Deep Freeze

An alternative to physically hackable cold storage.

It’s not as simple as “get a hardware wallet”

After Ledger Hack, Who Can You Trust For Bitcoin Storage?

Cryptocurrency Hardware Wallets Can Get Hacked Too

New research shows vulnerabilities in popular cold-storage options that would have revealed their PINs.

Inside The Scam: Victims Of Ledger Hack Are Receiving Fake Hardware Wallets

A new attack vector following the Ledger data breach of July 2020 involves sending convincing but fake hardware wallets to victims.

Nor as simple as multi-factor authentication

Hackers drain cryptocurrency accounts of thousands of Coinbase users

> They apparently phished for passwords and then used a flaw in Coinbase's 2FA to empty the victims' accounts. Comments.

2FA isn’t Enough To Protect Your Data — Take The Extra Step and Lock Down Your Data with a Security Key

An Old School Hack Threatens Two-Factor Authentication

Security Framework: Know | Have | Are

- You can know a password
- You can have a hardware wallet
- You can be yourself (e.g., fingerprint, retinal scan)

Deep Freeze adds: *Calculate*

- Blockchains like Ethereum offer credibly neutral hosting of assets within a smart contract.
- With 2 modifications to a template smart contract, you can use a smart contract as a type of “deep freeze” cold storage with any programmable protection you like.
- Here, I discuss, *Calculate.*
  - You can *calculate* the key to withdraw assets, which then destroys the vault.
  - New vaults can be spun up with simple changes to the calculation, that (through cryptography) result in large changes in the key.

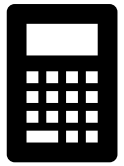
Leverage contract hashing to create simple to calculate, impossible to guess keys.

1. Create a hint/answer pair.
2. Generate the keccak hash for the answer.
3. Spin up a smart contract with that hash as the passphrase.
4. Add Assets.
5. Destroy the contract & withdraw all assets but giving it the un-hashed answer.
6. If the contract calculates the same hash you set it with when you created it, you've ended your asset deep freeze!

Alice uses a childhood memory to deep freeze

Alice remembers that in 3rd grade, there was food fight that got her hit in the head with an apple.
She sets following hint/answer pair:

Hint	Answer
What happened in 3rd grade with the apple?	I got hit in the head



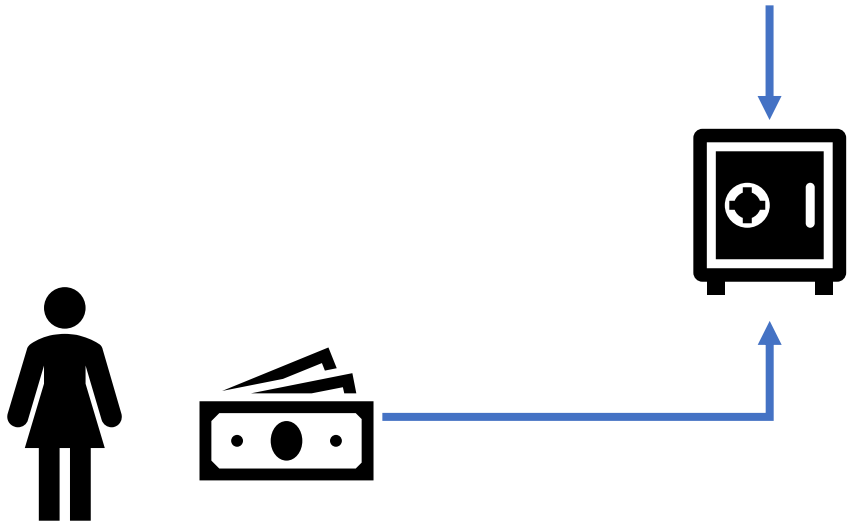
Using a public Keccak calculator for her answer:
"I got hit in the head"

Answer	Hash
I got hit in the head	876009e5f4588d510c19fc1e9abb92464eb54cd2e975f2099d4c10a6dd035e65

Alice uses a childhood memory to deep freeze

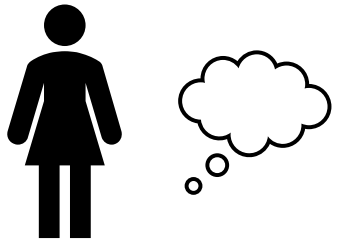
Alice spins up a smart contract with the parameters she calculated:

Hint	Hash
What happened in 3rd grade with the apple?	876009e5f4588d510c19fc1e9abb92464eb54cd2e975f2099d4c10a6dd035e65

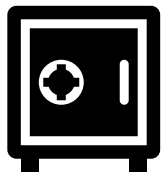


Alice adds assets to the contract

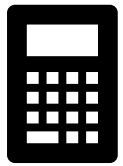
Alice uses a childhood memory to deep freeze



To withdraw, she sends her answer:
“I got hit in the head”
as transaction input data*



“I got hit in the head”



The smart contract calculates
the hash of the input data

If correct, the money is withdrawn
and the contract SELFDESTRUCTS*

Otherwise, it fails.

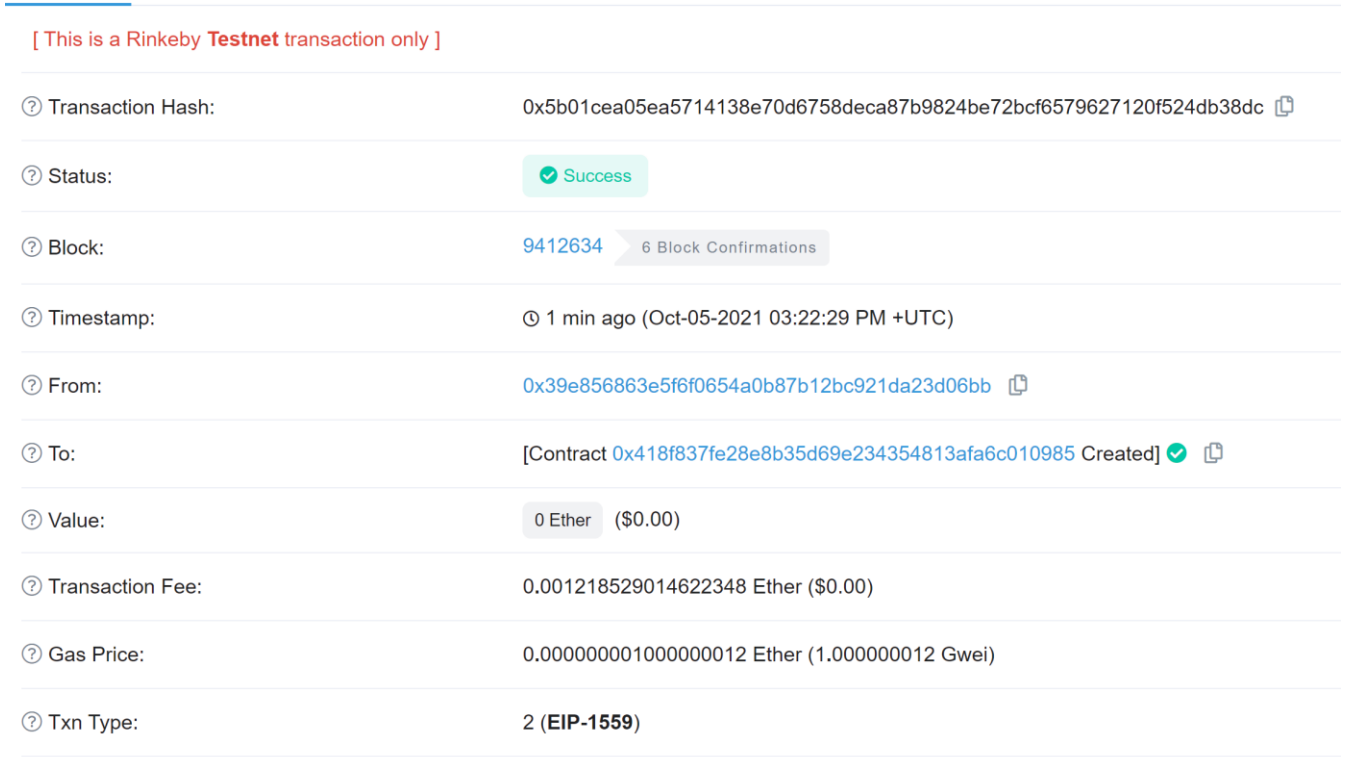
*Because transactions are public (including input data) Alice should never re-use the same contract. SELFDESTRUCT is the appropriate solution.

Proof of Concept

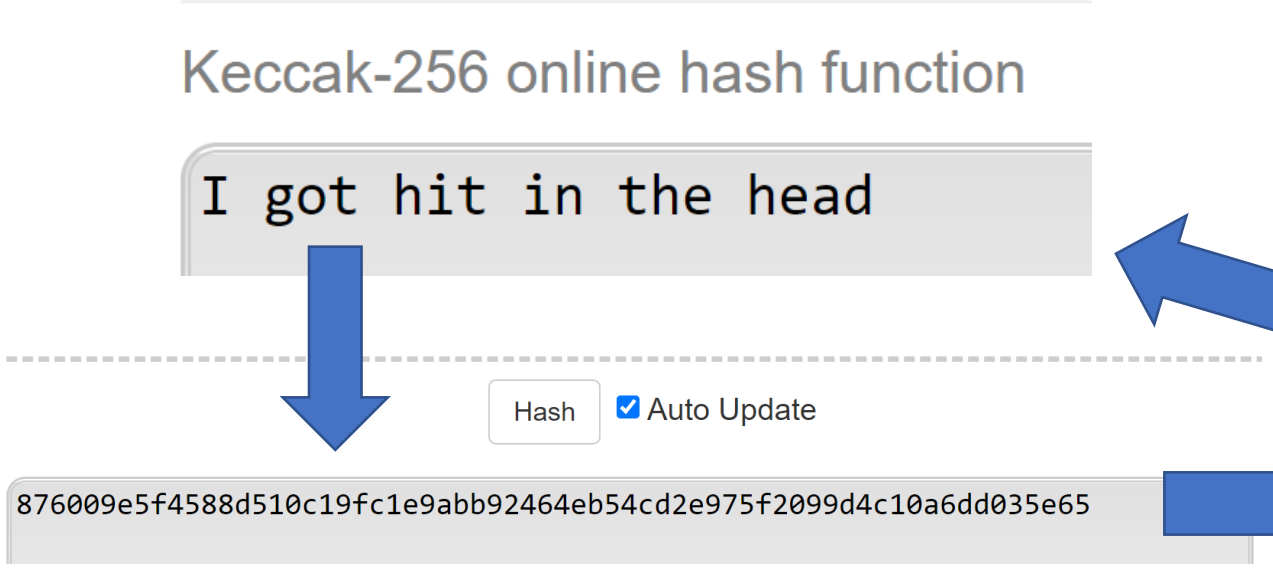
1. Launch CreateFreezer on Rinkeby
2. Use it to deploy a deep freeze [hint, “0x…”] parameters
3. Add Asset
4. Withdraw using commit/reveal
5. Confirm contract self destructed

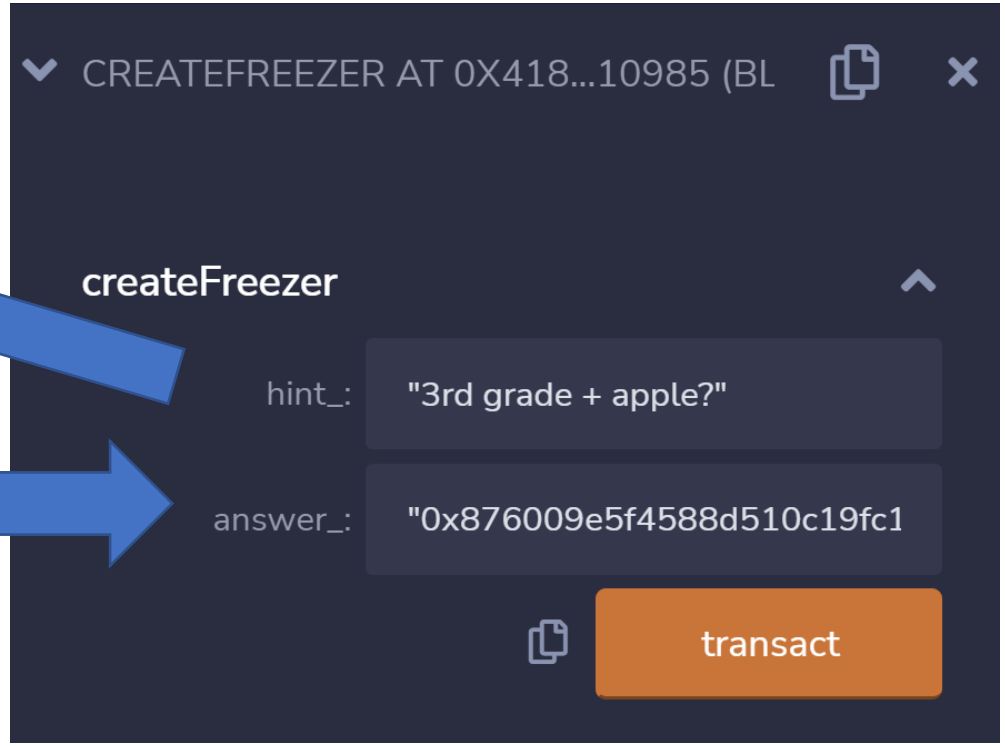
1. Create Rinkeby Contract

- https://rinkeby.etherscan.io/tx/0x5b01cea05ea5714138e70d6758deca87b9824be72bcf6579627120f524db38dc

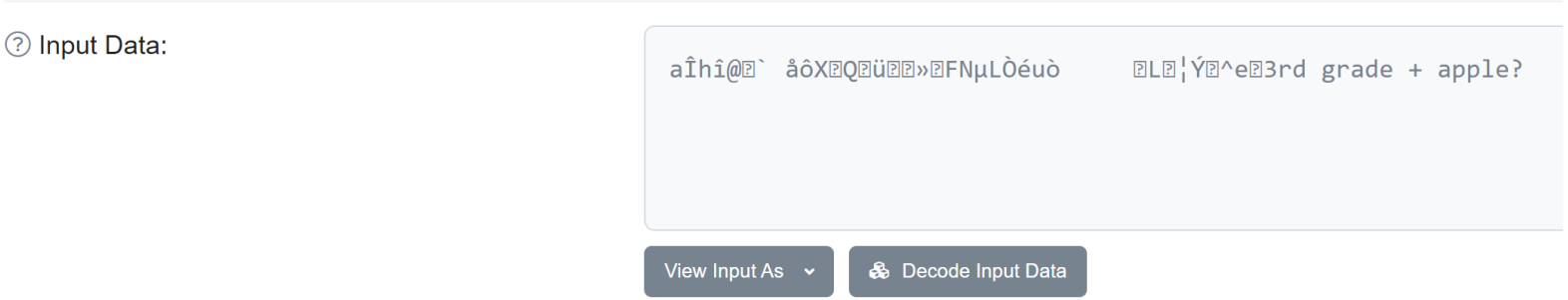


2. Deploy a Deep Freeze



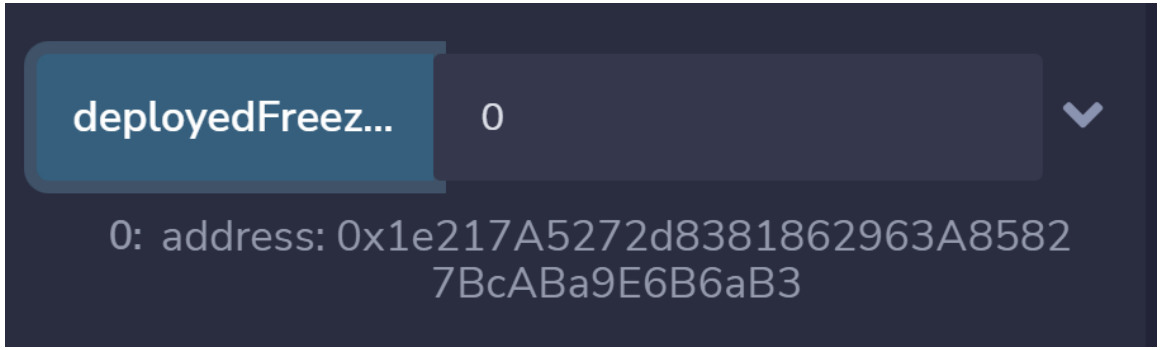


https://rinkeby.etherscan.io/tx/0xb7b436af1b005a4b07a17bc0c64f9595d88b0f96f7e6fd829ed9a28d1ba459a5



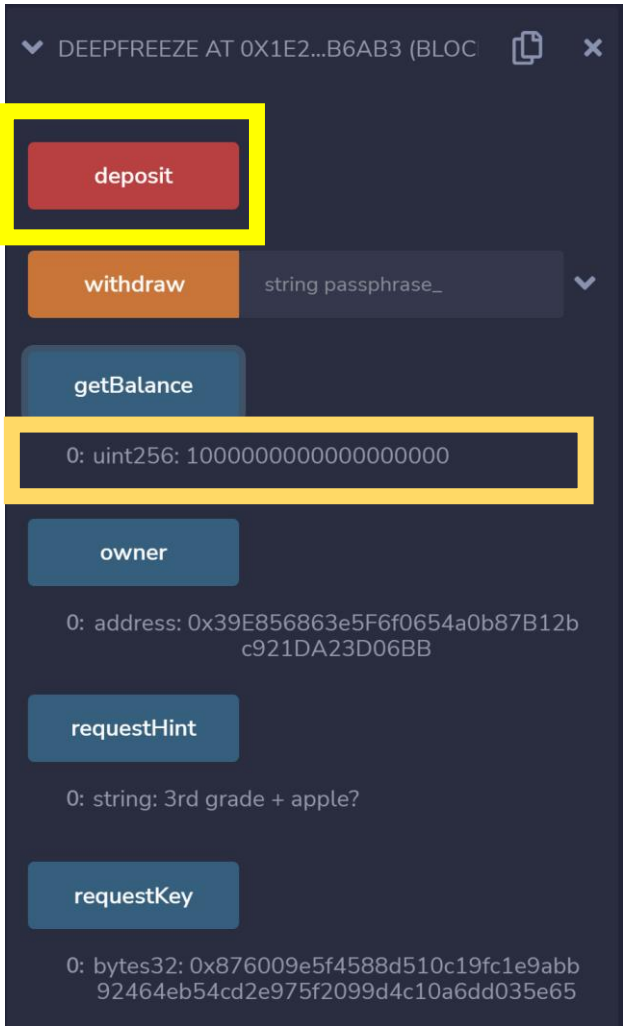
NOTE:
This input data is public!

3. Add Assets



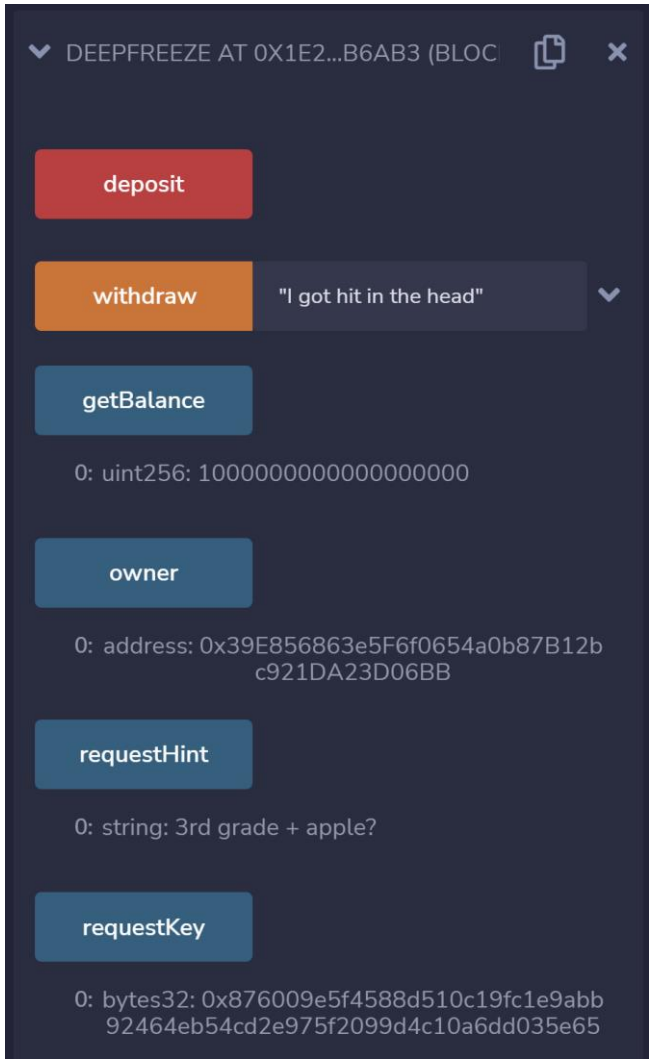
[This is a Rinkeby **Testnet** transaction only]

Transaction Hash:	0x8251d973e8fdbef0aeb325a9927595caf3e0f9e8ee3dedcf45ca61a8150f3d7c
Status:	Success
Block:	9412688 1 Block Confirmation
Timestamp:	20 secs ago (Oct-05-2021 03:36:03 PM +UTC)
From:	0x39e856863e5f6f0654a0b87b12bc921da23d06bb
To:	Contract 0x1e217a5272d8381862963a85827bcaba9e6b6ab3
Value:	1 Ether ($0.00)
Transaction Fee:	0.000021272000255264 Ether ($0.00)
Gas Price:	0.000000001000000012 Ether (1.000000012 Gwei)
Txn Type:	2 (**EIP-1559**)



https://rinkeby.etherscan.io/tx/0xcd3a198a9eaab662307a0cb2c7761272d40ce64af595e9afe51d1ee4cbbd2b0e

4. Withdraw Assets & 5. Confirm Self-destruct

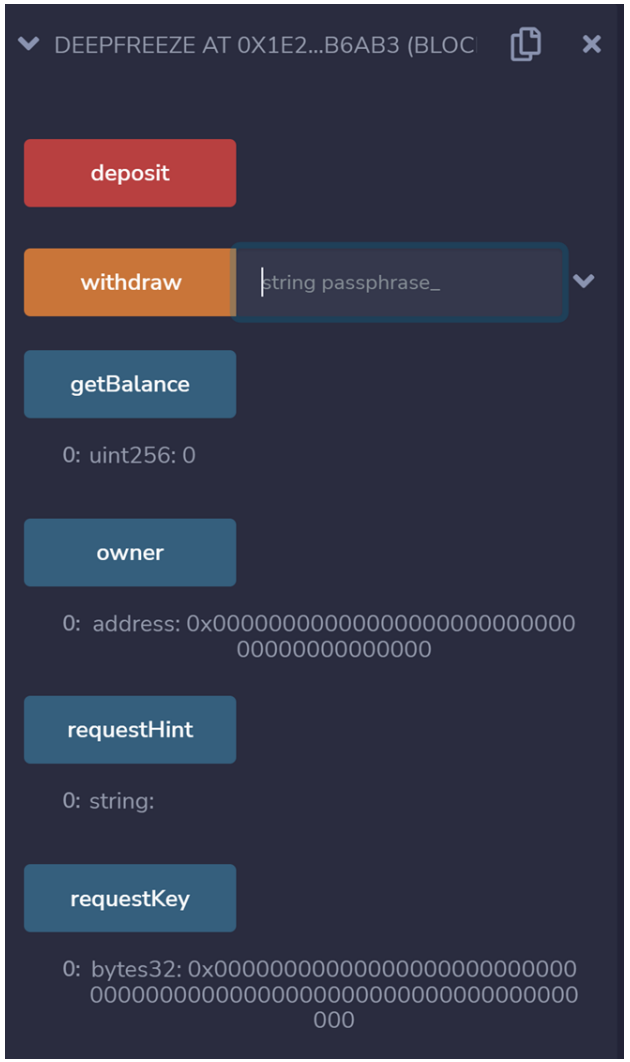


Balance is Empty

Owner is now Burn Address

Hint in storage is deleted

Answer in storage is deleted



How is this different from a password?

- *Calculate* is very similar to “Know” in the security framework, and yes, this really is barely different than a password. More advanced hints make the distinction a little clearer:
  1. “Whose anniversary is on your birthday?”
  2. Human Answer: “My best friend Bobby!”
  3. Keccak: 3a28e5b0176d6bb82ab100f524b774c3fd0fbcdf452d8b06bb630ed303a20ac7
- You give the contract (1) + (3)
- You can access (1) & (3) at anytime (using your main wallet)
- The secret is knowing your answer (“Bobby”) + how you would type it (“My best friend []!”).
- The key difference is you can independently test out different ways of typing your answer on a public Keccak calculator until you make the correct hash.