Gravity Bridge: How a Stolen Signing Key Drained $5.4 Million From the Ethereum and Cosmos Bridge

And why this was not the “denom mapping” bug that a lot of people think it was.

Summary

Here is the short version, so you know what you are reading before you dig in.

- **Protocol and commit.** Gravity Bridge, the bridge that connects Ethereum and the Cosmos ecosystem. The Ethereum contract is `0xa4108aA1Ec4967F8b52220a4f7e94A8201F2D906`, shown on Etherscan as “Gravity Bridge: Bridge” and verified. Any code I quote is pinned to the `althea-net/cosmos-gravity-bridge` repo at commit `92d0e12cea813305e6472851beeb80bd2eaf858d`.
- **Bug class.** Access control and key management. Someone got hold of the bridge signing key. This was not a logic bug in the Solidity, and it was not the denom mapping bug.
- **Impact.** Roughly $5.4 million gone: about $4.3M in USDC, 274 ETH and WETH worth about $553K, about $434K in USDT, and 14.164 PAXG worth about $64K.
- **Status.** This happened in the real world on 30 May 2026. The bridge was halted. Some of the money was cashed out through ChangeNOW and Binance, and some was pushed through Tornado Cash. As of 11 June 2026 the team has not published an official postmortem.

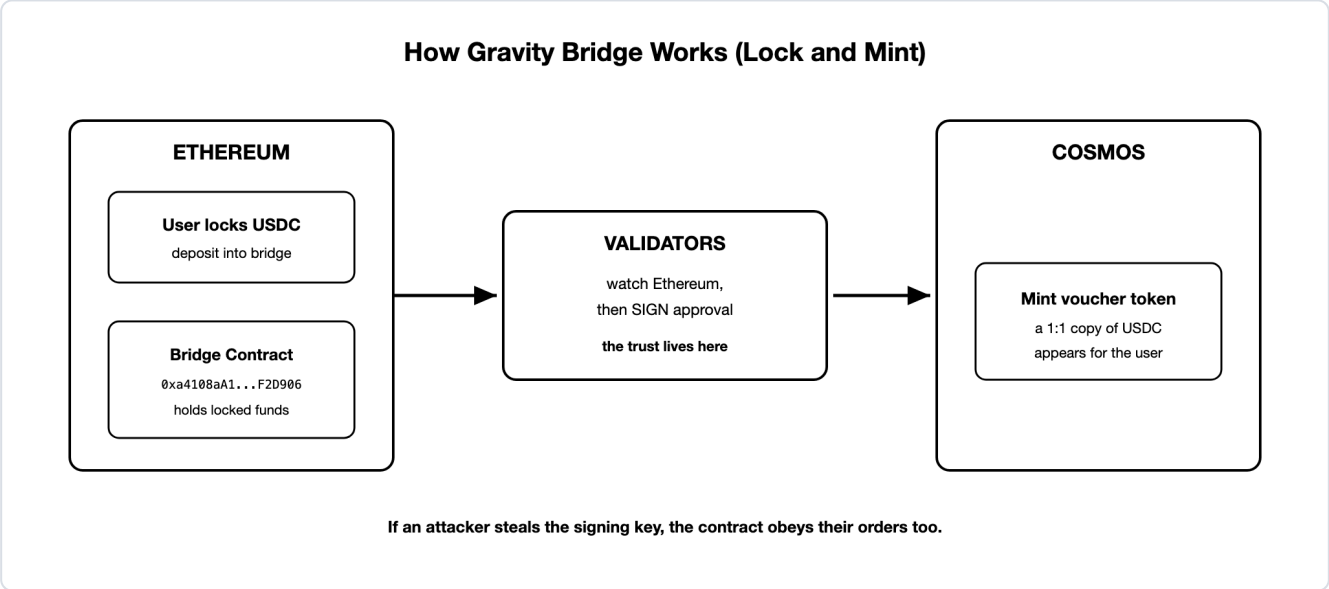
One thing to get straight before we start. A lot of people keep calling this a “denom mapping exploit.” It wasn’t. The money left because someone got the key that controls the bridge. That is a key management failure, not a coding flaw. The denom mapping bugs are real, but they belong to a separate and older story from audits back in 2020 and 2021. I cover both here, and I keep them clearly apart.

Background

A bridge lets you move tokens between two blockchains that otherwise cannot talk to each other. Gravity Bridge connects Ethereum and the Cosmos ecosystem.

It runs on a simple idea that people call lock and mint:

1. You lock an ERC20 token, say USDC, inside the bridge's Ethereum contract.
2. A group of validators, the computers that run the Cosmos chain, watch Ethereum. When enough of them see your deposit, they sign an approval.
3. The Cosmos side then mints a matching token for you. To go back the other way, you burn the Cosmos token and the validators sign a release on Ethereum.



How Gravity Bridge works. Lock on Ethereum, validators sign, mint on Cosmos. The whole system trusts the signing keys.

Everything rests on one assumption: only the real validator keys can approve a transfer. The Ethereum contract checks those signatures before it moves a single dollar. So if an attacker ever gets the signing keys, the contract will happily do whatever the attacker tells it, because as far as the contract can tell, the orders are real.

Hold on to that sentence. It is the whole incident in one line.

You can read the official docs at https://www.gravitybridge.net/ and https://github.com/Gravity-Bridge/Gravity-Docs.

Vulnerable Code

This story has two code threads, and only one of them is the real cause. Let me separate them.

Thread A: the real attack had no single vulnerable function

The 30 May theft did not exploit a flaw in the bridge's Solidity. The contract checked the signatures correctly and released the funds correctly. The weak point sat off the chain: the secret signing key was in the wrong hands. You cannot point at a line of code here, because the failure was in how that key was kept and protected. I get into that fully in the Root Cause section.

Thread B: the denom mapping bugs were real, but auditors found them, and they did not cause the 2026 attack

This is the part that gives the "denom mapping" name its meaning. These bugs are genuine, but they come from audit work in 2020 and 2021, and nothing connects them to the 2026 theft.

Everything in Thread B is audit material, not something used in the wild in 2026.

What does denom mapping even mean? On Cosmos, every token has a name called a denom, short for denomination. When an Ethereum ERC20 token crosses the bridge, the Cosmos side has to decide which denom this token maps to. That match between an ERC20 contract address and a Cosmos denom is the denom mapping. Get it wrong and two different tokens can look like the same thing.

Here is the original duplicate token report, GitHub issue #181, opened on 13 April 2020:

> An attacker can duplicate any ERC20 that has come across the bridge by creating a copy of that ERC20's contract on Ethereum and relaying over copied tokens. The copied tokens will be indistinguishable from originals on the Cosmos side.

Source: https://github.com/cosmos/gravity-bridge/issues/181

The mapping logic lives in `module/x/gravity/keeper/ethereum_event_handler.go`:

```
// Is this token originally from Cosmos, or from Ethereum?
isCosmosOriginated, denom := a.keeper.ERC20ToDenomLookup(ctx, event.TokenContract)

if !isCosmosOriginated {
    // Ethereum native token: mint a Cosmos voucher AFTER a safety check.
    // DetectMaliciousSupply(...) guards against an impossible supply,
    // ... then MintCoins(...)
}

// Overflow guard so a token cannot claim an absurd supply.
if newSupply.BitLen() > 256 {
    return sdkerrors.Wrap(types.ErrSupplyOverflow, denom)
}
```

On the Ethereum side, in `solidity/contracts/Gravity.sol` at commit `92d0e12` (611 lines), anyone can register a token:

```
// Gravity.sol, lines 546 to 565 (commit 92d0e12)
function deployERC20(
    string memory _cosmosDenom,
    string memory _name,
    string memory _symbol,
    uint8 _decimals
) public {                      // public, with NO access control: anyone can call it
    // deploys a fresh ERC20 and emits an event that the oracles must parse
}
```

That `public` with no modifier is the foothold. Put it together with how the oracles read the name, symbol, and denom, and the Code4rena audit (26 August to 8 September 2021) found these:

ID	Severity	Explanation
H-02	High	Put strange non UTF8 characters in a token name and the oracle's log reader chokes. It gets stuck in a fail loop and freezes attestations, so the bridge stops confirming transfers.
H-03	High	Push an oversized denom, name, or symbol through `deployERC20` and you blow past the oracle's roughly 10MB log limit, which knocks the oracles out of sync.
M-04	Medium	`sendToCosmos` trusts the amount you say you sent. For a fee token that burns a cut on transfer, the contract actually receives less, so the Cosmos side credits more than really arrived.

The full audit is at https://code4rena.com/reports/2021-08-gravitybridge.

Root Cause

The 2026 attack: the key, not the code

Here is the rule that was supposed to hold:

Only the real bridge signers can approve a withdrawal from the Ethereum contract.

On 30 May 2026 that rule broke. Not because the contract did its math wrong, but because the secret signing key itself landed in the attacker's hands. Once you hold the key, every signature you make looks completely valid. The contract did its job perfectly. It just took orders from the wrong person.

The firms that flagged the incident, PeckShield and Cyvers, and the investigator known as Specter, all point to the same thing: the bridge contract key, or the signing path, was compromised, and the problem was not in the smart contract code itself.

It helps to split the symptom from the cause:

- The symptom: funds left the contract through what looked like normal, signed withdrawals.
- The cause: the attacker controlled the key that makes those signatures.

No amount of Solidity auditing catches this, because the bug is not in the Solidity. It lives in how the key was stored, shared, or protected away from the chain. We still do not know exactly how it leaked, whether it was phishing, a hacked server, a leaked secret, or someone on the inside. The team has not said.

The denom mapping bugs: identity you can fake

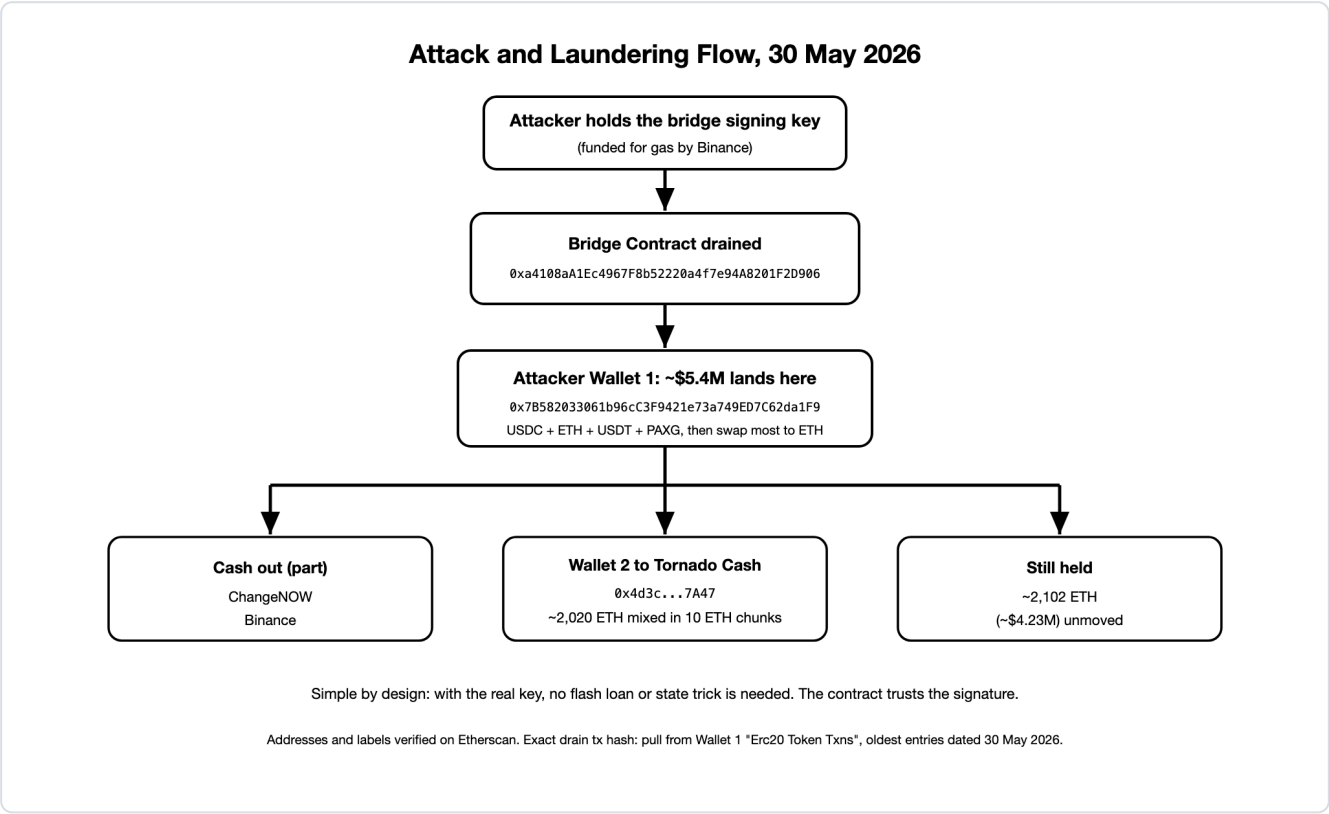
The audit bugs share a different broken rule:

A token's Cosmos identity must be tied to something an attacker cannot fake, namely its real contract address, and every string or amount the attacker hands you must be treated as hostile.

The early design leaned on copyable metadata and on amounts the sender reported about themselves. That is the classic Web3 mistake: treating attacker supplied input as if it were trustworthy data.

Exploit Walkthrough

This was an operational attack, so there is no flash loan and no clever state trick. It is brutally simple once you have the key.



The full attack and laundering flow, with the wallet and contract addresses verified on Etherscan.

1. **Setup.** The attacker gets gas money from Binance into Wallet 1. With the stolen key in hand, no loans or positions are needed.
2. **The trigger.** The attacker sends signed withdrawal orders to the bridge contract. The signatures check out, so the contract releases the locked tokens.
3. **The drain.** About $4.3M USDC, 274 ETH and WETH, about $434K USDT, and 14.164 PAXG move into Wallet 1, roughly $5.4 million in total.
4. **Convert.** Most of the stablecoins get swapped into ETH, which is easier to move and hide.
5. **Launder.** Part of it is cashed out through ChangeNOW and Binance. ETH is forwarded to Wallet 2, which pushes about 2,020 ETH through Tornado Cash in neat 10 ETH pieces. Partial transaction IDs seen on Wallet 2 include `0x989127...`, `0xcaddca...`, `0xf2d528...`, and `0x934dd3...`.

6. **The rest sits.** At reporting time the attacker still held about 2,102 ETH, worth roughly $4.23 million, sitting onchain and unmoved.

Every address below is something you can check on Etherscan right now.

Role	Address (click to open on Etherscan)	Etherscan label
Bridge contract that was robbed	`0xa4108aA1Ec4967F8b52220a4f7e94A8201F2D906`	“Gravity Bridge: Bridge” (verified)
Attacker Wallet 1, the main one	`0x7B582033061b96cC3F9421e73a749ED7C62da1F9`	“Gravity Bridge Exploiter”
Attacker Wallet 2, the helper	`0x4d3ca32e687e871a58b78AcAc73bE59AC37C7A47`	“Gravity Bridge Exploiter 2”
Mixer that was used	`0xd90e2f925da726b50c4ed8d0fb90ad053324f31b`	“Tornado.Cash: Router”

About the exact theft transaction ID

Let me be honest about a gap. No public source published the exact drain transaction ID, and Etherscan blocks automated tools from its date filter pages, while its normal view only shows the latest 25 transactions, and the hack is older than that. You can pull it yourself in about two minutes:

1. Open Wallet 1 at https://etherscan.io/address/0x7B582033061b96cC3F9421e73a749ED7C62da1F9.
2. Click the “Erc20 Token Txns” tab.
3. Scroll down to the oldest entries dated 30 May 2026. The big USDC transfer whose “From” field is the bridge contract `0xa4108aA1...` is the theft. Click it to see the full ID and block number.

Proof of Concept

For the 2026 attack there is no proof of concept to write. You cannot reproduce a key compromise on a test fork, because the exploit simply is holding the secret key. A test would just say “call withdraw with the real signer’s key,” which proves nothing. For a live key compromise, the chain data above is the proof.

For the denom mapping thread, here is an illustrative sketch of the fee token issue, M-04. This is a teaching sketch, not a tested run, and the bug is already patched. It is not a ready to use weapon.

```
// ILLUSTRATIVE ONLY. The idea behind the M-04 fee token mismatch.
// A deflationary token takes a cut on every transfer.
contract FeeToken is ERC20 {
    function transferFrom(address f, address t, uint256 amt) public override returns (bool)
{
        uint256 fee = amt / 100;        // 1 percent burned on transfer
        super.transferFrom(f, t, amt - fee);
        return true;
    }
}

// The old sendToCosmos trusted `amount`, not the real change in balance:
//   token.safeTransferFrom(msg.sender, address(this), amount);   // received is less than
amount
//   state.lastEventNonce++;                                      // but it credits the full
amount
// So the Cosmos side mints more than the bridge actually holds.
```

A real, reproducible proof of concept would pin the repo, a fork block number, and a `forge test` command to run. That is exactly why I am flagging this one as conceptual only.

Impact

For the real 2026 attack:

- **Funds lost.** Roughly $5.4 million in total: about $4.3M in USDC, 274 ETH and WETH worth about $553K, about $434K in USDT, and 14.164 PAXG worth about $64K.
- **Who it hurts.** Anyone whose assets were locked in the bridge at the time, plus the wider pool of liquidity between Cosmos and Ethereum that depended on it.
- **What the attacker needed.** Just the signing key. That is the entire precondition, and it is also the hardest thing to get, which is why these attacks are rare but devastating.
- **Still exploitable?** The bridge was halted. Since the key was already abused, the fix is operational: rotate the keys, rebuild the signer setup, and find out how the key leaked.

For the denom mapping bugs: patched years ago, not used in 2026, and listed here only so you can learn from them.

The Fix

For the 2026 key compromise, the fix is operational, not a code diff. Rotate every signer key, move the keys into hardware security modules or a stronger multisig, and audit how the leak happened. You cannot patch your way out of a stolen key. You have to replace it and harden how it is stored.

For the denom mapping thread, the fixes are code. The duplicate token problem was closed by tying a token's identity to its real ERC20 contract address, stored in both directions, so a copycat contract maps to a different denom and cannot pretend to be the original:

```
// cosmos_originated.go: store the mapping in BOTH directions, keyed by contract address
+ store.Set(types.MakeDenomToERC20Key(denom), tokenContract.Bytes())
+ store.Set(types.MakeERC20ToDenomKey(tokenContract), []byte(denom))
  // A copied ERC20 has a DIFFERENT address, so it gets a DIFFERENT denom. No impersonation.
```

For M-04, the documented fix is to measure the balance before and after, and credit only what actually arrived:

```
- token.safeTransferFrom(msg.sender, address(this), _amount);
- // credited _amount blindly
+ uint256 beforeBal = token.balanceOf(address(this));
+ token.safeTransferFrom(msg.sender, address(this), _amount);
+ uint256 received = token.balanceOf(address(this)) - beforeBal;
+ // credit `received`, never the requested `_amount`
```

These fix the real cause, fakeable identity and trusted input, not just the surface symptom.

Honesty note. These diffs show the documented fix approach, not the exact merged commits, because I could not pin down the precise fix commit hash for each finding. Treat them as faithful reconstructions of the mitigation.

Takeaways

1. **Most bridge losses are key losses, not logic losses.** A flawless contract is only as safe as the keys that command it. When you threat model a bridge, spend at least as much time on key custody, things like hardware modules, multisig, and signer rotation, as you spend on the Solidity.

2. **A verified contract is not the same as a safe protocol.** Etherscan's green check only means the published code matches the source. It says nothing about who holds the keys.
3. **Never trust input that the attacker controls.** The denom mapping bugs all share one root: trusting a name, a symbol, or a self reported amount. Tie identity to things an attacker cannot fake, like contract addresses, and measure real balances instead of believing claimed ones.
4. **Name the bug correctly.** Calling this a denom mapping exploit sends defenders to fix the wrong thing. Keep the symptom and the cause apart. The symptom was signed withdrawals. The cause was a stolen key.
5. **Follow the money, not just the code.** For a live incident, the chain is the source of truth. Labels, funding paths like the Binance gas coming in, and mixer flows like the 10 ETH chunks into Tornado Cash, tell the story even when no postmortem exists yet.

Disclosure Timeline

Date	Event
13 April 2020	The denom mapping duplicate token issue, #181, is opened. Historical and separate.
26 Aug to 8 Sep 2021	The Code4rena audit surfaces the denom, name, and symbol findings, H-02, H-03, and M-04. Fixed later.
30 May 2026	The bridge is drained of about $5.4 million. Flagged the same day by PeckShield and Cyvers. Specter pins it on a signing key compromise. The bridge is halted.
30 May to early June 2026	The attacker swaps to ETH, cashes out through ChangeNOW and Binance, mixes about 2,020 ETH through Tornado Cash, and still holds about 2,102 ETH.
11 June 2026	No official Gravity Bridge postmortem has been published yet.

Responsible disclosure: every detail here is already public, and the code bugs are long fixed, so nothing in this post is a live weapon.

Sources

- Etherscan, bridge contract: https://etherscan.io/address/0xa4108aa1ec4967f8b52220a4f7e94a8201f2d906
- Etherscan, Exploiter 1: https://etherscan.io/address/0x7B582033061b96cC3F9421e73a749ED7C62da1F9
- Etherscan, Exploiter 2: https://etherscan.io/address/0x4d3ca32e687e871a58b78AcAc73bE59AC37C7A47
- PeckShield via Bitcoin.com: https://news.bitcoin.com/gravity-bridge-exploit-5-4-million-binance-changenow-2026/
- Cryptopolitan: https://www.cryptopolitan.com/hacker-drain-gravity-bridge/
- BeInCrypto on the key compromise: https://beincrypto.com/gravity-bridge-hack-key-compromise-5m/
- CryptoTimes: https://www.cryptotimes.io/2026/05/30/gravity-bridge-hit-in-5-4m-exploit-amid-suspected-key-compromise/
- GitHub issue #181: https://github.com/cosmos/gravity-bridge/issues/181
- Gravity.sol at commit 92d0e12: https://github.com/althea-net/cosmos-gravity-bridge/blob/92d0e12cea813305e6472851beeb80bd2eaf858d/solidity/contracts/Gravity.sol
- Code4rena audit: https://code4rena.com/reports/2021-08-gravitybridge