# Cross-Chain Cryptography Implementation Review

## Onchain

August 7, 2020 – Version 0.95


Prepared for
Walter Yang

Prepared by
Eric Schorn

# Executive Summary

## Synopsis

Starting in late July 2020, OnChain engaged NCC Group's Cryptography Services team to conduct a cryptography implementation review of a variety of cross-chain connectivity components written in Golang. These components included code from both the Ontology and Poly Network repositories as described below. The review was open ended but was time boxed to four person-days of effort. All source code was in scope.

## Scope

NCC Group's evaluation included the following components:

- Ontology, cross-chain – commit `f3cedab`[1] of https://github.com/ontio/ontology
  - smartcontract/service/native/cross_chain/cross_chain_manager/*
  - smartcontract/service/native/cross_chain/header_sync/*
  - smartcontract/service/native/cross_chain/common/*
- Cosmos poly – commit `f917a9a`[2] of https://github.com/polynetwork/cosmos-poly-module
  - btcx/internal/keeper/keeper.go
  - ccm/internal/keeper/keeper.go
  - ft/internal/keeper/ft_crossed_indenpently.go
  - ft/internal/keeper/keeper.go
  - headersync/internal/keeper/keeper.go
  - lockproxy/internal/keeper/keeper.go

## Limitations

NCC Group was able to achieve robust coverage of all the above listed components. Note that these components interact with a much larger system that was out of scope. Nonetheless, immediate aspects of the larger system along with adjacent supporting source code was considered where reasonably possible.

## Key Findings

The code demonstrated a number of excellent design and implementation choices. These include a careful focus on input/data validation, verbose error reporting, and consistency between related functionality across separate source files. However, the review did uncover a small number of issues including:

- One instance of a missing validation check prior to a uint64 → uint32 conversion that may silently lose data.
- Concerns relating to string UTF-8 character encoding and Unicode normalization that may affect `denom` uniqueness.
- Several instances of functions detecting an error condition and returning `'value, error'` where the `value` is legal data rather than `nil`. Returning legal data does not force calling code to check for errors, and may allow it to silently continue with erroneous data. Separately, several instances of an unchecked `nil` return value are present.
- An outdated Golang toolchain specification for Ontology code that does not incorporate the latest security revisions, does not utilize recent improvements in dependency version management and the standard library, and does not align with the toolchain specified by the Poly Network code.
- A number of informational code style improvements were identified, where unused parameters may indicate incomplete refactoring.

## Strategic Recommendations

As the code moves forward towards production and/or incorporating new functionality, NCC Group believes it would be most valuable to prioritize the following strategic efforts:

- Continued focus on validation with additional checks on expected data schema wherever possible (e.g. specific length of address, upper/lower scalar bounds, and empty strings).
- Ensure all toolchains and dependencies are aligned and the most recent recommended for production deployment.
- Enhance testing to include a broader range of input values and scenarios, and increased coverage of unhappy paths.

[1] https://github.com/ontio/ontology/tree/f3cedabc969f485301e501d96e972c25f9de32c1
[2] https://github.com/polynetwork/cosmos-poly-module/tree/f917a9a3331f34a7d9ee86bdbd42a8337a3d2c18

# Dashboard

## Target Metadata

| | |
|---|---|
| **Name** | Cross-chain components |
| **Type** | Native code |
| **Platforms** | Golang |

## Engagement Data

| | |
|---|---|
| **Type** | Manual source code review |
| **Dates** | 2020-07-28 to 2020-08-06 |
| **Consultants** | 1 |
| **Level of Effort** | Four person-days |

## Targets

| | |
|---|---|
| https://github.com/ontio/ontology | Ontology, cross-chain – commit `f3cedab` |
| https://github.com/polynetwork/cosmos-poly-module | Poly Network, cosmos – commit `f917a9a` |

## Finding Breakdown

| | | |
|---|---|---|
| Critical issues | 0 | |
| High issues | 0 | |
| Medium issues | 2 | Medium, Medium |
| Low issues | 4 | Low, Low, Low, Low |
| Informational issues | 2 | Informational, Informational |
| **Total issues** | **8** | |

## Category Breakdown

| | | |
|---|---|---|
| Data Validation | 3 | Medium, Low, Low |
| Error Reporting | 3 | Medium, Low, Informational |
| Other | 1 | Informational |
| Patching | 1 | Low |

## Component Breakdown

| | | |
|---|---|---|
| Ontology | 4 | Low, Low, Low, Informational |
| Poly Network | 4 | Medium, Medium, Low, Informational |

## Key

Critical High Medium Low Informational

# Table of Findings

For each finding, NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. For an explanation of NCC Group's risk rating and finding categorization, see Appendix B on page 19.

## Ontology

| Title | ID | Risk |
| --- | --- | --- |
| Missing MaxUint32 Validation Check | 001 | Low |
| Functions Return Legal Values On Error Condition | 002 | Low |
| Outdated Compiler Specification | 004 | Low |
| Hardcoded Nil Error Return | 003 | Informational |

## Poly Network

| Title | ID | Risk |
| --- | --- | --- |
| Missing MaxUint64 Validation Check | 007 | Medium |
| Unchecked `nil` Return Condition | 008 | Medium |
| Potential for String `Denom` Miscomparisons | 006 | Low |
| Golang Code Improvements | 005 | Informational |

**Finding** **Missing MaxUint32 Validation Check**

**Risk** **Low** Impact: Medium, Exploitability: Low

**Identifier** NCC-ONCH008-001

**Category** Data Validation

**Component** Ontology

**Location** `ontology/smartcontract/service/native/cross_chain/cross_chain_manager/param.go`

**Impact** Information on chain height may be silently lost on a uint64 → uint32 conversion that may cause subsequent calculations to diverge.

**Description** The `Deserialization()` function for `ProcessCrossChainTxParam` structs in `param.go`[3] as shown below neglects to check the value of the uint64 `height` deserialized on line 92 prior to storing it as a uint32 on line 106. As a result, information may be silently lost.

```
83  func (this *ProcessCrossChainTxParam) Deserialization(source
 ➜      *common.ZeroCopySource) error {
84      address, err := utils.DecodeAddress(source)
85      if err != nil {
86          return fmt.Errorf("ProcessCrossChainTxParam deserialize addres...", err)
87      }
88      fromChainID, err := utils.DecodeVarUint(source)
89      if err != nil {
90          return fmt.Errorf("ProcessCrossChainTxParam deserialize fromCh...", err)
91      }
92      height, err := utils.DecodeVarUint(source)
93      if err != nil {
94          return fmt.Errorf("ProcessCrossChainTxParam deserialize heigh...", err)
95      }
96      proof, err := utils.DecodeString(source)
97      if err != nil {
98          return fmt.Errorf("ProcessCrossChainTxParam deserialize proof...", err)
99      }
100     header, err := utils.DecodeVarBytes(source)
101     if err != nil {
102         return fmt.Errorf("ProcessCrossChainTxParam deserialize head...", err)
103     }
104     this.Address = address
105     this.FromChainID = fromChainID
106     this.Height = uint32(height)
107 ...
```

Note that `../header_sync/states.go` performs this '`height > math.MaxUint32`' check on line 123.

**Recommendation** Validate that the value of `height` is less than `MaxUint32` prior to storing it as a narrower value.

[3] https://github.com/ontio/ontology/blob/f3cedabc969f485301e501d96e972c25f9de32c1/smartcontract/service/native/cross_chain/cross_chain_manager/param.go

**Finding** **Functions Return Legal Values On Error Condition**

**Risk** **Low** Impact: Low, Exploitability: Low

**Identifier** NCC-ONCH008-002

**Category** Error Reporting

**Component** Ontology

**Location**

- `ontology/smartcontract/service/native/cross_chain/cross_chain_manager/cross_chain.go`
- `ontology/smartcontract/service/native/cross_chain/header_sync/header_sync.go`
- `ontology/smartcontract/service/native/utils/serialization.go`

**Impact** A function that returns a potentially legal value alongside a detected error condition does not force the calling code to immediately check for returned errors. If the calling code neglects to check for errors, it will silently continue processing with erroneous data.

**Description** The `cross_chain.go`[4] and `header_sync.go` source files contain several instances of a function returning '`value, error`' (with both a legal value and error indicator) as shown below on lines 77, 83, 90, and 93. If the calling code neglects to check errors, then it may silently proceed despite a detected error condition. For this reason, best practices suggest[5,6] a return condition of '`nil, error`'. The vast majority of the Ontology code follows these best practices, with these instances being an exception.

Upon closer inspection, it appears several instances of this pattern consider the first return value to be equivalent to a success indicator, which is thus redundant with the second return error indicator. These instances can be simplified.

```
74 func ProcessCrossChainTx(native *native.NativeService) ([]byte, error) {
75     params := new(ProcessCrossChainTxParam)
76     if err := params.Deserialization(common.New...e(native.Input)); err != nil {
77         return utils.BYTE_FALSE, fmt.Errorf("ProcessCrossChainTx, contract pa..."
78     }
79 
80     //get block header
81     header, err := header_sync.GetHeaderByHeight(native, params.FromChainID...)
82     if err != nil {
83         return utils.BYTE_FALSE, fmt.Errorf("ProcessCrossChainTx, %d, %d"...)
84     }
85 
86     if header == nil {
87         header2 := new(ccom.Header)
88         err := header2.Deserialization(common.NewZeroCopySource(params.Header))
89         if err != nil {
90             return utils.BYTE_FALSE, fmt.Errorf("ProcessCrossChainTx, deseri..."
91         }
92         if err := header_sync.ProcessHeader(native, header2, ...); err != nil {
93         return utils.BYTE_FALSE, fmt.Errorf("ProcessCross..., error: %s", err)
94         }
```

[4] https://github.com/ontio/ontology/blob/f3cedabc969f485301e501d96e972c25f9de32c1/smartcontract/service/native/cross_chain/cross_chain_manager/cross_chain.go
[5] https://blog.golang.org/error-handling-and-go
[6] https://cwe.mitre.org/data/definitions/253.html

Note that primitive return values cannot be replaced with `nil`; only values returned by reference can be replaced.

The `DecodeAddress()` function in `serialization.go` also returns legal 'zeroed structures' upon an error condition. This should be reviewed.

**Recommendation** Wherever possible, return a nil value alongside the error condition. For example, line 77 should read

`return nil, fmt.Errorf("ProcessCrossChainTx, contract... error: %v", err)`.

| | |
|---|---|
| **Finding** | **Outdated Compiler Specification** |
| **Risk** | **Low** Impact: Low, Exploitability: Low |
| **Identifier** | NCC-ONCH008-004 |
| **Category** | Patching |
| **Component** | Ontology |
| **Location** | `ontology/go.mod` |
| **Impact** | An outdated toolchain and/or dependencies will increase the risk of exposure to known vulnerabilities that were subsequently remediated. |

**Description**

As excerpted below, the project `go.mod` file specifies Go version 1.12. Note that Golang is on a six-month release[7] cycle where version 1.13 was released on 3 September 2019 and version 1.14 was released on 25 February 2020. These newer versions incorporate a significant number of security improvements as well as relevant changes to the standard library, TLS configurations and dependency management.

```
module github.com/ontio/ontology

go 1.12

require (
	github.com/JohnCGriffin/overflow v0.0.0-20170615021017-4d914c927216
	github.com/Workiva/go-datastructures v1.0.50 // indirect
...
```

This wasn't investigated further as it is marginally outside of the project scope. However, note that the in-scope Poly Network material is using the current version (1.14) of Golang that does not match the Ontology version.

**Recommendation**

Update the compiler specification to the latest version recommended for production deployment. Ensure all dependencies are also fully updated.

Add a gating milestone to the development process that involves ensuring the compiler specification is updated and reviewing all dependencies for outdated or vulnerable versions.

[7] https://golang.org/doc/devel/release.html

**Finding** Hardcoded Nil Error Return

**Risk** **Informational** Impact: None, Exploitability: None

**Identifier** NCC-ONCH008-003

**Category** Error Reporting

**Component** Ontology

**Location** `ontology/smartcontract/service/native/cross_chain/cross_chain_manager/utils.go`

**Impact** Calling code may expect (or rely on) the function to detect a potential error condition and return an error indicator, but the function cannot do this, so the check may be missed.

**Description** The function signature for the `putRequest()` function[8] shown below suggests it can detect and return an error, but it cannot do this – it will always return `nil`.

```
func putRequest(native *native.NativeService, crossChainIDBytes, chainIDBytes,
➜ request []byte) error {
    contract := utils.CrossChainContractAddress
    utils.PutBytes(native, utils.ConcatKey(contract, []byte(REQUEST),
     ➜ chainIDBytes, crossChainIDBytes), request)
    return nil
}
```

**Recommendation** Either implement an error check, such as retrieving the key after Put, or adapt the function signature to return nothing.

[8] https://github.com/ontio/ontology/blob/f3cedabc969f485301e501d96e972c25f9de32c1/smartcontract/service/native/cross_chain/cross_chain_manager/utils.go

| | |
|---|---|
| **Finding** | **Missing MaxUint64 Validation Check** |
| **Risk** | **Medium** Impact: High, Exploitability: Low |
| **Identifier** | NCC-ONCH008-007 |
| **Category** | Data Validation |
| **Component** | Poly Network |
| **Location** | `btcx/internal/keeper/keeper.go` |
| **Impact** | Information on lock amounts may be silently lost on a BigInt → Uint64 conversion that may cause subsequent calculations to diverge. |

**Description**

The `Lock()` function excerpted below is intended to perform locking of BTC value. The function receives an `amount` of type `sdk.Int` on line 118 which is effectively a 256-bit BigInt. However, when constructing `ToBTCArgs` and `BTCArgs` on lines 135 and 144 respectively, the `amount` is converted to a `Uint64`. If the original value is too large to fit in the destination, the result is undefined.

```
118 func (k Keeper) Lock(ctx sdk.Context, fromAddr sdk.AccAddress, sourceAssetDenom
    ➜ string, toChainId uint64, toAddr []byte, amount sdk.Int) error {
119     // transfer back to btc
120     store := ctx.KVStore(k.storeKey)
121     toAssetHash := store.Get(GetBindAssetHashKey([]byte(sourceAssetDenom)...
122     if toAssetHash == nil {
123         return types.ErrLock(fmt.Sprintf("Invoke Lock of `btcx` module for de..."
124     }
125     sink := polycommon.NewZeroCopySink(nil)
126     // construct args bytes
127     if toChainId == types.BtcChainId {
128         creator := k.ccmKeeper.GetDenomCreator(ctx, sourceAssetDenom)
129         if creator.Empty() {
130             return types.ErrLock(fmt.Sprintf("Creator of denom: %s is Empty"...
131         }
132         redeemScript := store.Get(GetScriptHashToRedeemScript(store.Get(GetCre...
133         toBtcArgs := types.ToBTCArgs{
134             ToBtcAddress: toAddr,
135             Amount:       amount.BigInt().Uint64(),
136             RedeemScript: redeemScript,
137         }
138         if err := toBtcArgs.Serialization(sink); err != nil {
139             return types.ErrLock(fmt.Sprintf("ToBTCArgs Serialization Error..."
140         }
141     } else {
142         args := types.BTCArgs{
143             ToBtcAddress: toAddr,
144             Amount:       amount.BigInt().Uint64(),
145         }
```

For reference, the `TxArgs` struct in `lockproxy/internal/keeper/keeper.go` uses a consistent BigInt for `amount` on lines 182 and 203.

**Recommendation**

Validate that the value of `amount` is less than `MaxUint64` prior to storing it in the narrower value.

| | |
|---|---|
| **Finding** | **Unchecked `nil` Return Condition** |
| **Risk** | **Medium** Impact: Medium, Exploitability: Medium |
| **Identifier** | NCC-ONCH008-008 |
| **Category** | Error Reporting |
| **Component** | Poly Network |
| **Location** | • `btcx/internal/keeper/keeper.go`<br>• `ccm/internal/keeper/keeper.go`<br>• `ft/internal/keeper/ft_crossed_indenpently.go`<br>• `lockproxy/internal/keeper/keeper.go` |
| **Impact** | A `KVStore` Get() operation that returns nil may represent an error condition that should be reported rather than allowed to progress into downstream logic. An untrusted key able to cause a downstream panic would be a denial of service vulnerability. |

**Description**

In the `Lock()` function of `btcx/internal/keeper/keeper.go`[9] shown below, the `store.Get()` operation may return `nil`. This condition is checked on line 122 and an error raised as appropriate. This is a positive example.

```
118 func (k Keeper) Lock(ctx sdk.Context, fromAddr sdk.AccAddress, sourceAssetDenom
     ➜ string, toChainId uint64, toAddr []byte, amount sdk.Int) error {
119     // transfer back to btc
120     store := ctx.KVStore(k.storeKey)
121     toAssetHash := store.Get(GetBindAssetHashKey([]byte(sourceAssetDenom),
         ➜ toChainId))
122     if toAssetHash == nil {
123         return types.ErrLock(fmt.Sprintf("Invoke Lock of `btcx` module for de..."
124     }
125     sink := polycommon.NewZeroCopySink(nil)
126     // construct args bytes
127     if toChainId == types.BtcChainId {
128         creator := k.ccmKeeper.GetDenomCreator(ctx, sourceAssetDenom)
129         if creator.Empty() {
130             return types.ErrLock(fmt.Sprintf("Creator of denom: %s is Empty..."
131         }
132         redeemScript := store.Get(GetScriptHashToRedeemScript(store.Get(
             ➜ GetCreatorDenomToScriptHashKey(creator, sourceAssetDenom))))
133         toBtcArgs := types.ToBTCArgs{
134             ToBtcAddress: toAddr,
135             Amount:       amount.BigInt().Uint64(),
136             RedeemScript: redeemScript,
```

However, the two `store.Get()` operations highlighted above on line 132 are not checked against `nil`. A `nil` returned by the inner call on the right may cause a panic when used in the outer call on the left. Alternatively, if a `nil` is returned by the outer call on the left, it will be placed into the `ToBTCArgs` struct on line 136 which may cause issues in downstream logic.

The Poly Network code contains several concerning instances of this pattern, including:

- `btcx/internal/keeper/keeper.go` lines 132, 180, 215, 217, 226
- `ccm/internal/keeper/keeper.go` line 101; positive examples on lines 260 and 273

[9] https://github.com/polynetwork/cosmos-poly-module/blob/f917a9a3331f34a7d9ee86bdbd42a8337a3d2c18/btcx/internal/keeper/keeper.go

- `ft/internal/keeper/ft_crossed_indenpently.go` lines 54 and 161; positive examples on lines 84 and 122
- `lockproxy/internal/keeper/jeeper.go` lines 101, 154, 184, 196

Some of the above instances may be benign depending upon the expectations of downstream logic. Alternatively, if any untrusted input from across a trust boundary were able to cause a panic, this would result in a denial of service vulnerability.

**Recommendation** Review **all** instances of `store.Get()` to determine if a `nil` return value will cause downstream issues. The defense-in-depth principle suggests being cautious and detecting `nil` whenever possible.

**Finding** **Potential for String `Denom` Miscomparisons**

**Risk** **Low** Impact: Medium, Exploitability: Low

**Identifier** NCC-ONCH008-006

**Category** Data Validation

**Component** Poly Network

**Location**
- `btcx/internal/keeper/keeper.go`
- `ccm/internal/keeper/keeper.go`

**Impact** Due to the absence of UTF-8 validation and Unicode normalization, two nearly identical `Denom` may cause seemingly duplicate entries, downstream miscomparisons and/or panics.

**Description** The `ExistDenom()` function shown below[10] is intended to check whether a `denom` already exists. This is performed by looking up the `denom` creator on line 107 and comparing the length of the resulting string to zero. Note that `denom` is a Golang string.

```
104 func (k Keeper) ExistDenom(ctx sdk.Context, denom string) (string, bool) {
105     storedSupplyCoins := k.supplyKeeper.GetSupply(ctx).GetTotal()
106     //return storedSupplyCoins.AmountOf(denom) != sdk.ZeroInt() || len(k.GetOp...
107     if len(k.GetDenomCreator(ctx, denom)) != 0 {
108         return fmt.Sprintf("ccmKeeper.GetDenomCreator(ctx,%s) is %s", denom,
          ➜  sdk.AccAddress(k.GetDenomCreator(ctx, denom)).String()), true
109     }
110     if !storedSupplyCoins.AmountOf(denom).Equal(sdk.ZeroInt()) {
111         return fmt.Sprintf("supply.AmountOf(%s) is %s", denom, storedSupplyCoi...
112     }
113     return "", false
114 }
```

UTF-8 is the de facto character encoding used on the modern web[11] and in Golang applications. However, a Golang string is no more than simply a slice of bytes. The official Golang blog[12] states:

> It's important to state right up front that a string holds arbitrary bytes. It is not required to hold Unicode text, UTF-8 text, or any other predefined format. As far as the content of a string is concerned, it is exactly equivalent to a slice of bytes.

The code above neglects to validate that the `denom` bytes are indeed a valid UTF-8 encoded string. As a result, downstream logic may attempt to perform string operations under the assumption of correctness (potentially involving Golang string literals which are always properly encoded UTF-8 bytes) with undesired results such as panics Golang provides a string validation utility function[13] that can validate proper encoding. Further, Golang version 1.13 and above provides `strings.ToValidUTF8`[14] which makes validation far easier; see finding NCC-ONCH008-004 on page 8.

Separately, characters with accents or other modifiers can have multiple correct Unicode encodings. For example, the Á (a-acute) glyph can be encoded as a single character U+00C1

[10] https://github.com/polynetwork/cosmos-poly-module/blob/f917a9a3331f34a7d9ee86bdbd42a8337a3d2c18/ccm/internal/keeper/keeper.go
[11] https://w3techs.com/technologies/details/en-utf8
[12] https://blog.golang.org/strings
[13] https://golang.org/pkg/unicode/utf8/#ValidString
[14] https://golang.org/pkg/strings/#ToValidUTF8

(the "composed" form) or as two separate characters U+0041 then U+0301 (the "decomposed" form). In some cases, the order of a glyph's combining elements is significant and in other cases different orders must be considered equivalent.[15] Normalization[16] is the process of standardizing string representation such that if two strings are canonically equivalent and are normalized to the same normal form, their byte representations will be the same. Only then can string comparison and ordering operations be relied upon.

String normalization is significant because Onchain partners may establish `denom` identifiers involving these type of characters. Without string normalization, different third parties may establish identifiers that appear identical but are in fact distinct such as Bank of Álpha and Bank of Álpha. There are four standard normalization forms,[17] of which NFC is the most popular and is suitable for the Onchain application.

The proper place for these checks may be outside of this file (e.g. when initially receiving the string), but this finding is reported as it may impact the project target files here. Ensure the checks are also in place for code that traverses `CreateDenom` in the `btcx/internal/keeper/keeper.go` source file as the logger `Sprintf` may panic on line 95 if non-UTF-8 is encountered.

More information about Unicode and UTF-8 attacks can be found in this Black Hat presentation: https://www.blackhat.com/presentations/bh-usa-09/WEBER/BHUSA09-Weber-UnicodeSecurityPreview-SLIDES.pdf.

**Recommendation** First, utilize the `strings.ToValidUTF8` function or the `ValidString()` function from the Golang `utf8` package, to ensure correct character encoding immediately after byte deserialization.

Second, perform string normalization to form NFC on both deserialization and serialization to ensure a consistent string representation. The Golang `norm` package[18] provides suitable functionality for this purpose.

Finally, documenting character encoding expectations and normalization operations on strings will be very valuable for Onchain users.

[15] http://unicode.org/reports/tr15/tr15-22.html
[16] https://blog.golang.org/normalization
[17] http://unicode.org/reports/tr15/#Norm_Forms
[18] https://godoc.org/golang.org/x/text/unicode/norm

**Finding** **Golang Code Improvements**

**Risk** **Informational** Impact: None, Exploitability: None

**Identifier** NCC-ONCH008-005

**Category** Other

**Component** Poly Network

**Location**
- `ft/internal/keeper/ft_crossed_indenpently.go`
- `ft/internal/keeper/keeper.go`
- `headersync/internal/keeper/keeper.go`
- `headersync/internal/keeper/keeper.go`

**Impact** Unused struct fields may be indicators of incomplete code or refactoring oversights. Extraneous functional calls may impact code legibility and/or maintainability.

**Description** Static analysis is an important part of the development flow because it helps support consistency, follow best practices, and find potential bugs. Consistency helps code legibility and maintenance, best practices typically avoid unknown/undesired behavior, and some common bugs have a known signature. The Poly Network code exhibits a small number of static issues that should be resolved.

- On line 87 of `ft/internal/keeper/ft_crossed_indenpently.go` shown below, the `fmt.Sprintf()` function is not needed as it contains a constant string.

```
86 if toAssetHash == nil {
87     return types.ErrLock(fmt.Sprintf("toAssetHash is empty"))
88 }
```

- On line 34 of `ft/internal/keeper/keeper.go` shown below, the struct field `paramSpace` is declared but never used. The adjacent files `./key.go` and `../types/keys.go` appear to have related unused references on lines 28 and 29 respectively.

```
31 type Keeper struct {
32     cdc           *codec.Codec
33     storeKey      sdk.StoreKey
34     paramSpace    params.Subspace
35     authKeeper    types.AccountKeeper
36     bankKeeper    types.BankKeeper
37     supplyKeeper  types.SupplyKeeper
38     ccmKeeper     types.CrossChainManager
39 }
```

- On line 41 of `headersync/internal/keeper/keeper.go` shown below, the struct field `paramSpace` is declared but never used. The adjacent files `./key.go` and `../types/keys.go` appear to have related unused references on lines 27 and 26 respectively.

```
38 type Keeper struct {
39     cdc        *codec.Codec
40     storeKey   sdk.StoreKey
41     paramSpace params.Subspace
42 }
```

- On line 121 of `headersync/internal/keeper/keeper.go` shown below, the use of the simpler `fmt.Errorf(..)` is preferred over `errors.New(fmt.Sprintf(..))`.

```
if header.Height <= consensusPeer.Height {
    return types.ErrSyncBlockHeader("Compare height", header.ChainID,
    ➜ header.Height, errors.New(fmt.Sprintf(
    ➜ "Stored consensus header.Height: %d, trying to sync height:%d",
    ➜ consensusPeer.Height, header.Height)))
}
```

Resolving the above issues will improve code robustness. Note that this is an informational finding only.

**Recommendation** Determine if there is a reason for the existence of the unused struct fields. If there is no reason for their existence, remove them and any related (unused) references. Adapt the use of `fmt.Sprintf` as suggested. Incorporate static analysis linting into the project development flow.

## 1.0 Overview

This informal *informational* section highlights selected portions of the engagement methodology used, priority concerns investigated, and a few observations that do not warrant security-related findings. The primary strategy for this project relied heavily on manual source code inspection, lightly supported by local compilation and testing. Priority was given to robust implementation techniques, the validation of inputs and outputs upon deserialization and serialization respectively, correctness of intermediate values, alignment to specifications, interoperability across components and system-level interactions, along with general secure coding practices that could potentially impact legitimate operation.

## 2.0 Project scope and reference material

The primary Ontology code was from commit `f3cedab`[19] of https://github.com/ontio/ontology and included the following targets:

- smartcontract/service/native/cross_chain/cross_chain_manager/[cross_chain.go|param.go|utils.go|test]
- smartcontract/service/native/cross_chain/header_sync/[header_sync.go|param.go|states.go|utils.go|test]
- smartcontract/service/native/cross_chain/common/[common.go|header.go|states.go]

The primary Poly Network code was from commit `f917a9a`[20] of https://github.com/polynetwork/cosmos-poly-module and included the following targets:

- btcx/internal/keeper/keeper.go
- ccm/internal/keeper/keeper.go
- ft/internal/keeper/ft_crossed_indenpently.go
- ft/internal/keeper/keeper.go
- headersync/internal/keeper/keeper.go
- lockproxy/internal/keeper/keeper.go

Note that while the above components are part of a *much larger system*, many of the necessary supporting functions were also lightly reviewed. Pertinent references supporting the overall review include the following:

- Keepers https://docs.cosmos.network/master/building-modules/keeper.html
- Ontology Golang SDK https://github.com/ontio/ontology-go-sdk
- Poly Network whitepaper https://www.poly.network/PolyNetwork-whitepaper.pdf
- CoinBugs whitepaper https://research.nccgroup.com/wp-content/uploads/2020/03/NCC-Group-Whitepaper-Coinbugs.pdf
- Blockchain attack surface https://arxiv.org/pdf/1904.03487.pdf

## 3.0 General survey

The code was cloned, the specific target commit configured and the material broadly surveyed. The GoLand IDE `inspect` function was run on each target followed by `staticcheck` from https://staticcheck.io/. While outside of the project scope, dependencies were examined via `go list -m -u all`. Test cases pertinent to the project targets were run where available.

The entire code base was briefly surveyed for overall context, followed by an inspection of each target source file.

### Observations

- The Ontology Golang version is outdated, see finding NCC-ONCH008-004 on page 8 for more details.
- The target filename `ft_crossed_indenpently.go` may contain a misspelling (e.g. `independently`)
- Four opportunities for coding improvement were found and documented in finding NCC-ONCH008-005 on page 15.

[19] https://github.com/ontio/ontology/tree/f3cedabc969f485301e501d96e972c25f9de32c1
[20] https://github.com/polynetwork/cosmos-poly-module/tree/f917a9a3331f34a7d9ee86bdbd42a8337a3d2c18

- The Poly Network code appears to have very minimal test coverage.
- The tests in `headersync/internal/keeper` **fail** (due to an inability get consensus peers on line 76 of the `keeper_test.go` source file).

## 4.0 Low-level implementation review

The code was manually reviewed for common implementation issues related to dangerous, exceptional or unintended code paths. State management operation was inspected in the context of successful and unsuccessful function calls. Sources of randomness were identified and evaluated along with the suitability of cryptographic primitives.

Additionally, function sequences and data flows were reviewed for unintended gaps in serialization and deserialization.

### Observations

- An unvalidated uint64 → uint32 conversion happens on line 106 of `ontology/smartcontract/service/native/cross_chain/cross_chain_manager/param.go` see finding NCC-ONCH008-001 on page 5 for details.
- The Poly Network code has several instances of a deterministic random number generator, but this is acceptable as it is used only for repeatable purposes (e.g. test or network collision backoff time).
- The `Deserialization()` function in `smartcontract/service/native/cross_chain/cross_chain_manager/param.go` could validate additional aspects of the deserialized data, such as length of strings and VarBytes > `0`. This would allow errors to be detected sooner rather than propagating into downstream logic. For example, the `AddressParseFromBytes()` function in `ontology/common/address.go` validates the length of the address on line 72.
- The `Serialization()` function for `KeyHeights` in `ontology/smartcontract/service/native/cross_chain/header_sync/states.go` sorts the `HeightList` prior to encoding on line 63. Should the adjacent `Deserialize()` function either check for a sorted list or perform sorting as a final step?
- In `btcx/internal/types/expected_keepers.go`, lines 23 and 24 appear to import the same thing twice.

## 5.0 System-level operation

The code was considered at the system level where possible. Functional and design documentation was reviewed to better understand the system-level requirements. Planned use cases were reviewed against the assurances provided by the cryptography employed. Specific cases related to inappropriate access to cryptographic key and malicious tampering, re-ordering and replaying of messages were inspected.

### Observations

- The potential for confusion around Golang string encoding is significant, see finding NCC-ONCH008-006 on page 13.
- Several instances where `store.Get()` may return nil and cause downstream errors were found, see finding NCC-ONCH008-008 on page 11.
- No other issues were observed.

# Appendix B: Finding Field Definitions

The following sections describe the risk rating and category assigned to issues NCC Group identified.

## Risk Scale

NCC Group uses a composite risk score that takes into account the severity of the risk, application's exposure and user population, technical difficulty of exploitation, and other factors. The risk rating is NCC Group's recommended prioritization for addressing findings. Every organization has a different risk sensitivity, so to some extent these recommendations are more relative than absolute guidelines.

### Overall Risk

Overall risk reflects NCC Group's estimation of the risk that a finding poses to the target system or systems. It takes into account the impact of the finding, the difficulty of exploitation, and any other relevant factors.

**Critical** Implies an immediate, easily accessible threat of total compromise.

**High** Implies an immediate threat of system compromise, or an easily accessible threat of large-scale breach.

**Medium** A difficult to exploit threat of large-scale breach, or easy compromise of a small portion of the application.

**Low** Implies a relatively minor threat to the application.

**Informational** No immediate threat to the application. May provide suggestions for application improvement, functional issues with the application, or conditions that could later lead to an exploitable finding.

### Impact

Impact reflects the effects that successful exploitation has upon the target system or systems. It takes into account potential losses of confidentiality, integrity and availability, as well as potential reputational losses.

**High** Attackers can read or modify all data in a system, execute arbitrary code on the system, or escalate their privileges to superuser level.

**Medium** Attackers can read or modify some unauthorized data on a system, deny access to that system, or gain significant internal technical information.

**Low** Attackers can gain small amounts of unauthorized information or slightly degrade system performance. May have a negative public perception of security.

### Exploitability

Exploitability reflects the ease with which attackers may exploit a finding. It takes into account the level of access required, availability of exploitation information, requirements relating to social engineering, race conditions, brute forcing, etc, and other impediments to exploitation.

**High** Attackers can unilaterally exploit the finding without special permissions or significant roadblocks.

**Medium** Attackers would need to leverage a third party, gain non-public information, exploit a race condition, already have privileged access, or otherwise overcome moderate hurdles in order to exploit the finding.

**Low** Exploitation requires implausible social engineering, a difficult race condition, guessing difficult-to-guess data, or is otherwise unlikely.

## Category

NCC Group categorizes findings based on the security area to which those findings belong. This can help organizations identify gaps in secure development, deployment, patching, etc.

| | |
|---|---|
| **Access Controls** | Related to authorization of users, and assessment of rights. |
| **Auditing and Logging** | Related to auditing of actions, or logging of problems. |
| **Authentication** | Related to the identification of users. |
| **Configuration** | Related to security configurations of servers, devices, or software. |
| **Cryptography** | Related to mathematical protections for data. |
| **Data Exposure** | Related to unintended exposure of sensitive information. |
| **Data Validation** | Related to improper reliance on the structure or values of data. |
| **Denial of Service** | Related to causing system failure. |
| **Error Reporting** | Related to the reporting of error conditions in a secure fashion. |
| **Patching** | Related to keeping software up to date. |
| **Session Management** | Related to the identification of authenticated users. |
| **Timing** | Related to race conditions, locking, or order of operations. |

# Appendix C: Project Contacts

The team from NCC Group has the following primary members:


- Eric Schorn — Technical Lead
  eric.schorn@nccgroup.com
- Javed Samuel — Vice President, Practice Director, Cryptography Services
  javed.samuel@nccgroup.com


The team from Onchain has the following primary member:


- Walter Yang — Onchain; Shanghai Distributed Technologies Co., Ltd.
  walteryang@onchain.com