Understanding Tornado Cash

rkm0959

HYPERITHM

March 14th

Outline

Table of Contents

Cryptography

Cryptography is an art that utilizes **computationally hard problems**.

RSA : For large primes p, q of similar size, $n = pq$ is hard to factor.
Based on this, we assume the map $m \to m^e \pmod{n}$ is hard to invert.

ECC : For generic groups of prime order, discrete logarithm is hard.

Discrete Logarithm

Let G be a generic group. For $g, h \in G$, find $\alpha \in \mathbb{Z}$ such that $h = g^\alpha$.

We assume that this is hard for the elliptic curve over finite field.

Lattice : Lattice problems, such as SVP or CVP, are hard.
Knowing the secret key makes computation easier, i.e. "trapdoor".

ZK

Goal : Prove your knowledge without actually showing it.
"The Knowledge Complexity of Interactive Proof-System" [GMR89]

Suppose $g, h \in G$ (with $|G| = q$) are public and $h = g^{\alpha}$, while α is private.
How would you prove your knowledge of α while not directly showing α?

ZK : Definition

- Perfect Completeness : honest prover is able to give a proof.
- Perfect Zero-Knowledge : proof does not leak any extra information - i.e. a **simulator** can simulate the proof script without the secret
- Computational Knowledge Soundness : An **extractor** with full access to the prover's state can (with overwhelming probability) extract the witness whenever the prover produces a valid argument. Here, the prover may not be honest, i.e. it may be adversarial.

Schnorr's Protocol

- Prover selects random α_t and sends $u_t = g^{\alpha_t}$ to Verifier
- Verifier sends random challenge $c \in \mathbb{F}_q$ to Prover
- Prover computes $\alpha_z = \alpha_t + \alpha c \in \mathbb{F}_q$ and sends it to Verifier
- Verifier checks whether $g^{\alpha_z} = u_t \cdot h^c$

We can now prove

- Prover can be accepted iff Prover indeed knows α
- Verifier does not gain any knowledge on α - in fact, the Verifier can **simulate the output distribution of the possible interactions between the prover and verifier** while only knowing g, h.

Details : `https://rkm0959.tistory.com/202`

Schnorr's Signature

To sign a message m, we

- Select α_t randomly and let $u_t = g^{\alpha_t}$
- Choose c deterministically as $H(m, u_t)$ with secure hash H
- Let $\alpha_z = \alpha_t + \alpha c \in \mathbb{F}_q$ as before and output (u_t, α_z)

Random Oracle Model is required for proving the security of the scheme.

Details : `https://rkm0959.tistory.com/202`

Table of Contents

Goal

Consider an arithmetic circuit with addition and multiplication gates over $\mathbb{F}$.
Some input, output wires are public values, while others are not.
We want to prove our knowledge of values that make the circuit consistent.

Example

How would you prove knowledge of a, b, c such that $(a + b)c = 7$?

Quadratic Arithmetic Programs

We rewrite this into system of equations

$$\sum_{i=0}^{m} a_i u_{i,q} \cdot \sum_{i=0}^{m} a_i v_{i,q} = \sum_{i=0}^{m} a_i w_{i,q}$$

for some known coefficients $u_{i,q}, v_{i,q}, w_{i,q}$ for qth equation.
Assume $|F|$ is large. We use **Lagrange Interpolation** to find

$$u_i(r_q) = u_{i,q}, \quad v_i(r_q) = v_{i,q}, \quad w_i(r_q) = w_{i,q}$$

Now our goal is

$$\sum_{i} a_i u_i(x) \cdot \sum_{i} a_i v_i(x) = \sum_{i} a_i w_i(x) + h(x)t(x)$$

where

$$t(x) = \prod_{i=1}^{n} (x - r_i)$$

Quadratic Arithmetic Programs

Here, we assume $a_0 = 1$ and $a_1, \cdots, a_l$ are public.
We want to prove our knowledge of $a_{l+1}, \cdots, a_m$ that works.

To do this, Groth16 utilizes two key ideas -

- Linear Non-Interactive Proofs
- Elliptic Curve Pairings and Homomorphic Hidings

NIZK : A Detailed Look

Let $\mathcal{R}$ be a efficiently decidable relation. If $(\phi, \omega) \in \mathcal{R}$, then we denote ϕ as the *statement* and ω as the *witness*. A NIZK has

- $(\sigma, \tau) \leftarrow \text{Setup}(\mathcal{R})$, providing CRS σ and simulation trapdoor τ
- $\pi \leftarrow \text{Prove}(\mathcal{R}, \sigma, \phi, \omega)$, providing proof of knowledge
- $0/1 \leftarrow \text{Verify}(\mathcal{R}, \sigma, \phi, \pi)$ which verifies the proof
- $\pi \leftarrow \text{Sim}(\mathcal{R}, \tau, \phi)$, which simulates the proof with the trapdoor

NILP : A Detailed Look

A non-interactive linear proof consists of

- $(\sigma, \tau) \leftarrow \text{Setup}(\mathcal{R})$, as before
- $\pi \leftarrow \text{Prove}(\mathcal{R}, \phi, \omega)$ generates the proof - here, it runs $\Pi \leftarrow \text{ProofMatrix}(\mathcal{R}, \phi, \omega)$ then constructs the proof as $\pi = \Pi\sigma$
- $0/1 \leftarrow \text{Verify}(\mathcal{R}, \sigma, \phi, \pi)$ which verifies the proof via evaluation of multivariate polynomials, and checking whether the result is 0

here, we assume σ contains 1 as an entry.

NILP for Groth16 : Setup Phase

Remark : This Groth16 is the modified version for Powers of Tau.

Select $\alpha, \beta, \delta, x \in \mathbb{F}^\star$ randomly. Set $\tau = (\alpha, \beta, \delta, x)$.
The CRS σ contains of the following values -

- β, δ
- x^i for $0 \leq i \leq 2n-2$
- αx^i for $0 \leq i < n$
- βx^i for $1 \leq i < n$
- $\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}$ for $l+1 \leq i \leq m$
- $\frac{x^i t(x)}{\delta}$ for $0 \leq i \leq n-2$

where $n = \deg t$. This concludes the setup phase.

NILP for Groth16 : Prove Phase

Assume we know $a_1, \cdots, a_m$. Our proof consists of

$$A = \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta$$

$$B = \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta$$

$$C = \frac{\sum_{i=l+1}^{m} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + As + rB - rs\delta$$

with r, s random values from $\mathbb{F}$.
Note that A, B, C are linear combinations of entries of σ.

NILP for Groth16 : Verify Phase and Simulator

The verifier simply checks the equation

$$A \cdot B = \alpha \cdot \beta + \sum_{i=0}^{I} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x)) + C \cdot \delta$$

For simulation, simply choose random $A, B \in \mathbb{F}$ then compute C so that the above equation is true. This concludes the description of NILP.

NILP for Groth 16 : Properties

The above NILP has

- Perfect Completeness - easy, just follow the protocol
- Perfect Zero-Knowledge - easy, r, s are created for this
- Statistical Knowledge Soundness for **affine prover strategies**

Affine Prover Strategies

Here, the adversary wants to give a proof via a linear combination of entries of σ. We want to extract (with overwhelming probability) a full witness under the assumption that a valid proof was given.

Sketch : "Comparing Coefficients" + "Schwartz-Zippel Lemma"

Schwartz-Zippel Lemma

Let P be a nonzero n-variable polynomial of total degree d over field F. Let S be a finite subset of F, and $r_1, \cdots, r_n$ be i.i.d. uniform over S. Then,

$$\Pr[P(r_1, \cdots, r_n) = 0] \leq \frac{d}{|S|}$$

(used for proving rabin-karp hash, randomized general matching)

Homomorphic Hidings

We now want to perform this NILP process in a *hidden* manner.
How do we compute proof or verify the proof if every value is *hidden*?

Addition : DLP

Discrete Logarithm Problem (DLP)

For a generic group G with $|G| = p$ a prime, we are given g and $h = g^a$. The DLP is to find a given g, h, and it is a computationally hard problem.

Now addition is not hard. Consider $a, b \in \mathbb{F}_p$ and a generic group G with $|G| = p$ and generator g. We can encode and hide the value a as $E(a) = g^a$ and the value b as $E(b) = g^b$. Then, the value corresponding to $a + b$ is $g^{a+b} = g^a \cdot g^b$. Therefore, we have a method of calculating $E(a + b)$ from $E(a)$ and $E(b)$, while hiding the actual values of a, b.

Multiplication : Bilinear Pairings

Bilinear Pairings

Let q be a prime and G_1, G_2, G_T be groups of order q.
A pairing is a map $e : G_1 \times G_2 \to G_T$ such that

- $\forall a, b \in \mathbb{F}_q$ and $P \in G_1,\ Q \in G_2,\ e(aP, bQ) = ab \cdot e(P, Q)$
- e is non-trivial, but still efficiently computable

Assume that DLP on each of G_1, G_2, G_T are all hard.
Choose generators $g_1, g_2, g_T \in G_1, G_2, G_T$ such that $e(g_1, g_2) = g_T$.
Denote $E_1(x) = xg_1,\ E_2(x) = xg_2,\ E_T(x) = xg_T$.
Now $e(E_1(a), E_2(b)) = E_T(ab)$, which enables multiplication.

NILP to NIZK : The Full Groth16

We will need the following values as CRS - here, $E = (E_1, E_2)$.

- $E(\alpha), E(\beta), E(\delta)$
- $E(x^i)$ for $0 \leq i < n$
- $E_1(x^i)$ for $n \leq i \leq 2n-2$
- $E_1(\alpha x^i)$ for $0 \leq i < n$
- $E_1(\beta x^i)$ for $1 \leq i < n$
- $E_1\left(\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right)$ for $l+1 \leq i \leq m$
- $E_1\left(\frac{x^i t(x)}{\delta}\right)$ for $0 \leq i \leq n-2$

the simulation trapdoor remains the same.

The Full Groth16 : Prove Phase

Assume we know $a_1, \cdots, a_m$. Our proof consists of

$$E_1(A) = E_1(\alpha) + \sum_{i=0}^{m} a_i E_1(u_i(x)) + rE_1(\delta)$$

$$E_2(B) = E_2(\beta) + \sum_{i=0}^{m} a_i E_2(v_i(x)) + sE_2(\delta)$$

$$E_1(C) = \sum_{i=l+1}^{m} a_i E_1\left(\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right) + sE_1(A)$$
$$+ rE_1(B) - rsE_1(\delta) + E_1\left(\frac{h(x)t(x)}{\delta}\right)$$

with r, s selected random values from $\mathbb{F}$.

The Full Groth16 : Verify Phase and Simulator

The verifier, given $E_1(A), E_2(B), E_1(C)$, checks

$$e(E_1(A), E_2(B)) = e(E_1(\alpha), E_2(\beta)) + \sum_{i=0}^{l} e(E_1(\beta u_i(x) + \alpha v_i(x) + w_i(x)), E_2(a_i)) + e(E_1(C), E_2(\delta))$$

and the simulator logic remains the same.

Table of Contents

Optional : Short Note on Cryptographic Models

There are several cryptographic models we implicitly used.

- Random Oracle Model : God Exists and its name is Hash Functions
- Generic Group Model : only basic group operations are possible
- Common Reference String Model : we can create a god solely for the setup phase, then kill it afterwards so that no one has **toxic waste**

All of them have

- their strengths (easier security proofs)
- their weaknesses (far from reality)

Common Reference String

Where are we going to get the CRS from?
We need $E(\beta)$ in our CRS, but no one should have access to β.
If someone has enough of these secret values, (for example, the entire simulation trapdoor) then everything is busted - proof forgery is possible.

Review : The Case of ZCash

Review : The Case of ZCash

- far from begin scalable
- extremely complicated
- tough requirements are placed
- therefore, **participants must be determined in advance**

We need a new protocol for CRS generation. Enter **Powers of Tau**.

Powers of Tau : Sketch

Assume we want to generate $E(x)$ with no one knowing $x \in \mathbb{F}_p^\star$.
N people will collaborate - the ith person will

- receive the $i-1$th person's result $E(x_{i-1})$
- broadcast $E(y_i)$ and proof of knowledge of random $y_i \in \mathbb{F}_p^\star$
- send the result $E(x_i)$ where $x_i = x_{i-1}y_i$ and prove it

Now $x = y_1 y_2 \cdots y_N$ with no one knowing y_i except for their own.

However, the last user may choose y_n **adaptively**. To solve this, either

- (previous works) use a commitment scheme (like KNS!)
- (PoT) **use a random beacon to randomize the final output**

Cryptographic Tools : Exponent Checks

Given $E_1(a), E_1(b), E_2(c), E_2(d)$, we can check $a : b = c : d$ via

$$e(E_1(a), E_2(d)) = e(E_1(b), E_2(c))$$

This is one cool fact about groups with pairings, **DDH** is easy on it.

Decisional Diffie Hellman Problem (DDH)

Given g, ag, bg, cg, determine whether $c = ab$. If pairing exists, check

$$e(g, cg) = e(ag, bg)$$

Cryptographic Tools : Knowledge of Exponent Proof

Assume there exists a hash function $\mathcal{R}$ that hashes values to $G_2^\star$.
With public $E(\alpha)$, we can proof the knowledge of α as follows.
Given α and a public, deterministic string v, the proof is

$$(r, s) = (E_1(\alpha), \alpha\mathcal{R}(E_1(\alpha), v))$$

To verify this proof, given r, s and v, we check

$$e(E_1(1), s) = e(r, \mathcal{R}(r, v))$$

Cryptographic Tools : Random Beacon

A *random beacon* should

- give a public uniform random i.i.d. of previous values
- should not output the random number in advance
- should not be possible to manipulate

Powers of Tau : Round 1

The goal is to calculate the following portion of the CRS

- $E(\alpha), E(\beta)$
- $E(x^i)$ for $0 \leq i < n$
- $E_1(x^i)$ for $n \leq i \leq 2n-2$
- $E_1(\alpha x^i)$ for $0 \leq i < n$
- $E_1(\beta x^i)$ for $0 \leq i < n$

note that these CRS values are **application independent**

Powers of Tau : Round 1

After the natural initialization process, each player j

- generates $E(\alpha_j)$, $E(\beta_j)$, $E(x_j)$
- gives knowledge of exponent proof - v is selected as the transcript
- computes each CRS value after jth player contributed
- anyone can verify computations via exponent checks anytime

After all players are done, perform last touch with random beacon.

This is easier to explain with a demonstration :)

Powers of Tau : Linear Combination

We now linearly combine the computed CRS values to get

- $E_1(x^i t(x))$ for $0 \leq i \leq n-2$
- $E_1(\beta u_i(x) + \alpha v_i(x) + w_i(x))$ for $l+1 \leq i \leq m$

Since we have $E_1(x^i)$ for $0 \leq i \leq 2n-2$ and $E_1(\alpha x^i), E_1(\beta x^i)$ for $0 \leq i < n$, this is certainly computable in theory.
However, if done naively, this takes $\mathcal{O}(n(n+m))$ group operations.

Powers of Tau : Linear Combination

To overcome this issue, we use Fast Fourier Transform (FFT, NTT) -

- select n to be a power of 2
- take the prime field to be suitable for FFT/NTT
- choose r_q's to be a appropriate roots of unity

Now $t(x) = x^n - 1$, so computing all $E_1(x^i t(x))$ can be done in $\mathcal{O}(n)$.

Powers of Tau : Linear Combination

Now we have to calculate all $E_1(\beta u_i(x) + \alpha v_i(x) + w_i(x))$.
The idea is that $u_{i,q}, v_{i,q}, w_{i,q}$ are all very **sparse**.

Consider the example earlier, about proving $(a + b)c = 7$.

Powers of Tau : Linear Combination

Therefore, we use **Lagrange Basis** $L_i(x)$, which satisfy $L_i(r_j) = \delta_{i,j}$.
If we have $E_1(L_i(x)), E_1(\alpha L_i(x)), E_1(\beta L_i(x))$ for each i, then each $E_1(\beta u_i(x) + \alpha v_i(x) + w_i(x))$ will be computed very efficiently.

If each variable is included in a constraint $\mathcal{O}(a)$ times in average, this will take $\mathcal{O}(am)$ group operations. Note $a = \mathcal{O}(1)$ usually.

Powers of Tau : Linear Combination

Let's calculate $E_1(L_i(x))$. The others are the same due to linearity.

Let ω be a nth root of unity over $\mathbb{F}_p$, and $r_i = \omega^i$.

Lemma

$$L_i(x) = \frac{1}{n}\sum_{j=0}^{n-1} \omega^{-ij} x^j$$

Now it's clear that this can be computed via FFT style Divide and Conquer.
Result : $\mathcal{O}(n(n+m))$ to $\mathcal{O}(n \log n + am) \approx \mathcal{O}(n \log n + m)$.

Powers of Tau : Round 2

Now we compute

- $E(\delta)$
- $E_1\left(\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right)$ for $l+1 \leq i \leq m$
- $E_1\left(\frac{x^i t(x)}{\delta}\right)$ for $0 \leq i \leq n-2$

This is similar to Round 1, each player give their share of δ.

List of Tools

- Perpetual Powers of Tau : Since Round 1 of the Powers of Tau protocol is **application independent** we can run Round 1 in a perpetual way - any protocol can begin their application-dependent Round 2 in any state of the Perpetual Powers of Tau Round 1.
- Circom, SnarkJS : PL that compiles constraints into Groth16 proofs.
- Groth16 Aggregation : Merges n Groth16 proofs to size $\mathcal{O}(\log n)$

Table of Contents

Tornado Cash : Introduction

Tornado Cash provides anonymous transactions for fixed size of ETH.

You deposit ETH with your public address, then you prove in ZK that you are one of the depositors, while not revealing which of the depositors.

When you withdraw, you give up some of the ETH as a gas fee for a *relayer*, so your recipient does not need to hold ETH.

Tornado Cash uses ZK

Tornado Cash is based on Groth16 + Powers of Tau.

- MPC Ceremony : https://ceremony.tornado.cash/
- MPC Ceremony Announcement
- Post MPC Ceremony Announcement
- Tornado Cash becomes trustless

zkSNARKs on Ethereum

From EIP196/197, EVM supports precompiled contracts for bn128 curve.
This elliptic curve supports pairing operations and is suitable for FFT.

ETH Yellowpaper :
`https://ethereum.github.io/yellowpaper/paper.pdf`
EIP196 : `https://eips.ethereum.org/EIPS/eip-196`
EIP197 : `https://eips.ethereum.org/EIPS/eip-197`

Tornado Cash : Sketch of Internals

Prepare a **Merkle Tree** $\mathcal{T}$. We will do the following.

- **Deposit** : Add a hashed value as a leaf to $\mathcal{T}$ then recalculate the root
- **Withdraw** : Prove we know the preimage of the leaf and Merkle Proof

Tornado Cash : Details - Deposit

Let $H_1 : \{0,1\}^\star \to \mathbb{F}_p$ and $H_2 : (\mathbb{F}_p, \mathbb{F}_p) \to \mathbb{F}_p$ be hash functions.
Let $\mathcal{T}$ be a Merkle Tree of height 20 using H_2 as the hash.

To deposit N ETH, (where N is a supported amount of ETH in Tornado)

- Generate two random $k, r \in \{0,1\}^{248}$.
- Compute $H_1(k||r)$ and add this value as a leaf to $\mathcal{T}$

Note that $\mathcal{T}$ itself is public and computable from previous transactions.

Tornado Cash : Details - Withdraw

The goal is to prove the following statement in ZK.

Statement

The following values are public, i.e. directly sent as a part of transaction.

- R, the root of Merkle Tree $\mathcal{T}$
- h, the *nullifier hash*

We claim that we **know** $k, r,$ **Proof** where

- $h = H_1(k)$
- $H_1(k||r)$ exists as a leaf in $\mathcal{T}$
- **Proof** is a Merkle Proof for $H_1(k||r)$

Tornado Cash : Details - Withdraw

There are some very nice tricks here we have to discuss -

- **Preventing Double Spending** : The contract keeps track of $h = H_1(k)$ that was used for withdrawals, so double spending can be prevented unless some serious hash collisions occur
- **Preventing Frontrunning** : We can consider the case that an adversary in the mempool front-run your proofs. To prevent this, the circuit actually adds some dummy constraints on the recipient address, relayer address, and fee. These values are actually used during the ZK proof, so the proof is binding and cannot be front-run. To prevent frontrunners from stopping withdrawals via depositing and changing the Merkle Root, the contract stores 100 recent Merkle Roots.

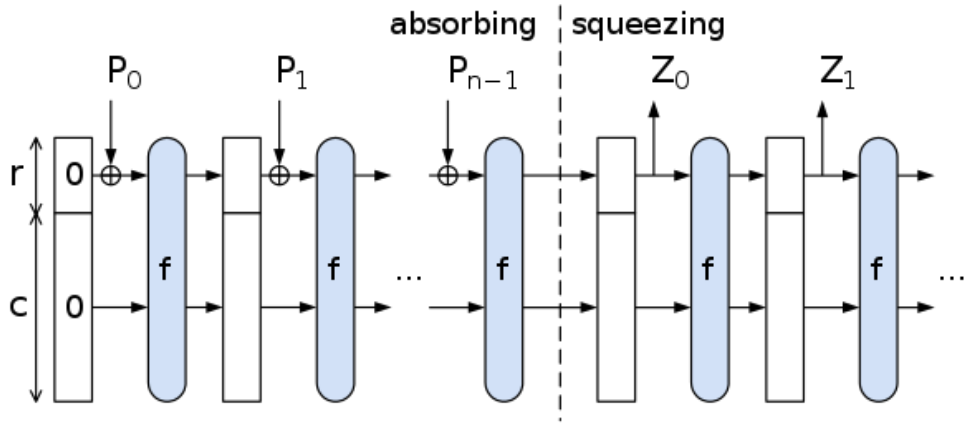
Tornado Cash : Choosing the Hash

H_1 is the Pedersen hash function (also used by dYdX for signatures!)
H_2 is the MiMC hash function via sponge + Feistel mode.

ABDK, the auditor, notes that H_1 is unoptimized for zkSNARKs.
H_2, on the other hand, is a hash specifically designed for zkSNARKS.

Optional : MiMC Hash

Sponge Construction is a method of creating a secure hash via pseudoranom permutation. It's also used by keccak256.



We use this with the MiMC permutation, which performs

$$(x, y) \rightarrow (y, x + (y + RK)^5)$$

sufficient times - this is where the name *Feistel* mode comes in.

Conclusion : Security

- No double spending (i.e. double withdrawals) is possible
- You must know both k, r to withdraw coins
- The proof is in ZK, so anonymity is preserved
- The proof is binding, so no frontrunning is possible
- 126-bit security, except the bn128 curve is now 100-bit secure

Tornado Cash Tokenomics

TORN token is currently around 43 dollars (listed on Binance)

- 10M TORN in total
- 0.5M TORN : Airdrop to early users for ETH pools
- 1M TORN : Anonymity Mining for ETH pools (vesting 1 year)
- 5.5M TORN : DAO treasury (vesting 5 years + 3 month cliff)
- 3M TORN : Developers, Supporters (vesting 3 years + 1 year cliff)

Tutela : Deanonymizing Tornado Cash

Dataset

- Labeled ETH Addresses via **Kaggle**
- Tornado Cash Router's **past tx data** to find depositors and recipients.

Heuristic

- **Deposit Address Reuse** : utilize CEX address
- **Learned Node Embedding** : embedd addresses via interactions
- **Anonymity Score** : via clusterization
- **Address Match** : i.e. depositor equals recipient
- **Unique Gas Price** : unique user-specified gas prices
- **Multiple Denominations** : using multiple pools to transfer
- **Anonymity Mining** : search based on Anonymity Points

Table of Contents

References

- Ethereum Yellowpaper
- Tornado Cash Whitepaper
- Tornado Cash Core on Github
- Useful Explanation on MiMC

References

- A Graduate Course in Applied Cryptography (Boneh, Shoup)
- On the Size of Pairing-based Non-interactive Arguments [Groth16]
- Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model [BGM17]
- MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity [AGR+16]
- Tutela: An Open-Source Tool for Assessing User-Privacy on Ethereum and Tornado Cash : `https://arxiv.org/pdf/2201.06811.pdf`