. Logs both command and ACK for future dispute resolution.
7. **Cloud Sync and Alerts**: Every transaction is uploaded to the cloud dashboard, where admins can review command history and set usage policies.

**Security Features:**

* Ephemeral ECDH session keys
* AES-GCM encryption and integrity
* Replay resistance with timestamp and nonce

### **Scenario 10: OTP Replay Attack in Banking**

**Objective:** To bind one-time passwords (OTPs) to transaction-specific context and prevent replay attacks during high-risk financial operations.

**Context-Bound OTP Verification Protocol**

**Entities Involved:**

* Bank Customer
* Transaction Gateway
* SMS/Email OTP Delivery System

**Steps:**

1. **Transaction Context Generation**: User initiates a transaction with amount, payee ID, and device ID.
2. **OTP Computation**: Server computes an HMAC-based OTP with a shared secret:
3. **OTP Delivery**: The OTP is sent via SMS or secure push notification. A copy of the transaction context is stored on the server.
4. **User Confirmation**: The user inputs the OTP back into the app to approve the transaction.
5. **Validation and Context Matching**: Server validates:
   * OTP match
   * Timestamp within 30-second window
   * Matching payee ID and device
6. **Usage Marking**: Once validated, the OTP entry is marked as used in a Redis cache. Replays are rejected.
7. **Audit Logging and Alerts**:

### **Scenario 8: Shared Credentials in Research Lab**

**Objective:** To eliminate shared account usage, enforce individual accountability, and ensure timely revocation of access when personnel leave the organization.

**Individualized Authentication and Audit Protocol**

**Entities Involved:**

* Researcher
* Authentication System
* Centralized Database Server
* Access Auditor

**Steps:**

1. **Credential Assignment**: Each researcher is issued a unique user ID and password along with a second authentication factor, such as an OTP (One-Time Password) or hardware token (e.g., YubiKey). The password is stored using a secure cryptographic hash function like bcrypt or Argon2 to prevent reverse engineering even if the database is compromised.
2. **Access Proxying and Logging**: All requests to the database are routed through an authentication proxy. This proxy acts as a gatekeeper and injects metadata such as user ID, timestamp, and terminal IP into every query request. For example, a SELECT query is logged as: UserID: R123 | Time: 10:02 | IP: 192.168.1.8 | Query: SELECT \* FROM ChemResults WHERE ID = 5
3. **Role-Based Access Control (RBAC)**: Access rights are enforced using matrix-based RBAC where roles (e.g., Lab Assistant, Lead Researcher) are assigned specific permissions (read, write, delete). Permissions are stored as tuples (UserID, Role, Action, Resource) and verified using SQL-like policy rules.
4. **Deactivation upon Exit**: An HR-integrated automation script checks for employee status updates. Upon termination, the script triggers a revocation event which disables login and invalidates any session tokens issued to that user.
5. **Audit Logging**: Every user action is logged with cryptographic hashes for integrity. SHA-256 hash of the log entry is appended to a hash chain. This ensures that tampering with logs is evident. For example:  
   * Entry: Access by R123 at 10:15, Hash: H1
   * Next Entry: Access by R124 at 10:17, Hash: H2 = SHA256(H1 + new\_entry)
6. **Anomaly Detection**: The audit system runs heuristics and anomaly detection algorithms. For instance, if a researcher attempts access outside working hours or from an unregistered IP, the system flags it using threshold-based models or unsupervised clustering.
7. **Data Retention and Review**: Logs are retained for at least 12 months, and weekly reports are reviewed by the security team. The logs can also be verified using Merkle tree verification to ensure long-term integrity.

**Security Features:**

* Strong cryptographic authentication
* Modular log integrity via SHA-256
* RBAC enforced through policy algebra
* Automated revocation and anomaly alerts

### **Scenario 9: IoT Smart Lock Data Modification**

**Objective:** To prevent interception or tampering of commands sent from a mobile application to a smart lock by enforcing encrypted and integrity-protected communication.

**Secure Smart Lock Command Protocol**

**Entities Involved:**

* Mobile Application
* Smart Lock Device
* IoT Cloud Broker

**Steps:**

1. **Session Initialization**: Perform ECDH key exchange between the mobile app and smart lock to establish a shared session key .
2. **Command Encryption**: Lock/unlock commands are encrypted using AES-GCM, including nonce and timestamp:
3. **HMAC Binding**: Each packet is signed using an HMAC to ensure integrity:
4. **Verification at Lock**: Lock decrypts the message, verifies MAC and ensures that timestamp is within ±5s of system clock.
5. **Replay Protection**: Nonces are stored in a sliding window. Duplicate nonces trigger denial and alerts.
6. **Acknowledge Response**: Lock sends back encrypted ACK with a fresh nonce. Logs both command and ACK for future dispute resolution.
7. **Cloud Sync and Alerts**: Every transaction is uploaded to the cloud dashboard, where admins can review command history and set usage policies.

**Security Features:**

* Ephemeral ECDH session keys
* AES-GCM encryption and integrity
* Replay resistance with timestamp and nonce

### **Scenario 10: OTP Replay Attack in Banking**

**Objective:** To bind one-time passwords (OTPs) to transaction-specific context and prevent replay attacks during high-risk financial operations.

**Context-Bound OTP Verification Protocol**

**Entities Involved:**

* Bank Customer
* Transaction Gateway
* SMS/Email OTP Delivery System

**Steps:**

1. **Transaction Context Generation**: User initiates a transaction with amount, payee ID, and device ID.
2. **OTP Computation**: Server computes an HMAC-based OTP with a shared secret:
3. **OTP Delivery**: The OTP is sent via SMS or secure push notification. A copy of the transaction context is stored on the server.
4. **User Confirmation**: The user inputs the OTP back into the app to approve the transaction.
5. **Validation and Context Matching**: Server validates:
   * OTP match
   * Timestamp within 30-second window
   * Matching payee ID and device
6. **Usage Marking**: Once validated, the OTP entry is marked as used in a Redis cache. Replays are rejected.
7. **Audit Logging and Alerts**: If an OTP is reused or attempted from a different IP/device, the system logs the event and alerts the security team.

**Security Features:**

* Context-bound OTP with HMAC
* Time and use restrictions
* Replay and device anomaly detection

### **Scenario 11: Exposed API Keys in Cloud Services**

**Objective:** To ensure that cloud service API keys are never exposed publicly and are protected against unauthorized reuse or abuse.

**API Key Rotation and Validation Protocol**

**Entities Involved:**

* Developer
* Secrets Manager
* Cloud API Gateway
* Logging and Monitoring Agent

**Steps:**

1. **API Key Generation and Storage**: Keys are generated using a secure key generator and stored in a secrets manager (e.g., AWS Secrets Manager).
2. **Time-Scoped Tokens**: Instead of static keys, apps request short-lived tokens from the secrets manager:
   * Token validity: 15 minutes
   * Bound to IP and service
3. **Least Privilege Policy**: Each key is associated with a JSON IAM policy defining allowed actions. This is verified at the API gateway before execution.
4. **Key Rotation**: A Lambda function rotates keys every 24 hours. Expired keys are moved to a retired pool and denied access.
5. **Rate and Behavior Monitoring**: API gateway logs all usage metrics. Suspicious patterns (e.g., high-frequency access or unknown IPs) are detected using moving average thresholds.
6. **Revocation and Alerting**: On detection of a key leak (e.g., GitHub public repo), the secrets manager revokes the key and issues a rotation alert.
7. **Compliance Review**: Monthly audits compare usage logs against declared access controls and alert on any over-privileged use.

**Security Features:**

* Temporary key lifecycle enforcement
* IAM-bound privilege scoping
* Behavior anomaly alerting and auto-revocation

### **Scenario 12: Poor Session Management in E-Commerce**

**Objective:** To enhance user session security by enforcing expiration policies, session binding, and protection against token reuse.

**Tokenized Session Control Protocol**

**Entities Involved:**

* User Browser
* Web Server
* Session Management Service

**Steps:**

1. **Session Token Generation**: Upon successful login, the server issues a JWT with embedded session metadata:
2. **Secure Cookie Storage**: Token is stored in a SameSite=Strict, HTTP-only, Secure cookie to prevent XSS access.
3. **Binding to Device**: Token includes a fingerprint of the device (e.g., User-Agent + screen resolution + platform hash).
4. **Session Expiry**: Sessions expire in 15 minutes of inactivity and 6 hours max lifetime. Re-authentication is required afterward.
5. **Sensitive Action Re-validation**: Actions like password change or order confirmation require re-entry of password or OTP.
6. **Remote Logout and Session Tracking**: Users can view active sessions and revoke them individually.
7. **Token Reuse Detection**: Server checks for token reuse from different IP/device and invalidates it on mismatch.

**Security Features:**

* Strong session isolation
* Re-authentication for critical actions
* Real-time session control and invalidation

### **Scenario 13: Industrial Control System Command Tampering**

**Objective:** To ensure that commands to critical infrastructure (e.g., water treatment systems) are authenticated, encrypted, and verifiable.

**Authenticated ICS Command Protocol**

**Entities Involved:**

* Operator Console
* ICS Gateway
* Programmable Logic Controller (PLC)
* Security Event Monitoring System

**Steps:**

1. **Operator Authentication**: Console enforces MFA and digital certificate validation for login. Certificates are validated against a Certificate Authority.
2. **Session Key Establishment**: Use Diffie-Hellman key exchange to derive session key:
3. **Command Encryption**: Commands are wrapped with a timestamp, command ID, and sequence number. Payload is encrypted using AES-GCM:
4. **MAC Verification and Replay Prevention**: PLC validates timestamp freshness and ensures sequence number is incrementing.
5. **Command Execution and Logging**: Valid commands are executed and logged. Logs are hashed and appended to an immutable ledger using:
6. **Live Event Stream**: Logs are streamed to the central monitoring system. Alerts are generated for MAC failures or out-of-order sequences.
7. **Emergency Shutdown**: On detection of command tampering, PLC triggers an auto-lockout and switches to manual override mode.

**Security Features:**

* Command integrity and authentication
* Tamper-evident logs and ledger
* Real-time alerting and fail-safe lockout