hello@advix.com
advix.com

BITCOIN.COM SECURITY ASSESSMENT AND PENETRATION TESTING REPORT

March 28, 2025, Version 1.0

Table of contents

1. Executive Summary

Advix has completed the security assessment of the code outlined in the Assessment Scope and Objectives section together with the penetration test.

Our findings show that the newly implemented authentication scheme provides strong security, utilizing high-assurance authentication factors such as Google OAuth and WebAuthn.

However, the analysis identified several deviations from true Multi-Party Computation (MPC) protocol specifications. Some of these deviations represent intentional design choices made to enhance user experience, while others present opportunities for security improvements, detailed below.

The overall architecture maintains strong authentication principles while balancing practical usability considerations.

In addition, cryptographic primitives are implemented correctly, with appropriate consideration given to the operational modes of the underlying algorithms. The selected cryptographic schemes are deemed secure against both current threats and potential future attack vectors.

2. Assessment Scope and Objectives

The assessment scope included the codebase and API/web endpoints in the following repositories:

- https://github.com/bitcoin-portal/mpc-sdk and related resources
- https://dev.accounts.bitcoin.com

The goal of the assessment was to identify security vulnerabilities, code quality issues, and potential weaknesses in exposed endpoints.

Additional resources related to main scope:

- https://tradfi-web.ops.bitcoin.com
- https://markets-ws.api.bitcoin.com

3. Assessment Methodology

Our approach is based on OWASP methodology and implies using advanced automated tools combined with expert manual review, providing a comprehensive assessment across all critical layers of the system.

Our assessment methodology focuses on verifying the implementation of key security controls across critical domains to ensure comprehensive protection of systems and data:

- **Architecture:** Reviews the design and security functions of application components, ensuring a robust and secure architecture.
- **Authentication:** Verifies that user identity verification processes and authenticator management follow best practices for secure authentication.
- **Session Management:** Assesses how sessions are managed, ensuring secure handling of session creation, maintenance, and termination to prevent unauthorized access.
- **Access Control:** Confirms the enforcement of proper restrictions on user permissions and resource access, adhering to the principle of least privilege.
- **Validation, Sanitization, and Encoding:** Ensures proper validation and sanitization of inputs to mitigate injection attacks and other security risks.
- **Stored Cryptography:** Verifies the effective use of encryption for protecting sensitive data stored at rest.
- **Error Handling and Logging:** Evaluates error management practices to ensure sensitive information is not exposed in error messages or logs.
- **Data Protection:** Confirms that sensitive information is securely protected both at rest and in transit, with appropriate encryption and safeguards.
- **Communication:** Ensures secure communication protocols are in place, maintaining data confidentiality and integrity during transmission.
- **Business Logic:** Reviews business workflows to ensure they cannot be manipulated or bypassed by attackers.
- **API and Web Service Security:** Verifies proper security measures for APIs and web services, focusing on authentication, authorization, and encryption.
- **Configuration:** Assesses system configuration to ensure secure default settings and hardening measures are in place to reduce vulnerabilities.

This approach ensures that all critical areas are thoroughly evaluated to maintain a high standard of security.

4. SDK Code Summary

The reviewed SDK acts primarily as a React + TypeScript wrapper around the closed-source `@sodot/sodot-web-sdk` library. It exposes selected functionality from the upstream SDK to the application layer but contains limited custom business logic.

While the codebase includes an extensive number of test files, effective test coverage is inconsistent. **Overall coverage is estimated at 58%**, with critical areas—such as backup and restore workflows—lacking meaningful tests. Notably, files like `mpc-sdk.ts` and those under `src/api/gdrive/` **currently exhibit 0% coverage**.

Key observations:

- Cryptographic primitives (e.g., AES-GCM, base64url) appear to be used correctly. However, the rationale for using a custom base64url implementation, instead of standard libraries, is unclear.
- Certain protocol elements, such as BIP-340 support, remain unimplemented, although marked as TODOs in the code.
- `ESLint` is configured but not formally declared as a dev dependency. Moreover, the current linting ruleset is permissive and lacks enforcement of common style conventions, which may hinder maintainability.

No direct vulnerabilities or insecure coding patterns were identified within the SDK wrapper. However, as the underlying `@sodot/sodot-web-sdk` library is closed-source, its cryptographic correctness and protocol adherence could not be independently verified. Integration patterns suggest reliance on upstream abstractions and documentation.

5. Key Findings

The assessment surfaced a series of misconfigurations, protocol handling weaknesses, and potential abuse vectors. Below is a high-level summary of the most critical issues.

While no remote code execution or severe authentication bypass was identified, the issues below weaken the system's security boundary and should be prioritized accordingly.

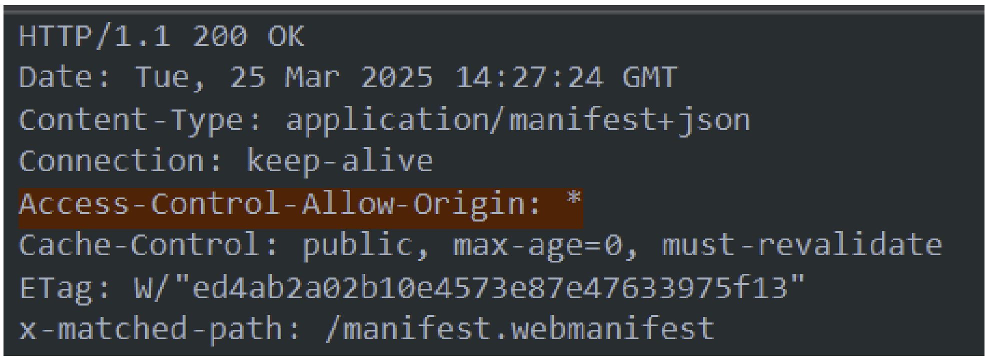
Severity classification:

- **Critical**: Exploitable vulnerabilities leading to full system compromise, data breaches, or remote code execution. Immediate remediation required.
- **High**: Serious security flaws allowing unauthorized access, privilege escalation, or major data leaks. Requires prompt attention
- **Medium**: Issues that could be exploited under certain conditions, such as security misconfigurations or weak access controls. Should be addressed in a reasonable timeframe.
- **Low**: Minor weaknesses with limited impact, often requiring other vulnerabilities to be exploitable. Address as part of routine maintenance.

A complete list is provided in the Appendix.

5.1 Cross-Domain Misconfiguration

Severity	Medium
Description	Server response contains header `Access-Control-Allow-Origin: *`, that allows requests from any third party domains. 1. **Data Exposure**: Any website can make requests to your API and access the response without additional protection 2. **CSRF & XSS**: If combined with authentication, attackers can exploit users' sessions 3. **API Abuse**: Unrestricted access may allow automated scraping or abuse.
Mitigation	• Configure `Access-Control-Allow-Origin` to more restricted set of valid domains - or - • Remove this header completely, let the browser enforce Same Origin policy by default.

Evidence of Vulnerability	

```
HTTP/1.1 200 OK
Date: Tue, 25 Mar 2025 14:27:24 GMT
Content-Type: application/manifest+json
Connection: keep-alive
Access-Control-Allow-Origin: *
Cache-Control: public, max-age=0, must-revalidate
ETag: W/"ed4ab2a02b10e4573e87e47633975f13"
x-matched-path: /manifest.webmanifest
```

5.2 Protocol Violation

Severity	Medium
Description	True MPC requires key shares to be generated independently by different parties
Mitigation	Ensure that key shares are generated independently by multiple parties

6. Solution Architecture Recommendations

To improve the platform's long-term security posture and align with the intended guarantees of MPC, the following architectural changes are recommended.

6.1. Implement End-to-End Encrypted Cloud Backup

All cloud backup workflows should use strong client-side encryption with locally generated keys. This ensures data confidentiality even if the underlying cloud infrastructure is compromised.

Context: Recent industry vulnerabilities underscore the criticality of proper encryption in distributed key management:

- GG18 and GG20 Paillier Key Vulnerability - Fireblocks
- Bitforge Vulnerability - SafeHeron
- Breaking the Shared Key in Threshold Signature Schemes - Trail of Bits

6.2. Distribute MPC Nodes Across Isolated Environments

1. To strengthen the Multi-Party Computation (MPC) security model, deploy MPC nodes across independent and isolated environments, such as separate cloud providers or secure enclaves. This distribution minimizes the risk of coordinated attacks and enhances system resilience.
2. Where appropriate, integrate cloud-based Hardware Security Modules (HSMs) like AWS Nitro Enclaves to protect server-side key shares using hardware-backed isolation.

6.3. Decentralize Architecture to Uphold MPC Integrity

Revise the system architecture to eliminate the centralization of critical infrastructure components — such as the server, relay, and authentication logic — which inherently conflict with the trust distribution principles of MPC.

Adopt a decentralized model where:

- No single component retains control over encrypted key shares or user backups.
- The relay server operates as a passive transport layer and does not have visibility into sensitive MPC communications during key operations.

7. Conclusion and Next Steps

The assessment confirms that Bitcoin.com's authentication framework and cryptographic primitives are generally well-implemented. However, certain design decisions and configuration gaps reduce the effectiveness of the MPC protocol and weaken the system's perimeter defenses.

We recommend the following immediate next steps:

1. **Short-term (0–2 weeks)**
   a. Address misconfigurations in headers (CSP, CORS, STS, X-Powered-By):
      i. 5.1 Cross-Domain Misconfiguration.
      ii. 5.2 Information Leak via X-Powered-By Header.
2. **Mid-term (2–4 weeks)**
   a. Harden testing practices for coverage of cryptographic and backup logic.
3. **Long-term (1–2 months)**
   a. Evaluate the feasibility of moving toward a more resilient and decentralized MPC architecture.
   b. In the event of substantial architecture changes, a follow-up audit should be conducted to validate the applied fixes.

Advix remains available to support the implementation of these recommendations or to provide validation upon completion.

8. About Us

Advix is a boutique technology consulting firm specializing in both traditional and emerging fintech. Applying our deep cybersecurity expertise, we help banks, exchanges, and web3 startups improve their security posture.

The present assessment and penetration testing were conducted by our advisors:

- **Yaroslav Rabovolyuk**, Cybersecurity Advisor – ex-CISO VK Group, 200+ projects in offensive cybersecurity.
- **Dmitry Yanchenko**, Cybersecurity Advisor – 20+ years in defensive cybersecurity, PCI DSS audits, NIST, SOC.

Contact Information

Should you have any questions regarding this report, please contact
Yaroslav Rabovolyuk: y.rabovolyuk@advix.com, or
Nail Iangazov: n.iangazov@advix.com

Sergey Kubasov
CEO, Advix FZCO
Dubai Silicon Oasis,
DDP, Building A1
United Arab Emirates

[illegible]

9. Appendix: Full List of Findings

9.1. File: `api/mpc/sign.ts`

#	Issue	Risk	Solution	Status
1	**Party coordination issues**: `num_parties: numParties // Client-controlled value`	**Severity: High** A malicious client could bypass threshold-based security guarantees by setting `num_parties=1`, effectively disabling MPC protections.	Enforce server-side validation of `num_parties` to ensure it meets the minimum threshold requirements for secure MPC execution.	Resolved
2	**Batch signing**: `sign_requests: signRequests.map(...) // Atomic handling`	**Severity: High** Absence of per-request nonce binding may allow attackers to splice elements from different batches, compromising integrity and traceability.	Bind a unique nonce to each signing request and enforce atomic processing of individual elements within the batch.	Resolved

9.2. File: `classes/sign.ts`

#	Issue	Risk	Solution	Status
1	**Relay server trust**: `new window.sodot.Ecdsa(this.relayUrl) // Trusted relay?`	**Severity: Medium** A malicious relay server, while unable to see secret data, could still disrupt the protocol's behavior or flow.	Validate and explicitly trust relay servers. Use secure connections and introduce verification mechanisms to prevent unauthorized interference.	Resolved
2	**Share persistence**: `const secret = await aesGcm.decrypt(...) // Clear after use?`	**Severity: Medium** Decrypted shares briefly remain in memory, introducing a potential window for in-memory attacks or data leaks.	Ensure decrypted data is immediately cleared from memory after use to reduce exposure.	Resolved

9.3. File: `types/mpc.ts`

#	Issue	Risk	Solution	Status
1	**Threshold protocol gaps**: `export type InitKeygenCreationResponse = { threshold: number };`	**Severity: High** The server may override the client's threshold preference, potentially leading to incorrect or insecure key generation.	Include the threshold field in the keygen response and implement client-side validation to ensure consistency.	Resolved
2	**Multi-Party coordination gaps**: `export type MpcSignResponse = { room_id: string; // No participant list };`	**Severity: High** Clients cannot verify the identities of other participants in the MPC session, which may enable man-in-the-middle (MITM) attacks.	Introduce participant verification and ensure secure identification mechanisms are integrated into protocol flows.	Resolved
3	**Authentication bypass vectors**: `type PublicKeyCredentialRequestOptionsJSON = { // No userVerification requirement // No attestation conveyance };`	**Severity: Low** Weak authentication may occur if WebAuthn settings lack essential security flags, such as `userVerification` or `attestation`.	Enforce inclusion of `userVerification` and `attestationConveyance` in WebAuthn typings to ensure robust authentication.	Resolved

9.4. File: `api/mpc/keygen.ts`

#	Issue	Risk	Solution	Status
1	**Threshold validation vulnerability**: `const numParties = 3; const threshold = 5; // Could be > numParties`	**Severity: High** A client-controlled threshold greater than numParties may undermine the core guarantees of threshold cryptography, potentially allowing invalid or insecure key operations.	Implement server-side validation to ensure `threshold ≤ numParties`, enforcing secure and consistent configuration.	Resolved
2	**Keygen ID trust issue**: `createKeygen({ keygenIds: ["malicious-id"] }) // No validation`	**Severity: High** An attacker may forge or manipulate `keygenId` values, potentially undermining the integrity of the key generation workflow.	Introduce strict validation by implementing a predefined allowlist of authorized keygenId values and enforce cryptographic verification of their authenticity.	Resolved

9.5. File: `classes/generate.ts`

#	Issue	Risk	Solution	Status
1	**Protocol violation**: `signer.keygen(..., numParties, threshold, ...); // Client controls threshold`	**Severity: Medium** True MPC requires key shares to be generated independently by different parties	Ensure that key shares are generated independently by multiple parties	Actual
2	**Threshold parameter mismanagement**: `signer.keygen(..., numParties, threshold, ...); // Server doesn't validate threshold <= numParties`	**Severity: High** Client-controlled threshold could bypass MPC security, violating protocol	Validate threshold to ensure it is `≤ numParties` before key generation	Resolved
3	**Keygen ID poisoning**: `[backupKey.keygenId, serverKeygenDetails.keygen_id] // Trusting client-generated IDs`	**Severity: High** Malicious client could inject invalid keygen IDs, compromising the security	Validate keygen IDs to prevent poisoning and enforce a secure ID generation process	Resolved

9.6. File: `api/mpc/reshare.ts`

#	Issue	Risk	Solution	Status
1	**Threshold reconfiguration vulnerability**: `new_threshold: newThreshold // No validation`	**Severity: High** An attacker could set the threshold below a secure minimum, potentially enabling a single party to reconstruct the secret.	Enforce server-side validation to ensure the new threshold is not lower than the minimum required for safe operation.	Resolved
2	**Keygen ID trust chain break**: `keygen_ids: keygenIds // Could include malicious IDs`	**Severity: High** Attackers could supply malicious or revoked keygenIds, undermining the integrity of the share generation process.	Implement validation mechanisms to verify the authenticity of each `keygenId` before accepting it.	Resolved
3	**Reshare initiation spoofing**: `initReshareExistingParty({ walletId }); // No proof of existing share ownership`	**Severity: High** Any user could trigger resharing for a wallet they do not own, leading to potential full system compromise.	Add authentication checks to verify the initiator holds a valid share before allowing resharing actions.	Resolved

9.7. Web App (dev.accounts.bitcoin.com)

#	Issue	Risk	Solution	Status
1	**Content Security Policy (CSP) is not implemented**	**Severity: Medium** The application does not implement a Content Security Policy, which helps prevent Cross-Site Scripting (XSS) and data injection attacks.	Implement CSP by adding a `Content-Security-Policy` header. Define allowed origins for each resource type to restrict external scripts, styles, and other assets.	Actual
2	**Cross-domain misconfiguration**	**Severity: Medium** The server allows requests from any third-party domain, exposing the application to potential data leakage and cross-origin abuse.	Restrict the `Access-Control-Allow-Origin` header to a specific list of trusted domains, or remove it entirely to rely on the browser's default same-origin policy.	Actual
3	**Information leak via X-Powered-By header**	**Severity: Medium** The `X-Powered-By` header reveals backend technologies, helping attackers identify frameworks and tailor attacks accordingly.	Remove or suppress the `X-Powered-By` header from all server responses..	Resolved
4	**Missing STS header**	**Severity: Low** Without the `Strict-Transport-Security` header, browsers may not enforce HTTPS, leaving users vulnerable to man-in-the-middle (MitM) attacks.	Ensure the server includes a `Strict-Transport-Security` header in responses to enforce secure connections.	Actual

10. Version history

Version	Date	Changes
1.0	28.03.2025	The first version of the report has been published.
1.1	07.04.2025	Minor updates have been made to Section 5.3
1.2	08.04.2025	Re-evaluation