CERTIK

AB

Security Assessment

CertiK Assessed on Mar 13th, 2025

CertiK Assessed on Mar 13th, 2025

AB

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Layer 1	Ethereum (ETH)	Manual Review, Static Analysis, Testnet Deployment

LANGUAGE	TIMELINE	KEY COMPONENTS
Golang	Delivered on 03/13/2025	N/A

CODEBASE

abiot:593e84644606a733d73e29358f289f8721a0214c

newchain:593e84644606a733d73e29358f289f8721a0214c

View All in Codebase Page

COMMITS

abiot:593e84644606a733d73e29358f289f8721a0214c

newchain:593e84644606a733d73e29358f289f8721a0214c

View All in Codebase Page

Vulnerability Summary

14	11	0	0	3	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
4	Major	4 Resolved	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
4	Medium	4 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	3 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
1	Informational	1 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | AB

CODEBASE | AB

Repository

abiot:593e84644606a733d73e29358f289f8721a0214c newchain:593e84644606a733d73e29358f289f8721a0214c

Commit

abiot:593e84644606a733d73e29358f289f8721a0214c newchain:593e84644606a733d73e29358f289f8721a0214c

AUDIT SCOPE | AB

6 files audited ● 2 files with Acknowledged findings ● 2 files with Resolved findings ● 2 files without findings

ID	Repo	File	SHA256 Checksum
BFG	ABFoundationGlobal/abiot	eth/gasprice/gasprice.go	bba823351a336278e5a71a4959d34c0ab60f5a3d0e28e7ba0f7ae7d783f846e1
SIG	ABFoundationGlobal/abiot	crypto/signature_r1.go	985f667ae82efdb8ef1d1d22b249eeef458aaa6eb5ae5e1db5f948f8911ae30c
ABG	ABFoundationGlobal/abiot	consensus/clique/clique.go	6fd0db5b04cd8ca3b0cfe723240295ca3abec846952fb28e680430aab0130e0d
CRY	ABFoundationGlobal/abiot	crypto/crypto.go	193f83a2ae29f0404ec8b54a1161f6421e359d578ccb4a5687a19a29215ef8c0
ABF	ABFoundationGlobal/abiot	consensus/clique/api.go	a76b611f117b2298061f4d5f98a78ca9a6253c7da9bf70ab55a2292251c64351
AFG	ABFoundationGlobal/abiot	consensus/clique/snapshot.go	9343b62de5872fb300e2d275388d01517e4bb4f47d1a87ce3f90ea0f7526974f

APPROACH & METHODS | AB

This report has been prepared for AB to discover issues and vulnerabilities in the source code of the AB project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review, Static Analysis, and Testnet Deployment techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

REVIEW NOTE | AB

The AB Chain is a fork of go-ethereum `v1.9.18` that introduces several key modifications, including a change in the cryptographic algorithm, adjustments to the gas market, and the implementation of a Proof of Authority (POA) consensus engine. Below is the scope of the auditing engagement:

- **Crypto Module**:
  The AB Chain replaces the `secp256k1` cryptographic algorithm with `secp256r1`, distinguishing it from the original geth implementation.
- **Clique Consensus Module**:
  Instead of the default consensus mechanism, the AB Chain employs a POA (Proof of Authority) consensus model.
- **Other Modifications**:
  The gas market has been adjusted, with corresponding updates to its verification mechanisms.
  Additionally, known vulnerabilities from the latest upstream go-ethereum release, specifically within the Execution Layer (EL), have been addressed.

FINDINGS | AB

14	0	4	4	5	1
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for AB. Through this audit, we have uncovered 14 issues ranging from different severity levels. Utilizing the techniques of Manual Review, Static Analysis & Testnet Deployment to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GO5-01	Denial Of Service Due To Go CVE-2020-28362	Denial of Service	Major	Resolved
PEE-01	DoS Attack Via Malicious P2p Message By Dumped Ping Requests	Logical Issue	Major	Resolved
SIG-02	Missing Signature Malleability Validation On `secp256r1`	Coding Style	Major	Resolved
STA-01	Potential Consensus Flaw During Transaction Processing Within StateDB	Logical Issue	Major	Resolved
CLI-01	Missing Checks On Gas Related In Clique	Volatile Code, Logical Issue	Medium	Resolved
CRY-01	Susceptible To Signature Malleability Allowed	Volatile Code	Medium	Resolved
SER-01	Potential DoS Attack In LES Server Via GetProofsV2 Msg	Denial of Service	Medium	Resolved
SIG-03	Incorrect Public Key Comparison In `comparePublicKey` Function	Logical Issue, Volatile Code	Medium	Resolved
593-01	Potential Issues From Upstream `go-ethereum`	Logical Issue	Minor	Resolved
CLI-02	Potential Race Condition With `signer`	Concurrency	Minor	Resolved
GAS-01	Unhandled Errors In `gasprice`	Volatile Code	Minor	Acknowledged

ID	Title	Category	Severity	Status
GAS-02	Volatile Code For Index In `txPrices` Slice	Volatile Code, Logical Issue	Minor	Acknowledged
PER-01	Potential DoS Attack Via Malicious P2p Message	Volatile Code, Denial of Service	Minor	Resolved
SIG-01	Concerns On Inconsistent Signature Validation In `checkSignature` Function	Inconsistency	Informational	Acknowledged

GO5-01 | DENIAL OF SERVICE DUE TO GO CVE-2020-28362

Category	Severity	Location	Status
Denial of Service	Major	go.mod: 3; go.mod (v1.9.18-newton): 3	Resolved

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `go.mod`

The chain application is currently using Go version 1.13, as specified in the `go.mod` file. This version is affected by a critical Denial of Service (DoS) vulnerability, registered as 'CVE-2020-28362'. This vulnerability impacts Geth versions built with Go versions less than 1.15.5 or less than 1.14.12. The flaw could potentially be exploited to disrupt service availability.

References

1. `golang-announce`
2. `GHSA-m6gx-rhvj-fh52`

Recommendation

Recommend to rebuild with Go 1.15.5 or 1.14.12 above, but be cautious with the version v1.19 and v1.21, according to the upstream geth PRs: `build: upgrade to go 1.19 #25726` , `all: update to go version 1.22.1 #28946` .

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding by rebuilding with Go 1.22. The change is reflected in the commit `391f1ae549292b67acbbaa04ecc68f93b8cc817a` .

PEE-01 DOS ATTACK VIA MALICIOUS P2P MESSAGE BY DUMPED PING REQUESTS

Category	Severity	Location	Status
Logical Issue	Major	p2p/peer.go: 250~266; p2p/peer.go (v1.9.18-newton): 250~266	Resolved

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `p2p/peer.go`

In the `pingLoop` for peer to handling the ping requests with `pingMsg`, there is no limit for ping requests from a single peer.

`p2p/peer.go`

```
func (p *Peer) pingLoop() {
    ping := time.NewTimer(pingInterval)
    defer p.wg.Done()
    defer ping.Stop()
    for {
        select {
        case <-ping.C:
            if err := SendItems(p.rw, pingMsg); err != nil {
                p.protoErr <- err
                return
            }
            ping.Reset(pingInterval)
        case <-p.closed:
            return
        }
    }
}
```

So there could be a DoS scenario by flooding a node with ping requests, and an unbounded number of goroutines can be created, leading to resource exhaustion and potentially crash due to OOM(Out Of Memory). This issue has been disclosed in CVE as well as GHSA in community(See References below), and it's highly recommended to fix it before the main net launch.

References:

- `CVE-2023-40591`
- `GHSA-ppjg-v974-84cm`

Recommendation

Recommend to restrict ping requests as upstream geth `p2p: move ping handling into pingLoop goroutine #27887`

Alleviation

[AB-Chain Team - 03/03/2025] :
The team heeded our advice and resolved the finding by restricting ping requests. The change is reflected in the commit `7bc85d773097cc506a47100770678a4d388ed8d0` .

SIG-02 MISSING SIGNATURE MALLEABILITY VALIDATION ON `secp256r1`

Category	Severity	Location	Status
Coding Style	Major	crypto/signature_r1.go: 132; crypto/signature_r1.go (v1.9.18-newton): 132	Resolved

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `crypto/signature_r1.go`
- `crypto/signature_test.go`

According to the `signature_test.go`, the `VerifySignature` should reject malleable signatures with upper s. However, the current implementation does not verify if the upper s provided. As a result, the attacker may build valid signatures by modifying the s.

Proof of Concept

```go
func TestUpperSVerification(t *testing.T) {
    // Generate a private key
    key, _ := GenerateKey()

    // Message to sign
    msg := []byte("test message")
    hash := Keccak256(msg)

    // Sign the message normally
    sig, err := Sign(hash, key)
    if err != nil {
        t.Fatal(err)
    }

    // Verify normal signature
    pubKey := CompressPubkey(&key.PublicKey)
    if !VerifySignature(pubKey, hash, sig[:64]) {
        t.Error("Failed to verify valid signature")
    }

    // Modify s to be in upper half
    curveOrderByteSize := S256().Params().P.BitLen() / 8
    s := new(big.Int).SetBytes(sig[curveOrderByteSize : 2*curveOrderByteSize])
    upperS := new(big.Int).Sub(secp256r1N, s)

    // Create modified signature with upper s
    modifiedSig := make([]byte, 64)
    copy(modifiedSig[:curveOrderByteSize], sig[:curveOrderByteSize])
    copy(modifiedSig[curveOrderByteSize:], upperS.Bytes())

    // Verify modified signature with upper s
    if VerifySignature(pubKey, hash, modifiedSig) {
        t.Error("Should not to verify signature with upper s")
    }
}
```

Recommendation

Recommend adding validation on upper `s` to avoid malleable signatures.

Alleviation

[AB-Chain Team - 03/07/2025] :

The team heeded our advice and resolved the finding with validation on upper s. The change is reflected in the commit `7d0c3859528973fec2799e18d4bb44338ec3ce9a` .

STA-01 POTENTIAL CONSENSUS FLAW DURING TRANSACTION PROCESSING WITHIN STATEDB

Category	Severity	Location	Status
Logical Issue	Major	core/state/statedb.go: 591~592; core/state/statedb.go (v1.9.18-newton): 591~592	Resolved

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `core/state/statedb.go`

A particular sequence of transactions can cause a consensus failure due to Geth's incorrect handling of account balances after a self-destruct operation. The issue arises when a self-destructed account, `0xaa`, is involved in subsequent transactions.

- **Transaction 1**:
  - The sender invokes `caller`.
  - `Caller` invokes `0xaa`, which has 3 wei and performs a self-destruct-to-self.
  - `Caller` then makes a 1 wei call to `0xaa`. Since the transaction is ongoing, `0xaa` executes code, but it doesn't redo the self-destruct.
- **Transaction 2**:
  - The sender makes a 5 wei call to `0xaa`. No execution occurs as there is no code.

In Geth, 0xaa ends up with 6 wei, whereas Open Ethereum correctly reports 5 wei. If the second transaction is not executed in Geth, `0xaa` is destructed, resulting in 0 wei. This discrepancy is due to Geth's incorrect balance handling after self-

destruct and subsequent transactions, causing potential consensus failures as different nodes may disagree on the state of `0xaa`.

The root cause is traced to a change in the `createObject` function, which now returns a non-nil object (with `deleted=true`) even if the account was destructed, leading to the new object inheriting the old balance:

```
func (s *StateDB) CreateAccount(addr common.Address) {
    newObj, prev := s.createObject(addr)
    if prev != nil {
        newObj.setBalance(prev.data.Balance)
    }
}
```

Reference:

- `GHSA-xw37-57qp-9mm4`
- `CVE-2020-26265`

Recommendation

Recommend to perform a minimal fix as below:

```
+++ b/core/state/statedb.go
@@ -589,7 +589,10 @@ func (s *StateDB) createObject(addr common.Address) (newobj,
prev *stateObject)
                s.journal.append(resetObjectChange{prev: prev, prevdestruct:
prevdestruct})
        }
        s.setStateObject(newobj)
-       return newobj, prev
+       if prev != nil && !prev.deleted {
+               return newobj, prev
+       }
+       return newobj, nil
```

or upgrade the codebase to Geth v1.9.20 `Paragade`.

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding with the minimal fix. The change is reflected in the commit `e5c074185b46624262ba01429047fdd82c41c406`.

CLI-01 | MISSING CHECKS ON GAS RELATED IN CLIQUE

Category	Severity	Location	Status
Volatile Code, Logical Issue	Medium	consensus/clique/clique.go: 299; consensus/clique/clique.go (v1.9.18-newton): 299	Resolved

Description

Repository:

- `AB Chain`

Commit hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `consensus/clique/clique.go`

The version of the clique provided in the AB chain is missing validation on `GasLimit` and `GasUsed`, which would accepted unexpected headers in e.g. `light` sync mode by verifying the headers without checking the gas metrics to keep consistency.

Reference:

- `consensus/clique: add some missing checks #22836`

Recommendation

Recommend adding corresponding validations on `GasLimit` and `GasUsed` according to `consensus/clique: add some missing checks #22836`.

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding with validations on gas. The change is reflected in the commit `35b4607da9645225444343bd604a3a2e2b366531`.

CRY-01 | SUSCEPTIBLE TO SIGNATURE MALLEABILITY ALLOWED

Category	Severity	Location	Status
Volatile Code	Medium	crypto/crypto.go: 250~262; crypto/crypto.go (v1.9.18-newton): 250~262	Resolved

Description

Repository:

- `AB Chain`

Commit hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `crypto/crypto.go`

The `ValidateSignatureValues` function checks the validity of the r and s values in an ECDSA signature, ensuring that they fall within a valid range. However, the current implementation allows for signature malleability due to the lack of a check to restrict the s value to the lower half of the curve's `order. As a result, an attacker could craft multiple valid signatures for the same message by modifying the s value, potentially allowing for signature manipulation or replay attacks.

`crypto/crypto.go`

```
248 // ValidateSignatureValues verifies whether the signature values are valid with
249 // the given chain rules. The v value is assumed to be either 0 or 1.
250 func ValidateSignatureValues(v byte, r, s *big.Int, homestead bool) bool {
251     if r.Cmp(common.Big1) < 0 || s.Cmp(common.Big1) < 0 {
252         return false
253     }
254     // reject upper range of s values (ECDSA malleability)
255     // see discussion in secp256k1/libsecp256k1/include/secp256k1.h
256     //if homestead && s.Cmp(secp256k1_halfN) > 0 {
257     //  return false
258     //}
259     // Frontier: allow s to be in full N range
260     return r.Cmp(secp256r1N) < 0 && s.Cmp(secp256r1N) < 0 && (v == 0 || v == 1)
261 }
```

Recommendation

Recommend adding validation on `s` to ensure it lies in the lower half of the valid range.

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding with validation on upper s. The change is reflected in the commit `72edfdd66bf4378e082e0cbc99a6de540188fb09` .

SER-01 POTENTIAL DOS ATTACK IN LES SERVER VIA GETPROOFSV2 MSG

Category	Severity	Location	Status
Denial of Service	Medium	les/server_handler.go: 599~602; les/server_handler.go (v1.9.18-newton): 599~602	Resolved

Description

Repository:

- `AB Chain`

Commit Hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `les/server_handler.go`

A Denial of Service (DoS) vulnerability exists in the LES (Light Ethereum Subprotocol) server when processing a malicious `GetProofsV2` request from a connected LES client. The vulnerability is triggered when the `request.BHash` is equal to a specific `Hash`, resulting in a nil pointer dereference(header being nil) which can cause the server to panic and crash.

The issue arises in the following code snippet:

```
var (
    header *types.Header
    trie   state.Trie
)
if request.BHash != lastBHash {
  //...
}
// If a header lookup failed (non existent), ignore subsequent requests for the same
header
if root == (common.Hash{}) {
    atomic.AddUint32(&p.invalidCount, 1)
    continue
}
// Open the account or storage trie for the request
statedb := h.blockchain.StateCache()

switch len(request.AccKey) {
case 0:
    // No account key specified, open an account trie
    trie, err = statedb.OpenTrie(root)
    if trie == nil || err != nil {
        p.Log().Warn("Failed to open storage trie for proof", "block",
header.Number, "hash", header.Hash(), "root", root, "err", err)
        continue
    }
default:
    // Account key specified, open a storage trie
    account, err := h.getAccount(statedb.TrieDB(), root,
common.BytesToHash(request.AccKey))
    if err != nil {
        p.Log().Warn("Failed to retrieve account for proof", "block", header.Number,
"hash", header.Hash(), "account", common.BytesToHash(request.AccKey), "err", err)
        atomic.AddUint32(&p.invalidCount, 1)
        continue
    }
```

Recommendation

Recommend to fix the issue according to the `PR-21896`

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding with validations on gas. The change is reflected in the commit `abf3bda3e9663d53230a93a9dc1ca78023dc5b5e` .

SIG-03 INCORRECT PUBLIC KEY COMPARISON IN `comparePublicKey` FUNCTION

Category	Severity	Location	Status
Logical Issue, Volatile Code	Medium	crypto/signature_r1.go: 237; crypto/signature_r1.go (v1.9.18-newton): 237	Resolved

Description

Repository:

- `AB Chain`

Commit Hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `crypto/signature_r1.go`

The `comparePublicKey` function is intended to compare two ECDSA public keys to determine if they are identical. However, there is a logical error in the comparison of the `Y` coordinate. The function incorrectly compares `key2.Y` to itself instead of comparing `key1.Y` to `key2.Y`, which can lead to false positives. Below is the problematic code snippet:

```
func comparePublicKey(key1, key2 *ecdsa.PublicKey) bool {
    // TODO: compare curve
    x := key1.X.Cmp(key2.X)
    y := key2.Y.Cmp(key2.Y) // Incorrect comparison
    if x == 0 && y == 0 {
        return true
    } else {
        return false
    }
}
```

Recommendation

Ensure that the Y coordinate of key1 is compared to the Y coordinate of key2. The corrected code should look like this:

```
func comparePublicKey(key1, key2 *ecdsa.PublicKey) bool {
    // TODO: compare curve
    x := key1.X.Cmp(key2.X)
    y := key1.Y.Cmp(key2.Y)
    if x == 0 && y == 0 {
        return true
    } else {
        return false
    }
}
```

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding by updating the `y` comparison. The change is reflected in the commit `0922214118362272eec65cce9d5ebd4b2bc502c0` .

593-01 | POTENTIAL ISSUES FROM UPSTREAM `go-ethereum`

Category	Severity	Location	Status
Logical Issue	Minor	accounts/abi/topics.go: 45; accounts/abi/topics.go (v1.9.18-newton): 45	Resolved

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `accounts/abi/topics.go`

The bigInt encoding is wrong in the contract topic if the bigInt is a negative number. For example, the bigInt(-1) is encoded as 0000000000...000000000001, while it should be ffffffffff...ffff instead.

Recommend to fix the bigInt encoding issue according to the `PR-28764`

Recommendation

Recommends to fix the issue above according to the corresponding PRs from go-ethereum.

Alleviation

[AB-Chain Team - 03/05/2025] :

The team heeded our advice and resolved the finding by restricting ping requests. The change is reflected in the commit `fb352c48eead4c9cd8cef69e4ccbfd0f14f494c6` .

CLI-02 | POTENTIAL RACE CONDITION WITH `signer`

Category	Severity	Location	Status
Concurrency	Minor	consensus/clique/clique.go: 502; consensus/clique/clique.go (v1.9.18-newton): 502	Resolved

Description

Repository:

- `AB Chain`

Commit hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `consensus/clique/clique.go`

The function `Prepare` prepares a block header for the next block in the Clique proof-of-authority consensus mechanism. It manages voting on proposals, adjusts block difficulty, sets extra block data, and ensures that timestamps are valid.

However, the `signer` is accessed when calculating the block's difficulty with `CalcDifficulty(snap, c.signer)`. If the signer field is being modified concurrently by another goroutine, it may cause a race condition, which could result in unexpected behavior, crashes, or incorrect block difficulty calculations.

`consensus/clique/clique.go`

```
501     if number%c.config.Epoch != 0 {
502         c.lock.RLock()
503 
504         // Gather all the proposals that make sense voting on
505         addresses := make([]common.Address, 0, len(c.proposals))
506         for address, authorize := range c.proposals {
507             if snap.validVote(address, authorize) {
508                 addresses = append(addresses, address)
509             }
510         }
511         // If there's pending proposals, cast a vote on them
512         if len(addresses) > 0 {
513             header.Coinbase = addresses[rand.Intn(len(addresses))]
514             if c.proposals[header.Coinbase] {
515                 copy(header.Nonce[:], nonceAuthVote)
516             } else {
517                 copy(header.Nonce[:], nonceDropVote)
518             }
519         }
520         c.lock.RUnlock()
521     }
522     // Set the correct difficulty
523     header.Difficulty = CalcDifficulty(snap, c.signer)
```

Reference:

- `consensus/clique: fix race condition #24957`

Recommendation

Recommend to protect the `c.signer` access with the same mutex lock according to `consensus/clique: fix race condition #24957` .

Alleviation

[AB-Chain Team - 03/07/2025] :

The team heeded our advice and resolved the finding by updating the lock. The change is reflected in the commit `f8ada776c1e02c53d46266be88a9024458811037` .

GAS-01 | UNHANDLED ERRORS IN `gasprice`

Category	Severity	Location	Status
Volatile Code	Minor	eth/gasprice/gasprice.go: 84; eth/gasprice/gasprice.go (v1.9.18-newton): 84	Acknowledged

Description

Repository:

- `AB Chain`

Commits:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `eth/gasprice/gasprice.go`

The errors in the code snippet below have been obscured or unhandled, and if not addressed properly, they could lead to unforeseen circumstances.

`eth/gasprice/gasprice.go`

```
84  head, _ := gpo.backend.HeaderByNumber(ctx, rpc.LatestBlockNumber)
```

Recommendation

Recommend adding corresponding error handlings.

Alleviation

[AB-Chain Team - 03/03/2025] :
The team acknowledged this finding and will not update the codebase this time.

GAS-02 | VOLATILE CODE FOR INDEX IN `txPrices` SLICE

Category	Severity	Location	Status
Volatile Code, Logical Issue	Minor	eth/gasprice/gasprice.go: 145~146; eth/gasprice/gasprice.go (v1.9.18-newton): 145~146	Acknowledged

Description

Repository:

- `AB Chain`

Commit hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `eth/gasprice/gasprice.go`

In the `SuggestPrice` function, the gas price is retrieved using the Gas Price Oracle (GPO), which incorporates logic to store prices in a slice called `txPrices`. After sorting the slice, the function is designed to return the price corresponding to the specified GPO percentile. However, on **line 145**, the index calculation for the `txPrices` slice involves multiplying the index by the corresponding percentage, which introduces a critical flaw.

`eth/gasprice/gasprice.go`

```
143     if len(txPrices) > 0 {
144         sort.Sort(bigIntArray(txPrices))
145         price = txPrices[(len(txPrices)-1)*gpo.percentile/100]
```

This flawed implementation can result in incorrect index calculation, potentially causing the function to return the wrong price —most likely the first element in the `txPrices` slice, regardless of the intended percentile. This volatility in the code undermines the reliability of the gas price suggestion mechanism.

Recommendation

Recommend to revisit the implementation to correct the code of line 145 as below:

```
145    price = txPrices[(len(txPrices)-1)]*gpo.percentile/100
```

Alleviation

[AB-Chain Team - 03/10/2025] :

After carefully reviewing the gasprice code, the team think the logic is correct. It should use the index, otherwise the gasprice may not meet the price set by the node.

[CertiK - 03/11/2025] :

A separate ticket has been submitted to EF client team pending review, adjust the severity to minor for configure dependency and will update the status accordingly with the reply.

PER-01 | POTENTIAL DOS ATTACK VIA MALICIOUS P2P MESSAGE

Category	Severity	Location	Status
Volatile Code, Denial of Service	Minor	p2p/peer_error.go: 57; p2p/peer_error.go (v1.9.18-newton): 57	Resolved

Description

Repository:

- `AB Chain`

Commit Hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `p2p/peer.go`

A vulnerability exists in the node's logging mechanism when configured to use high verbosity logging. An attacker can exploit this by sending specially crafted peer-to-peer (p2p) messages, causing the node to crash. The issue arises from improper handling of the decoding process in the p2p message handler. Specifically, decoding the last message from the attacker node triggers a panic due to the lack of error handling in the following code snippet:

```
case msg.Code == discMsg:
    var reason [1]DiscReason
    // This is the last message. We don't need to discard or
    // check errors because, the connection will be closed after it.
    rlp.Decode(msg.Payload, &reason)
    return reason[0]
```

References

1. `GHSA-wjxw-gh3m-7pm5`
2. `CVE-2022-29177`

Recommendation

Recommend to update the DiscReason type to uint8 as upstream geth `p2p: define DiscReason as uint8`

Alleviation

[AB-Chain Team - 03/03/2025] :

The team heeded our advice and resolved the finding by updating the DiscReason type to uint8. The change is reflected in the commit `d2bd242136b8adca15ad01a35de7fd7b00f80bb5` .

SIG-01 CONCERNS ON INCONSISTENT SIGNATURE VALIDATION IN `checkSignature` FUNCTION

Category	Severity	Location	Status
Inconsistency	Informational	crypto/signature_r1.go: 75~77, 126~129; crypto/signature_r1.go (v1.9.18-newton): 75~77, 126~129	Acknowledged

Description

Repository:

- `AB Chain`

Commit Hash:

- `593e84644606a733d73e29358f289f8721a0214c`
- `abiot:593e84644606a733d73e29358f289f8721a0214c`

Files:

- `crypto/signature_r1.go`

The `checkSignature` function is responsible for validating the format of a given signature. The function checks if the last byte of the signature (`sig[64]`) is equal to or greater than 4, returning an error if this condition is met. The relevant code snippet is as follows:

```
func checkSignature(sig []byte) error {
    if len(sig) != 65 {
        return ErrInvalidSignatureLen
    }
    if sig[64] >= 4 {
        return ErrInvalidRecoveryID
    }
    return nil
}
```

However, according to the comment on the function `Sign`

> The produced signature is in the [R || S || V] format where V is 0 or 1

Similarly, inside the `ecRecovery2` , the `recId` is limited to [0, 3].

This discrepancy suggests that the check for sig[64] >= 4 might be incorrect, introducing what seems to be an "insanity check". We would like to confirm if it is designed as expected.

Recommendation

Recommend to restrict `sig[64]` to its target value.

Alleviation

[AB-Chain Team - 03/03/2025] :
The team stated that the function `checkSignature()` is based on http://github.com/ethereum/go-ethereum/blob/master/crypto/secp256k1/secp256.go#L174

APPENDIX | AB

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.