Audit of Ethermint

Crypto.com

11 February 2022
Version: 1.2

Presented by:
Kudelski Security Research Team
Kudelski Security – Nagravision SA

Corporate Headquarters
Kudelski Security – Nagravision SA
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

DOCUMENT PROPERTIES

Version:	1.2
File Name:	Report_ethermint
Publication Date:	11 February 2022
Confidentiality Level:	For public release
Document Owner:	Tommaso Gagliardoni
Document Recipient:	Federico Kunze Küllmer
Document Status:	Approved

TABLE OF CONTENTS

TABLE OF FIGURES

EXECUTIVE SUMMARY

Kudelski Security ("Kudelski", "we"), the cybersecurity division of the Kudelski Group, was engaged by Crypto.org ("the Client") to conduct an external security assessment in the form of a code audit of selected components of the Ethermint platform developed by the Client.

The assessment was conducted remotely the Kudelski Research Team. The audit took place from August 16, 2021 to August 27, 2021 and focused on the following objectives:

- To provide a professional opinion on the maturity, adequacy, and efficiency of the software solution in exam.
- To check compliance with existing standards.
- To identify potential security or interoperability issues and include improvement recommendations based on the result of our analysis.

This report summarizes the analysis performed and findings. It also contains detailed descriptions of the discovered vulnerabilities and recommendations for remediation.

1.1 Engagement Scope

The scope of the audit was a security review of the following code repositories and files:

• Crypto - https://github.com/tharsis/ethermint/tree/main/crypto

• Json RPC - https://github.com/tharsis/ethermint/tree/main/ethereum/rpc

• Ante handler - https://github.com/tharsis/ethermint/tree/main/app

• EVM state transition - specifically
https://github.com/tharsis/ethermint/blob/main/x/evm/spec/02_state.md and
https://github.com/tharsis/ethermint/blob/main/x/evm/keeper/state_transition.go

1.2 Engagement Analysis

The engagement consisted of a ramp-up phase where the necessary documentation about the technological standards and design of the solution in exam was acquired, followed by a manual inspection of the code provided by the Client and the drafting of this report.

As a result of our work, we identified **1 High**, **5 Medium**, **10 Low**, and **7 Informational** findings.

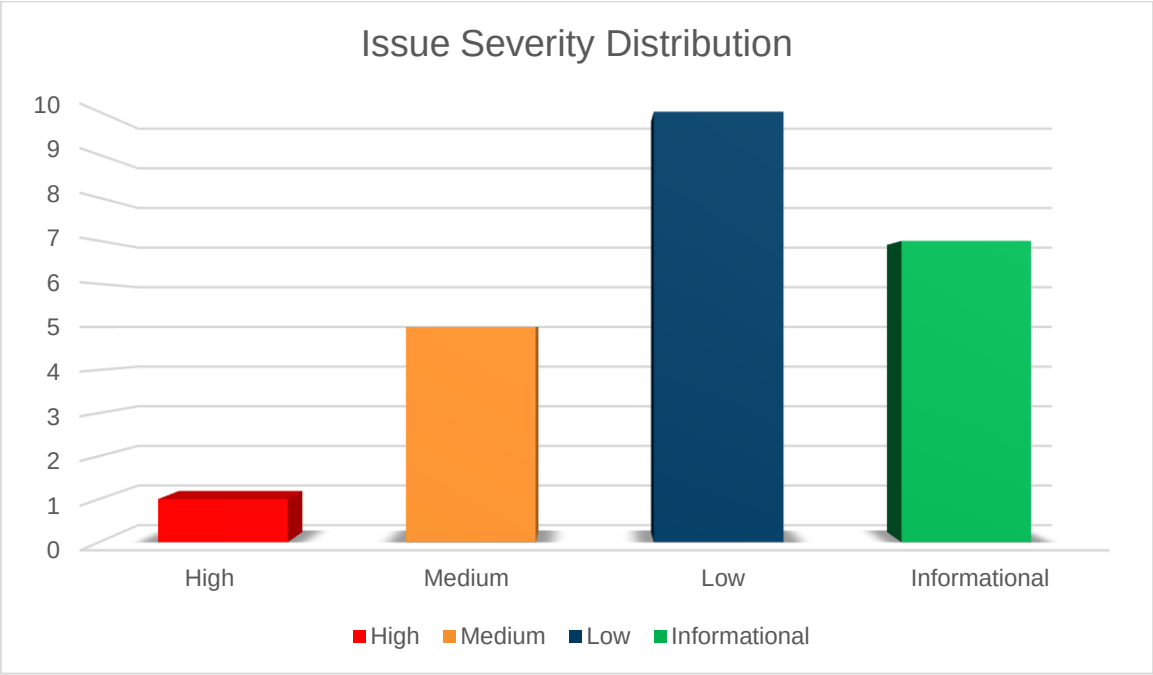

Figure 1 Issue Severity Distribution

1.3 Observations

Ethermint is a Proof-of-Stake blockchain based on Tendermint consensus and Cosmos SDK which aims at being compatible with Ethereum's EVM. The part of the code in scope for this audit is mainly responsible for key management and interfacing within Ethereum's EVM. Timing side-channel attacks were not considered in scope for this engagement.

In general, we found the implementation to be of high standard and we believe that all the identified vulnerabilities can be easily addressed. Moreover, we did not find evidence of any hidden backdoor or malicious intent in the code.

1.4 Issue Summary List

The following security issues were found:

ID	SEVERITY	FINDING	STATUS
KS-OFS-F-01	High	Websocket RPC server does not use TLS	Remediated
KS-OFS-F-02	Medium	`Address` function panics if point is not on the curve	Remediated
KS-OFS-F-03	Medium	Dependencies with security issues	Partially Remediated
KS-OFS-F-04	Medium	Publicly exposed APIs through RPC	Remediated
KS-OFS-F-05	Medium	`ExpandHome` restrictions can be bypassed	Remediated
KS-OFS-F-06	Medium	Filter creation via RPC-JSON could create OOM	Remediated
KS-OFS-F-07	Low	Cast of unsigned to signed integer in `GetHashFn`	Remediated
KS-OFS-F-08	Low	Cast of unsigned to signed integer in API	Open
KS-OFS-F-09	Low	`ToECDSA` function panics if scalar is malformed	Remediated
KS-OFS-F-10	Low	Loop bounds not enforced in `AnteHandle`	Open
KS-OFS-F-11	Low	Ambiguity in string format conversion	Open
KS-OFS-F-12	Low	Private key not reduced modulo curve order	Open
KS-OFS-F-13	Low	Use of deprecated methods	Open
KS-OFS-F-14	Low	Unhandled errors when closing files	Open
KS-OFS-F-15	Low	Lack of truncation detection	Partially Remediated
KS-OFS-F-16	Low	`NewAccount` fails if several calls are performed in sequence	Open

The following are non-security observations related to general design and optimization:

ID	SEVERITY	FINDING	STATUS
KS-OFS-O-01	**Informational**	Password parameter is never used in `SendTransaction` and `Sign` methods	**Informational**
KS-OFS-O-02	**Informational**	Size of secp256k1 public key	**Informational**
KS-OFS-O-03	**Informational**	Detecting need of hashing a message before signing	**Informational**
KS-OFS-O-04	**Informational**	Golang's ”panic, defer, recover” logic is sometimes replaced by extra upper layer for error handling	**Informational**
KS-OFS-O-05	**Informational**	Commented out test suites	**Informational**
KS-OFS-O-06	**Informational**	Unused or unassigned variables	**Informational**
KS-OFS-O-07	**Informational**	Packages being imported more than once	**Informational**

2. METHODOLOGY

For this engagement, Kudelski used a methodology that is described at high-level in this section. This is broken up into the following phases.

Figure 2 Methodology Flow

2.1 Kickoff

The project was kicked off when all of the sales activities had been concluded. We set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting we verified the scope of the engagement and discussed the project activities. It was an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there was an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

2.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps needed for gaining familiarity with the codebase and technological innovations utilized, such as:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for the languages used in the code
- Researching common flaws and recent technological advancements

2.3 Review

The review phase is where a majority of the work on the engagement was performed. In this phase we analyzed the project for flaws and issues that could impact the security posture. This included an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project

3. Assessment of the cryptographic primitives used
4. Compliance of the code with the provided technical documentation

The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This is a general and not comprehensive list, meant only to give an understanding of the issues we have been looking for.

Cryptography

We analyzed the cryptographic primitives and components as well as their implementation. We checked in particular:

- Matching of the proper cryptographic primitives to the desired cryptographic functionality needed
- Security level of cryptographic primitives and their respective parameters (key lengths, etc.)
- Safety of the randomness generation in general as well as in the case of failure
- Safety of key management
- Assessment of proper security definitions and compliance to use cases
- Checking for known vulnerabilities in the primitives used

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

2.4 Reporting

Kudelski delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project, which is also the general structure of the current final report.

The executive summary contains an overview of the engagement, including the number of findings as well as a statement about our general risk assessment of the project as a whole.

In the report we not only point out security issues identified but also informational findings for improvement categorized into several buckets:

- High
- Medium
- Low
- Informational

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

As we performed the audit, we also identified issues that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

2.5 Verify

After the preliminary findings have been delivered, we verified the fixes applied by the Client. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase was the current, final report with any mitigated findings noted.

2.6 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

Most of the audit was performed through a manual inspection of the code, but additional automated tools were used, to check code integrity and common programming issues.

Correct memory management is left to Golang and was therefore not in scope. Zeroization of secret values from memory is also not enforceable at a low level in a language such as Golang.

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in **Appendix C** of this document.

3. TECHNICAL DETAILS OF SECURITY FINDINGS

This section contains the technical details of our findings as well as recommendations for mitigation.

3.1 Websocket RPC server does not use TLS

Finding ID: KS-THA-F-01

Severity: High

Status: Remediated

Location: `ethereum/rpc/websockets.go @line 84`

Description and Impact Summary

The import raw method contains the password needed to decrypt the imported key. If TLS is not enforced, this sensitive value is exposed.

```
        err := http.ListenAndServe(s.wsAddr, ws)
```

Recommendation

Use 'http.ListenAndServeTLS' instead.

```
http.ListenAndServeTLS(s.wsAddr, certFile, keyFile, ws)
```

See https://golang.org/pkg/net/http/#ListenAndServeTLS for more information.

Status Details

This has been fixed in #600. If a valid TLS certificate and configuration is detected, then TLS connection to the Websocket is provided, otherwise it is left unencrypted.

3.2 `Address` function panics if point is not on the curve

Finding ID: KS-THA-F-02

Severity: Medium

Status: Remediated

Location: `crypto/ethsecp256k1/ethsecp256k1.go @line 140`

Description and Impact Summary

The function `Address()` panics if the input public key is a point not on the curve.

```
// Address returns the address of the ECDSA public key.
// The function will panic if the public key is invalid.
func (pubKey PubKey) Address() tmcrypto.Address {
    pubk, err := ethcrypto.DecompressPubkey(pubKey.Key)
    if err != nil {
        panic(err)
    }
```

We consider this undesirable behaviour because it can be easily exploited remotely.

Recommendation

We recommend using proper error handling rather than just panicking.

Status Details

This has been fixed in #535. Now the function will return an empty address if the public key is invalid rather than panicking.

3.3 Dependencies with security issues

Finding ID: KS-THA-F-03

Severity: Medium

Status: Partially Remediated

Description and Impact Summary

Ethermint relies on packages with known security issues.

```
pkg:golang/github.com/coreos/etcd@3.3.13
[CVE-2020-15114]
[CVE-2020-15136]
[CVE-2020-15115]

pkg:golang/github.com/dgrijalva/jwt-go@3.2.0
[CVE-2020-26160]
```

These packages are not used by the SDK, however they can be part of the dependency tree of other dependencies.

Recommendation

We recommend checking these dependencies and using patched versions, or removing them altogether where possible.

Status Details

This has been partially addressed:

- Added lint checker for versions
- Added additional CI check
- Bumped to Go 1.17

3.4 Publicly exposed APIs through RPC

Finding ID: KS-THA-F-04

Severity: Medium

Status: Remediated

Location: `ethereum/rpc/apis.go @line 86`

Description and Impact Summary

All the exposed APIs through RPC are public. This includes sensitive information such as key list and transactions.

```
rpc.API{
                    Namespace: PersonalNamespace,
                    Version:   apiVersion,
                    Service:   personal.NewAPI(ctx.Logger, clientCtx, evmBackend),
                    Public:    true,
```

Recommendation

We recommend restricting access to sensitive APIs.

Status Details

This has been fixed in #535.

3.5 `ExpandHome` restrictions can be bypassed

Finding ID: KS-THA-F-05

Severity: Medium

Status: Remediated

Location: `ethereum/rpc/namespaces/debug/utils.go @line 27`

Description and Impact Summary

In order to avoid expansions of type `~someuser/tmp`, a prefix in the path "~" is filtered to catch those cases. However, if this is case, the home directory is obtained from the system variable `$HOME`. An attacker that wants to bypass the expansion restriction only needs to set `$HOME` to `/tmp` (or to `/etc`, `/root`, etc.). If ethermintd is running with privileges, setting `$HOME` to sensitive parts of the system can be used to write without permission.

```
func ExpandHome(p string) string {
    if strings.HasPrefix(p, "~/") || strings.HasPrefix(p, "~\\") {
        home := os.Getenv("HOME")
        if home == "" {
            if usr, err := user.Current(); err == nil {
                home = usr.HomeDir
            }
        }
        if home != "" {
            p = home + p[1:]
        }
```

We consider this of Medium severity: privilege write operations are related to specific files of ethermintd – *unless* `writeProfile` (`ethereum/rpc/namespaces/debug/utils.go`) input files can be changed. At the moment, `writeProfile` calls use the following file names: `block, heap, mutex`.

Recommendation

We recommend enforcing sanitization of the full home path, and avoiding the use of `$HOME` if possible.

Status Details

This has been fixed in #535.

3.6 Filter creation via RPC-JSON could create OOM

Finding ID: KS-THA-F-06

Severity: Medium

Status: Remediated

Location: `ethereum/rpc/namespaces/eth/filters`

Description and Impact Summary

The RPC methods `eth_newPendingTransactionFilter`, `eth_newBlockFitler`, and `eth_newFilter` can be invoked at will creating a potentially large number of arbitrary filters, thereby saturating memory.

Recommendation

Enforce a limit in the number of filters that can be created to reduce the probability of attacking a node via the RPC-JSON interface by creating a large number of filters.

Status Details

This has been fixed in #535.

3.7 Cast of unsigned to signed integer in `GetHashFn`

Finding ID: KS-THA-F-07

Severity: Low

Status: Remediated

Location: `x/evm/keeper/state_transition.go @line 66`

Description and Impact Summary

The variable `height` is converted from signed to unsigned integer without check.

```
func (k Keeper) GetHashFn() vm.GetHashFunc {
    return func(height uint64) common.Hash {
        h := int64(height)
```

The risk is that a large enough value for `height` could be converted to a negative number.

Recommendation

Although probably difficult to exploit in practice, as a defense in depth we recommend checking that the cast range is correct.

Status Details

This has been fixed in #597.

3.8 Cast of unsigned to signed integer in API

Finding ID: KS-THA-F-08

Severity: Low

Status: Open

Location: `ethereum/rpc/namespaces/eth/api.go @line 575`

Description and Impact Summary

The variable `idx` is converted from unsigned to signed integer without check. The risk is that a large enough value for `idx` could be converted to a negative number.

Recommendation

Although probably difficult to exploit in practice, as a defense in depth we recommend checking that the cast range is correct.

Status Details

Waiting for feedback from Client.

3.9 `ToECDSA` function panics if scalar is malformed

Finding ID: KS-THA-F-09

Severity: Low

Status: Remediated

Location: `crypto/ethsecp256k1/ethsecp256k1.go @line 122`

Description and Impact Summary

The function converts a byte-array representation of a private key into a scalar, but panics if the representation is malformed.

```
func (privKey PrivKey) ToECDSA() *ecdsa.PrivateKey {
    key, err := ethcrypto.ToECDSA(privKey.Bytes())
    if err != nil {
        panic(err)
    }
    return key
}
```

We consider this less severe than KS-THA-F-02, as it would require an alteration of a private rather than public key.

Recommendation

We recommend using proper error handling rather than just panicking.

Status Details

This has been fixed in #535.

3.10 Loop bounds not enforced in `AnteHandle`

Finding ID: KS-THA-F-10

Severity: Low

Status: Open

Location: `app/ante/eth.go @lines 121, 199, 279, 375, 448, 489, 512`

Description and Impact Summary

`AnteHandle` performs a series of checks looping through all the messages embedded in a transaction. However, there is no loop bound in place. This might cause malicious transactions to slow down the processing or even cause an OOM condition.

```
    for i, msg := range tx.GetMsgs) {
        msgEthTx, ok := msg.(*evmtypes.MsgEthereumTx)
```

Recommendation

We recommend enforcing a (large) upper bound to the number of messages that can be processed within a single transaction.

Status Details

Waiting for feedback from Client.

3.11 Ambiguity in string format conversion

Finding ID: KS-THA-F-11

Severity: Low

Status: Open

Location: `crypto/hd/algorithm @line 62`

Description and Impact Summary

ParseDerivationPath parses strings starting with `0` to octal values, and strings starting with `0x` and `0b` to hexadecimal and binary values, respectively. This can easily cause collisions in the resulting address, which can be exploited e.g. with a JSON-RPC call:

```
~ curl -X POST --data
'{"jsonrpc":"2.0","method":"web3_sha3","params":["0x"],"id":1}' -H "Content-Type:
application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"result":"0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273
b7bfad8045d85a470"}
```

```
~ curl -X POST --data '{"jsonrpc":"2.0","method":"web3_sha3","params":[""],"id":1}'
-H "Content-Type: application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"result":"0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273
b7bfad8045d85a470"}
```

```
~ curl -X POST --data '{"jsonrpc":"2.0","method":"web3_sha3","params":[""],"id":1}'
-H "Content-Type: application/json" http://localhost:8545
```

```
~ curl -X POST --data
'{"jsonrpc":"2.0","method":"web3_sha3","params":["ab"],"id":1}' -H "Content-Type:
application/json" http://localhost:8545 {"jsonrpc":"2.0","id":1,"error":{"code":-
32602,"message":"invalid argument 0: json: cannot unmarshal hex string without 0x
prefix into Go value of type hexutil.Bytes"}}
```

Recommendation

We recommend making the parsing rules stricter in order to avoid this ambiguity.

Status Details

Waiting for feedback from Client.

3.12 Private key not reduced modulo curve order

Finding ID: KS-THA-F-12

Severity: Low

Status: Open

Location: `crypto/hd/algorithm @line 98`

Description and Impact Summary

The `Generate` function does not reduce the private key modulus the order of the curve. This may be problematic when two private keys are compared with the `Equals` function.

```
func (s ethSecp256k1Algo) Generate() hd.GenerateFn {
    return func(bz []byte) cryptotypes.PrivKey {
        var bzArr = make([]byte, ethsecp256k1.PrivKeySize)
        copy(bzArr, bz)

        return &ethsecp256k1.PrivKey{Key: bzArr}
    }
}
```

Recommendation

We recommend reducing the key modulo order of the curve before returning.

Status Details

Waiting for feedback from Client.

3.13 Use of deprecated methods

Finding ID: KS-THA-F-13

Severity: Low

Status: Open

Location: various

Description and Impact Summary

We report the use of deprecated methods across the codebase:

```
ethereum/rpc/namespaces/personal/api.go:42:14: cfg.GetFullFundraiserPath is deprecated: This method is supported for backward compatibility only and will be removed in a future release. Use GetFullBIP44Path instead. (SA1019)

x/evm/types/query.pb.gw.go:16:2: package github.com/golang/protobuf/descriptor is deprecated: See the "google.golang.org/protobuf/reflect/protoreflect" package for how to obtain an EnumDescriptor or MessageDescriptor in order to programatically interact with the protobuf type system. (SA1019)

x/evm/types/query.pb.gw.go:17:2: package github.com/golang/protobuf/proto is deprecated: Use the "google.golang.org/protobuf/proto" package instead. (SA1019)

x/evm/types/query.pb.gw.go:32:9: descriptor.ForMessage is deprecated: Not all concrete message types satisfy the Message interface. Use MessageDescriptorProto instead. If possible, the calling code should be rewritten to use protobuf reflection instead. See package "google.golang.org/protobuf/reflect/protoreflect" for details. (SA1019)
```

Recommendation

We recommend avoiding the use of deprecated methods.

Status Details

Waiting for feedback from Client.

3.14 Unhandled errors when closing files

Finding ID: KS-THA-F-14

Severity: Low

Status: Open

Location: `ethereum/rpc/namespaces/debug/trace.go @line 65`
`ethereum/rpc/namespaces/debug/trace.go @line 44`
`ethereum/rpc/namespaces/debug/internal.go @line 158`
`ethereum/rpc/namespaces/debug/internal.go @line 134`
`ethereum/rpc/namespaces/debug/utils.go @line 50`
`ethereum/rpc/websockets.go @line 250`

Description and Impact Summary

Errors returned when closing files are not properly handled. This can cause panics.

Moreover, deferring unsafe method "Close" on type "*os.File" can create panic.

```
// writeProfile writes the data to a file
func writeProfile(name, file string, log log.Logger) error {
    p := pprof.Lookup(name)
    log.Info("Writing profile records", "count", p.Count(), "type", name, "dum
p", file)
    f, err := os.Create(ExpandHome(file))
    if err != nil {
        return err
    }
    defer f.Close()

    return p.WriteTo(f, 0)
}
```

Recommendation

We recommend always handling errors when closing files and not deferring closure. See also https://www.joeshaw.org/dont-defer-close-on-writable-files/ .

Status Details

Waiting for feedback from Client.

3.15 Lack of truncation detection

Finding ID: KS-THA-F-15

Severity: Low

Status: Partially Remediated

Location: `ethereum/rpc/types/block.go @line 28`
`ethereum/rpc/namespaces/eth/filters/utils.go @line 16`
`ethereum/rpc/namespaces/miner/api.go @line 186`
`ethereum/rpc/types.go @line 117, 121`

Description and Impact Summary

When a `Big.Int` variable whose value is greater than `math.MaxInt64` is converted to `Int64` a truncation happens. In this case, the value of the resulting variable will be always set to `math.MaxInt64` without raising any error and without being detected. For example, in the following code snippet.

```
// NewBlockNumber creates a new BlockNumber instance.
func NewBlockNumber(n *big.Int) BlockNumber {
    return BlockNumber(n.Int64())
}
```

Recommendation

We recommend checking for overflows when truncating.

Status Details

Fixed in `block.go` and `types.go`.

3.16 `NewAccount` fails if several calls are performed in sequence

Finding ID: KS-THA-F-16

Severity: Low

Status: Open

Location: `ethereum/rpc/namespaces/personal/api.go @line 122`

Description and Impact Summary

Calling `NewAccount` many times in sequence fails.

```
~ curl -X POST --data
'{"jsonrpc":"2.0","method":"personal_newAccount","params":["This is the
passphrase"],"id":1}' -H "Content-Type: application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"result":"0xb453a4f30c92524a6b88b775e9a25a9f1a1fe1fe"}

~ curl -X POST --data
'{"jsonrpc":"2.0","method":"personal_newAccount","params":["This is the
passphrase"],"id":1}' -H "Content-Type: application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,"message":"public key already exists
in keybase"}}

~ curl -X POST --data
'{"jsonrpc":"2.0","method":"personal_newAccount","params":["This is the
passphrase"],"id":1}' -H "Content-Type: application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,"message":"public key already exists
in keybase"}}

~ curl -X POST --data
'{"jsonrpc":"2.0","method":"personal_newAccount","params":["This is the
passphrase"],"id":1}' -H "Content-Type: application/json" http://localhost:8545
{"jsonrpc":"2.0","id":1,"error":{"code":-32000,"message":"public key already exists
in keybase"}}

~ curl -X POST --data
'{"jsonrpc":"2.0","method":"personal_newAccount","params":["This is the
passphrase"],"id":1}' -H "Content-Type: application/json" http://localhost:8545
^[[A{"jsonrpc":"2.0","id":1,"error":{"code":-32000,"message":"public key already
exists in keybase"}}
```

We consider this hardly exploitable in practice, but it might be an indication of entropy pool exhaustion.

Recommendation

We recommend checking that `NewAccount` does not deplete the entropy pool. Always prefer using `/dev/urandom` rather than `/dev/random`.

Status Details

Waiting for feedback from Client.

4. OTHER OBSERVATIONS

This section contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

4.1 Password parameter is never used in `SendTransaction` and `Sign` methods

Observation ID: KS-THA-O-01

Location: `ethereum/rpc/namespaces/personal/api.go @line 167, 153`

Description and Impact Summary

The password field is part of the arguments of the API, however it is never used. It this API is public and not TLS protected, this exposes a sensitive value that is not needed.

Recommendation

We recommend removing sensitive fields from the RPC API interface if not needed.

4.2 Size of secp256k1 public key

Observation ID: KS-THA-O-02

Location: `crypto/ethsecp256k1/ethsecp256k1.go`

Description and Impact Summary

The size of the public key is 33 bytes, but only 32 are needed.

Recommendation

We recommend documenting the choice of format as it is done in the Tendermint code.

4.3 Detecting need of hashing a message before signing

Observation ID: KS-THA-O-03

Location: `crypto/ethsecp256k1/ethsecp256k1.go @line 110`

Description and Impact Summary

The function `Sign` checks if the input equals the length of the keccack256 output in order to determine if the message needs to be hashed or not prior to signing. This is not robust, as it can easily mistake unhashed messages for hashed ones, or input coming from different hash functions with similar bitsize, leading to interoperability issues.

```
func (privKey PrivKey) Sign(digestBz []byte) ([]byte, error) {
    if len(digestBz) != ethcrypto.DigestLength {
        digestBz = ethcrypto.Keccak256Hash(digestBz).Bytes()
    }

    return ethcrypto.Sign(digestBz, privKey.ToECDSA())
}
```

Recommendation

We recommend either assuming that the input is always hashed, as it is done in other libraries, or add a parameter to decide whether hashing is needed or not.

4.4 Golang's ”panic, defer, recover” logic is sometimes replaced by extra upper layer for error handling

Observation ID: KS-THA-O-04

Location: `various`

Description and Impact Summary

Functions such as `try` in `websockets.go` and `Recover` (`x/evm/handler.go`) are used in Ethermint to replace sometimes the “panic, defer, recover” strategy of Go. They are used through the code in a not systematic manner (that is, still defer and `recover()` calls are used in some parts of the code base).

Recommendation

We recommend sticking to the simpler default error handling strategy in Go.

4.5 Commented out test suites

Observation ID: KS-THA-O-05

Location: `ethereum/rpc/pubsub/pubsub_test.go`
`app/simulation_test.go`

Description and Impact Summary

These test suites are commented out.

Recommendation

We recommend implementing these test suites correctly.

4.6 Unused or unassigned variables

Observation ID: KS-THA-O-06

Location: `app/export.go @line 174`

`x/evm/keeper/state_transition_test.go @line 52`

Description and Impact Summary

The variables `counter` and `validator` are never used.

Recommendation

We recommend removing unused variables.

Notes

This has been fixed in #680.

4.7 Packages being imported more than once

Observation ID: KS-THA-O-07

Location: `various`

Description and Impact Summary

Some packages are imported more than once throughout the codebase.

```
x/evm/keeper/grpc_query_test.go:12:2: package "github.com/ethereum/go-ethereum/common" is being imported more than once (ST1019)
x/evm/keeper/grpc_query_test.go:13:2: other import of "github.com/ethereum/go-ethereum/common"
x/evm/keeper/grpc_query_test.go:16:2: package "github.com/ethereum/go-ethereum/crypto" is being imported more than once (ST1019)
x/evm/keeper/grpc_query_test.go:17:2: other import of "github.com/ethereum/go-ethereum/crypto"
x/evm/keeper/keeper_test.go:27:2: package "github.com/ethereum/go-ethereum/common" is being imported more than once (ST1019)
x/evm/keeper/keeper_test.go:28:2: other import of "github.com/ethereum/go-ethereum/common"
x/evm/keeper/keeper_test.go:90:2: this value of err is never used (SA4006)
x/evm/keeper/statedb_test.go:18:2: package "github.com/tharsis/ethermint/x/evm/types" is being imported more than once (ST1019)
x/evm/keeper/statedb_test.go:19:2: other import of "github.com/tharsis/ethermint/x/evm/types"
x/evm/types/dynamic_fee_tx.go:14:6: func newDynamicFeeTx is unused (U1000)
x/evm/types/msg_test.go:14:2: package "github.com/ethereum/go-ethereum/common" is being imported more than once (ST1019)
x/evm/types/msg_test.go:15:2: other import of "github.com/ethereum/go-ethereum/common"
x/evm/types/msg_test.go:16:2: package "github.com/ethereum/go-ethereum/core/types" is being imported more than once (ST1019)
x/evm/types/msg_test.go:17:2: other import of "github.com/ethereum/go-ethereum/core/types"
x/evm/types/utils_test.go:19:2: package "github.com/ethereum/go-ethereum/common" is being imported more than once (ST1019)
x/evm/types/utils_test.go:20:2: other import of "github.com/ethereum/go-ethereum/common"
```

Recommendation

We recommend checking if a package has been already imported and only importing it if otherwise.

APPENDIX A: ABOUT KUDELSKI SECURITY

Kudelski Security is an innovative, independent Swiss provider of tailored cyber and media security solutions to enterprises and public sector institutions. Our team of security experts delivers end-to-end consulting, technology, managed services, and threat intelligence to help organizations build and run successful security programs. Our global reach and cyber solutions focus is reinforced by key international partnerships.

Kudelski Security is a division of Kudelski Group. For more information, please visit https://www.kudelskisecurity.com.

Kudelski Security

Route de Genève, 22-24

1033 Cheseaux-sur-Lausanne

Switzerland

Kudelski Security

5090 North 40th Street

Suite 450

Phoenix, Arizona 85018

APPENDIX B: DOCUMENT HISTORY

VERSION	STATUS	DATE	AUTHOR	COMMENTS
0.1	Draft	1 September 2021	Tommaso Gagliardoni	Initial draft
0.2	Draft	19 November 2021	Tommaso Gagliardoni	Draft updated with fixes
1.0	Final	26 November 2021	Tommaso Gagliardoni	Final version
1.1	Final	30 November 2021	Tommaso Gagliardoni	Fixed client name
1.2	Final	11 February 2022	Nathan Hamiel	For public release

REVIEWER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Security Research	1 September 2021	0.1	
Nathan Hamiel	Head of Security Research	19 November 2021	0.2	
Nathan Hamiel	Head of Security Research	29 November 2021	1.0	
Nathan Hamiel	Head of Security Research	30 November 2021	1.1	
Nathan Hamiel	Head of Security Research	11 February 2022	1.2	

APPROVER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Security Research	1 September 2021	0.1	
Nathan Hamiel	Head of Security Research	19 November 2021	0.2	
Nathan Hamiel	Head of Security Research	29 November 2021	1.0	
Nathan Hamiel	Head of Security Research	30 November 2021	1.1	

APPROVER	POSITION	DATE	VERSION	COMMENTS
Nathan Hamiel	Head of Security Research	11 February 2022	1.2	

APPENDIX C: SEVERITY RATING DEFINITIONS

Kudelski Security uses a custom approach when determining criticality of identified issues. This is meant to be simple and fast, providing customers with a quick at a glance view of the risk an issue poses to the system. As with anything risk related, these findings are situational. We consider multiple factors when assigning a severity level to an identified vulnerability. A few of these include:

- Impact of exploitation
- Ease of exploitation
- Likelihood of attack
- Exposure of attack surface
- Number of instances of identified vulnerability
- Availability of tools and exploits

SEVERITY	DEFINITION
High	The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
Medium	The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
Low	The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
Informational	Informational findings are best practice steps that can be used to harden the application and improve processes.