Blockchain Turbulences: From Trust to Censorship Resistance

Davor Svetinovic

Khalifa University, EECS, Abu Dhabi UAE

and Vienna University of Economics and Business, Austria

Director/Head Research Institute for Cryptoeconomics

Outline

- Welcome and introduction to the presentation
- Overview of our current work
- Importance of trust and censorship resistance in blockchain systems
- Combination of blockchain, AI, and IoT

All credit goes to amazing students and collaborators who contributed to the work discussed in the presentation!

Improving Cryptocurrency Crime Detection: CoinJoin Community Detection Approach

Publisher: IEEE

Anton Wahrstätter ; Jorão Gomes ; Sajjad Khan ; Davor Svetinovic

All Authors

Abstract:
The potential of Bitcoin for money laundering and terrorist financing represents a significant challenge in law enforcement. In recent years, the use of privacy-improving CoinJoin transactions has grown significantly and helped criminal actors obfuscate Bitcoin money flows. In this study, we use unsupervised machine learning to analyze the complete Bitcoin user graph in order to identify suspicious actors potentially involved in illegal activities. In contrast to the existing studies, we introduce a novel set of features that we use to identify potential criminal activity more accurately. Furthermore, we apply our clustering algorithm to a CoinJoin-adjusted variant of the Bitcoin user graph, which enables us to analyze the network at a more detailed, user-centric level while still offering opportunities to address advanced privacy-enhancing techniques at a later stage. By comparing the results with our ground truth data set, we find that our improved clustering method is able to capture significantly more illicit activity within the most suspicious clusters. Finally, we find that users associated with illegal activities commonly have significant short paths to CoinJoin wallets and show tendencies toward outlier behavior. Our results have potential contributions to anti-money laundering efforts and combating the financing of terrorism and other illegal activities.

Published in: IEEE Transactions on Dependable and Secure Computing (Early Access)

Page(s): 1 - 11

DOI: 10.1109/TDSC.2023.3238412

Date of Publication: 20 January 2023

Publisher: IEEE

TL;DR

- Using information about CoinJoins to identify criminal entities more accurately on Bitcoin

Parse local data of Full Node

- **Set up Full Node using Bitcoin-core Client**
- **Parse the blk*****.dat files for transactions using forked parser**
  - Sample block of 1st blk-file (Genesis block):


```
f9beb4d91d010000010000000000000000000000000000000000000000000000000000000000000000000000003ba3edfd7a7b12b27ac72c3e67768f617f
c81bc3888a51323a9fb8aa4b1e5e4a29ab5f49ffff001d1dac2b7c01010000000100000000000000000000000000000000000000000000000000000000
00000000ffffffff4d04ffff001d0104455468652054696d65732030332f4a616e2f32303039204368616e63656c6c6f72206f6e206272696e6b206f66
207365636f6e64206261696c6f757420666f722062616e6b73ffffffff0100f2052a01000000434104678afdb0fe5548271967f1a67130b7105cd6a828
e03909a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5fac00000000
```

Data aquisition

- Full Node
  - Forked python-blockchain-parser.git ==> python-bitcoin-graph.git
    - https://github.com/Nerolation/python-bitcoin-graph (example)

```
(btc) nero@nero-ThinkPad-P53s:~/python/wu/btc/python-bitcoin-graph$ python3 run.py -loc "./.bitcoin/blocks" -collectvalue -collectblk  --help
usage: run.py [-h] [-sf STARTFILE] [-ef ENDFILE] [-st STARTTX] [-et ENDTX] [-ets ENDTS] [-loc BLKLOCATION] [-path TARGETPATH] [-collectvalue]
              [-bucket BUCKET] [-c CREDENTIALS] [-project PROJECT] [-ds DATASET] [-tid TABLEID]

optional arguments:
  -h, --help                                          show this help message and exit
  -sf STARTFILE, --startfile STARTFILE                .blk start file (included) - default: blk00000.dat
  -ef ENDFILE, --endfile ENDFILE                      .blk end file (excluded) - default: None
  -st STARTTX, --starttx STARTTX                      start transaction id (included) - default: None
  -et ENDTX, --endtx ENDTX                            end transaction id (excluded) - default: None
  -ets ENDTS, --endts ENDTS                           end timestamp of block - default: None
  -loc BLKLOCATION, --blklocation BLKLOCATION         .blk|.csv file location - default: ~/.bitcoin/blocks
  -path TARGETPATH, --targetpath TARGETPATH           path to store raw edges locally - default: ./
  -collectvalue, --collectvalue                       collect output values - default: No
  -collectblk, --collectblk                           collect blk file numbers with every edge - default: No
  -upload, --upload                                   upload edges to google bigquery - default: False
  -parquet, --parquet                                 use parquet format - default: False
  -mp, --multiprocessing                              use multiprocessing - default: False
  -ut UPLOADTHRESHOLD, --uploadthreshold UPLOADTHRESHOLD  uploading threshold for parquet files - default: 5
  -bucket BUCKET, --bucket BUCKET                     bucket name to store parquet files - default: btc_<timestamp>
  -c CREDENTIALS, --credentials CREDENTIALS           path to google credentials (.*json)- default: ./.gcpkey/.*json
  -project PROJECT, --project PROJECT                 google cloud project name - default: btcgraph
  -ds DATASET, --dataset DATASET                      bigquery data set name - default: btc
  -tid TABLEID, --tableid TABLEID                     bigquery table id - default: bitcoin_transactions
```

Data acquisition – Google BigQuery

Table schema

Filter Enter property name or value

Field name	Type	Mode
ts	TIMESTAMP	NULLABLE
txhash	STRING	NULLABLE
input_txhash	STRING	NULLABLE
vout	INTEGER	NULLABLE
output_to	STRING	NULLABLE
output_index	INTEGER	NULLABLE
value	INTEGER	NULLABLE
blk_file_nr	INTEGER	NULLABLE

Row	ts	txhash
1	2009-01-12 02:30:25 UTC	f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16
2	2009-01-12 06:16:40 UTC	4385fcf8b14497d0659adccfe06ae7e38e0b5dc95ff8a13d7c62035994a0cd79
3	2009-01-12 13:21:00 UTC	298ca2045d174f8a158961806ffc4ef96fad02d71a6b84d9fa0491813a776160

input_txhash	vout
0437cd7f8525ceed2324359c2d0ba26006d92d856a9c20fa0241106ee5a597c9	0
12b5633bad1f9c167d523ad1aa1947b2732a865bf5414eab2f9e5ae5d5c191ba	0
591e91f809d716912ca1d4a9295e70c3e78bab077683f79350f101da64588073	0

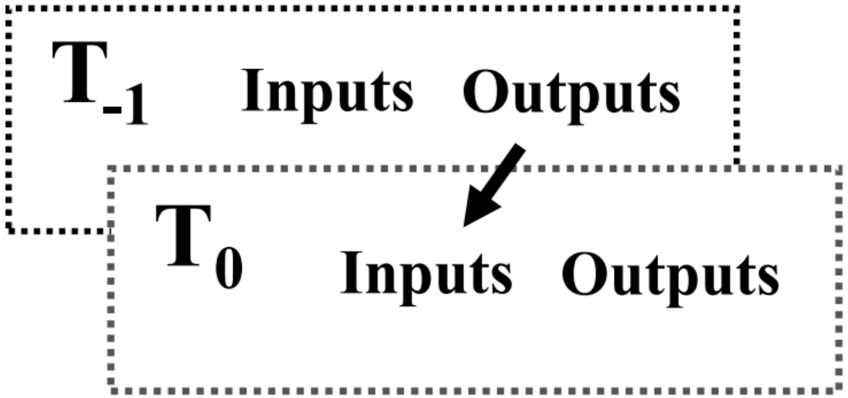
output_to	output_index	value	blk_file_nr
1Q2TWHE3GMdB6BZKafqwxXtWAWgFt5Jvm3	0	1000000000	0
15NUwyBYrZcnUgTagsm1A7M2yL2GntpuaZ	0	100000000	0
1BDvQZjaAJH4ecZ8aL3fYgTi7rnn3o2thE	0	100000000	0

Data preprocessing – Google BigQuery

Input/Output Mapping:

Input: Non-mapped Transaction Graph
$E_{RAW} = \{(TxID,\ IN_{TxID},\ IN_{Vout},\ OUT_{Address}$ and $OUT_{Index}),\ldots\}$
Output: Set of edges $E = \{\ (IN_{Address}, OUT_{Address}),\ldots\}$

$edges\ E \leftarrow empty\ set\ \emptyset$
for all $e \in E_{RAW}$ **do**
  $e_{t-1} \leftarrow \{e_i \ldots e_j\} \mid IN_{TxID}(e) = TxID(e_i)$
  $utxo \leftarrow e \mid IN_{Vout}(e) = OUT_{Index}(e_{t-1})$
  $E \mathrel{+}= \{OUT_{Address}(uxto), OUT_{Address}(e)\}$
end for
return E



Data preprocessing – Google BigQuery

CoinJoin detection (Wasabi):

- Block heigth 530,500 – 609,999
  - 2 static coordinator addresses
- Since block height 610,000
  - Tx Output Values contain at least 10 equal values
  - Most frequent output value equals 0.1 BTC ± 0.02 BTC
  - More Inputs than Outpus of equal values
  - One unique Output Value
  - 3 distinct Output Values
  - Bech32 addresses
  - NOT IF:
    - Input Addresses != Output Addresses
    - Output Values between 0.08-0.085, 0.115-0.12 or exactly 0.09 or 0.1

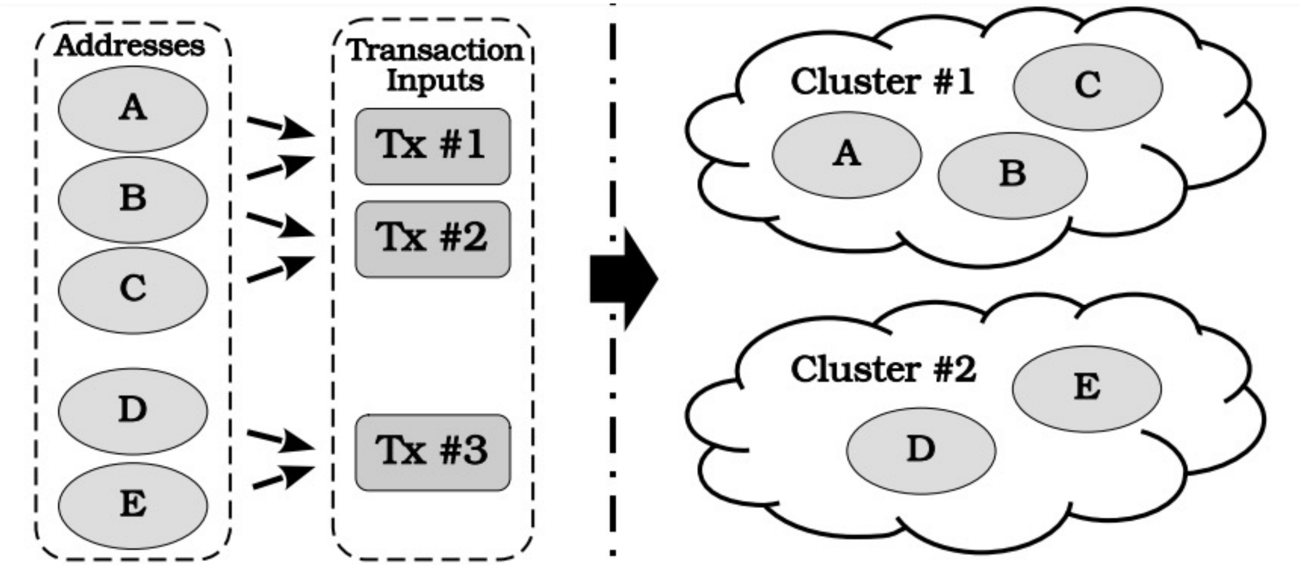
Data preprocessing – Google BigQuery

CoinJoin detection (Samurai):

- Since block height 570,000:
  - 5 Inputs
  - 5 Outputs
  - At least 1 and at most 3 equal Input Values that match a pool size – the rest is in the range poolsize ± 0.0011 BTC

- Pool sizes:
  - 0.001 BTC
  - 0.01 BTC
  - 0.05 BTC
  - 0.5 BTC

Clustering – Google BigQuery



K-Means Analysis

Dividing users into 100 k-means clusters using the follwoing features:

Turnover-features:

- Total amount received,
- Total amount sent,
- Avg. amount received,
- Avg. amount sent

Connectivity-features:

- In-degree,
- Out-degree,
- Total-degree,
- Shortest-Path to CoinJoin transaction

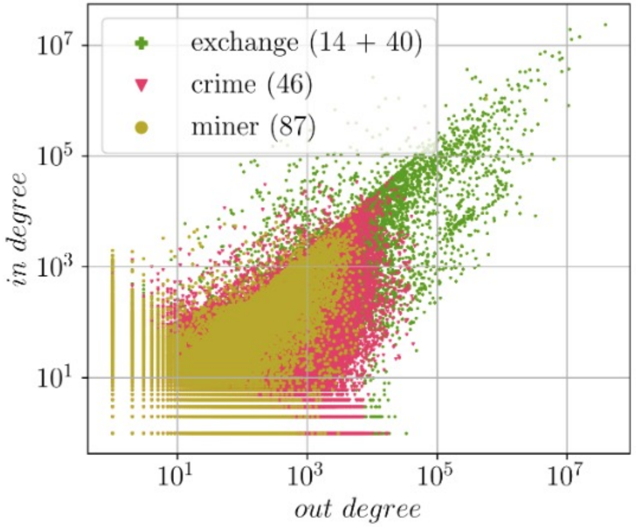
Activity-features:

- Activity period in days,
- avg. hour of activity,
- degree per active day

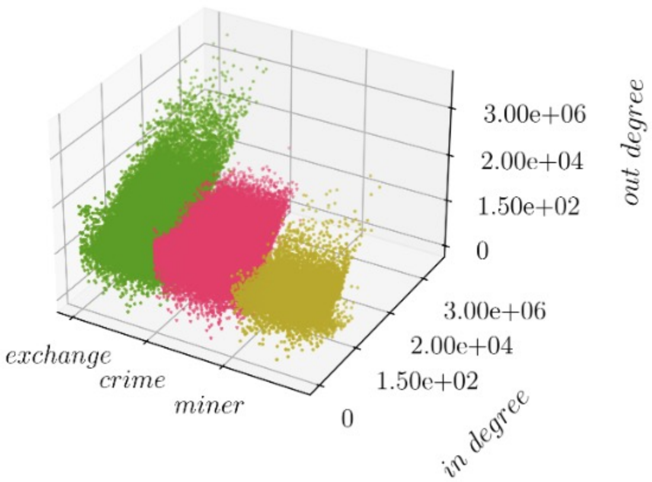
Utxos-specific-features:

- Avg. age of Utxos in days,
- final balance in BTC

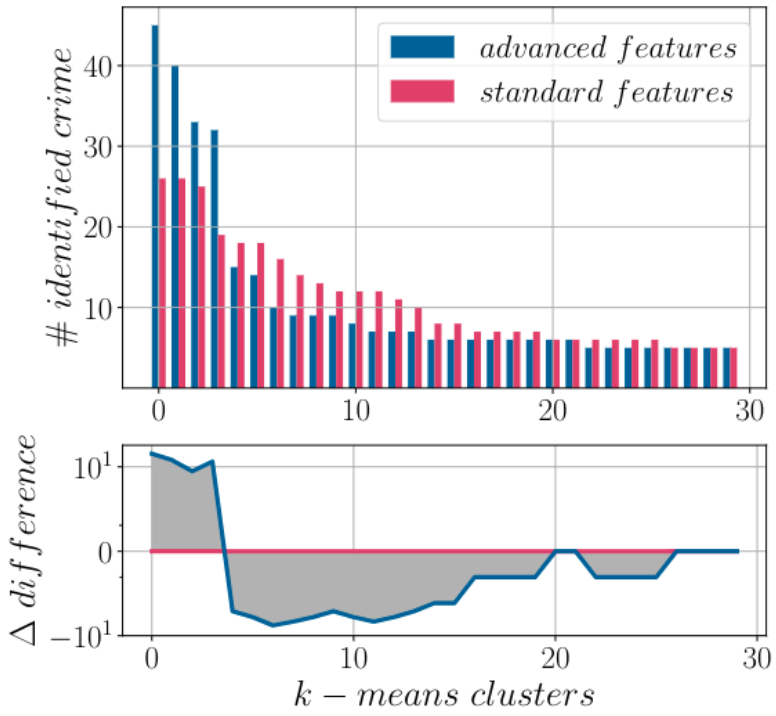
Analysis



exchange (14 + 40)
crime (46)
miner (87)
in degree
out degree
10^1
10^3
10^5
10^7



exchange
crime
miner
out degree
in degree
3.00e+06
2.00e+04
1.50e+02
0



advanced features
standard features
identified crime
Δ difference
k − means clusters

Analysis

TABLE 2

K-Means Cluster ids with the most identified users of the categories *crime* (46-39), *exchange* (14-93) and *miner* (7-87). The *identified* measure tells how many services/entities of the respective category we were able to identify from the ground truth data set. For example, within the cluster with the id 46 we identified 45 different services that were labeled *crime*.

id	out degree	in degree	total degree	active days	degree/active day	total amount received	avg amount received
46	126	31	157	34	2.30	1,341,893,825	385,899,462
4	30	22	52	11	2.34	4,608,143,763	700,231,216
13	17	11	28	123	1.89	882,592,602	554,798,252
39	5	5	10	9	2.28	1,074,169,522	731,787,073
14	623	366	989	19	2.68	306,490,213,308	806,156,543
40	347	270	617	45	7.33	227,735,226,561	58,337,174,815
93	7	17	24	4	15.30	843,569,064	617,765,779
87	6	6	11	8	2.72	7,931,131,789	6,295,672,191
49	5	5	10	7	2.84	3,896,333,630	2,891,014,513
7	3	3	6	5	2.61	1,209,110,525	1,003,317,080

TABLE 3

crime (46-39), *exchange* (14-93) and *miner* (7-87).

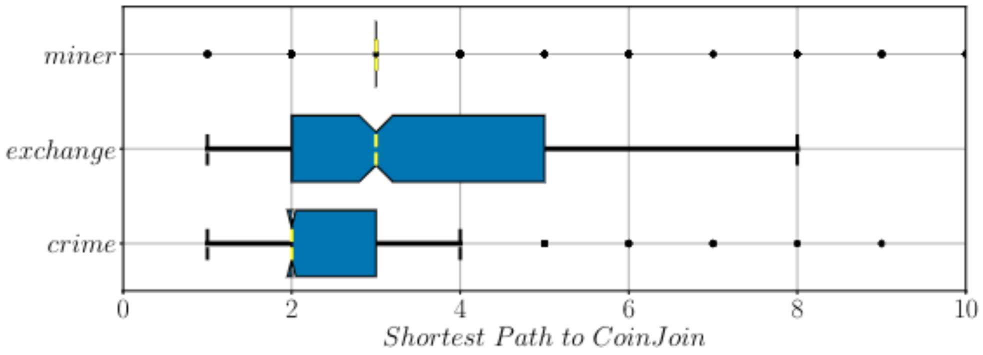
id	total amount sent	avg amount sent	utc_{active}	final balance	utxo age	shortest path to CJ	identified	cluster size
46	1,031,250,101	230,751,349	12	2,461,562	43	2.78	45	1,267,320
4	4,646,407,102	348,898,093	12	4,370,580	215	3.57	33	8,341,798
13	876,187,417	295,531,546	12	7,947,454	62	3.06	32	3,698,687
39	1,077,985,797	370,146,805	12	3,449,235	255	3.65	15	7,237,207
14	305,526,521,897	398,150,196	12	291,499,451	219	3.52	35	1,392,551
40	226,333,624,912	31,084,918,008	12	2,001,274	126	3.26	31	358,716
93	954,548,147	294,456,527	12	10,738,269	59	3.03	7	3,591,987
87	8,147,243,858	3,381,518,584	12	2,276,419	102	3.34	22	2,786,947
49	3,875,128,428	1,545,644,518	12	9,229,881	98	3.33	13	1,994,675
7	1,207,236,399	517,160,946	12	7,232,830	86	3.33	9	4,909,275

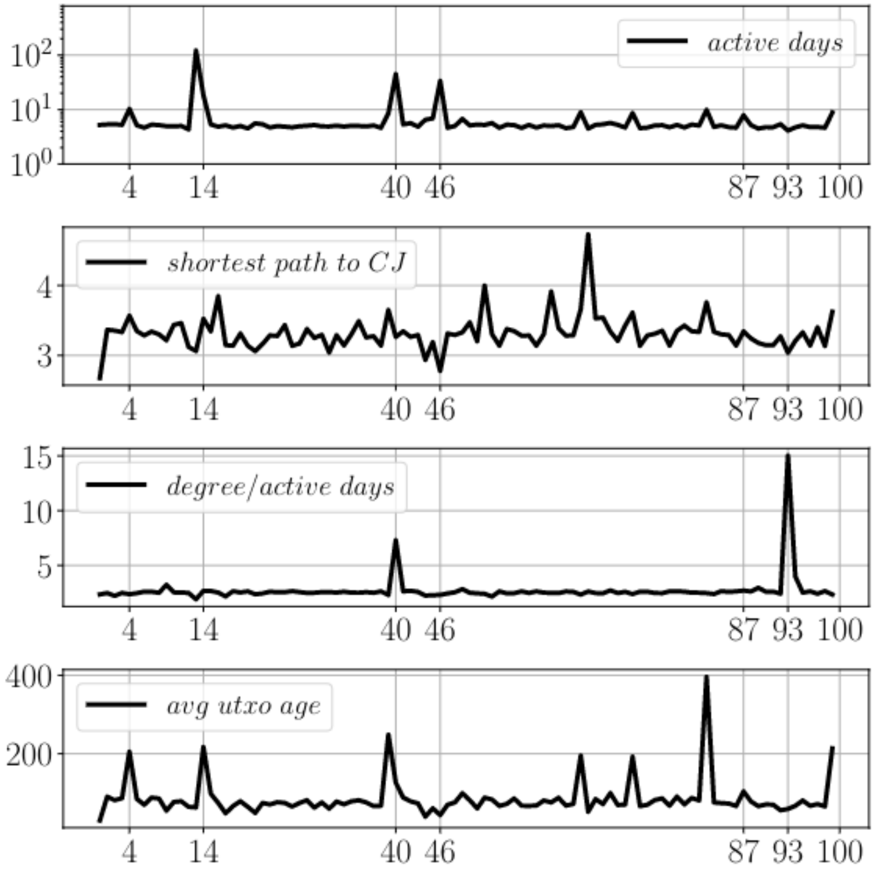
Analysis

Findings:

The shortest path to CJ feature improves unsupervised clustering --> We concentrated more criminal entities within a few clusters

Criminal entities, in general, tend towards outlier bahviour. (degree/active-days)





Unlinkable Payments:

The beauty of stealth addresses

Paper working title:
“ModSAP - A Composition of Modular Stealth Address Protocols on Public Blockchain” by Toni Wahrstätter et al.
Planned submission venue (May 2023):
IEEE Transactions on Information Forensics and Security

eip	title	description	author	discussions-to	status	type	category	created
5564	Stealth Addresses	Private, non-interactive transfers and interactions	Toni Wahrstätter (@nerolation), Matt Solomon (@mds1), Ben DiFrancesco (@apbendi), Vitalik Buterin (@vbuterin)	https://ethereum-magicians.org/t/eip-5566-stealth-addresses-for-smart-contract-wallets/10614	Draft	Standards Track	ERC	2022-08-13

Abstract

This specification defines a standardized way of creating stealth addresses. This EIP enables senders of transactions/transfers to non-interactively generate private stealth addresses for their recipients that only the recipients can unlock.

Motivation

The standardization of non-interactive stealth address generation holds the potential to greatly enhance the privacy capabilities of Ethereum by enabling the recipient of a transfer to remain anonymous when receiving an asset. This is achieved through the generation of a stealth address by the sender, using a shared secret between the sender and recipient. Only the recipient is able to unlock the funds at the stealth address, as they are the only ones with access to the private key required for this purpose. As a result, observers are unable to link the recipient's stealth address to their identity, preserving the privacy of the recipient and leaving only the sender with this information.

Specification

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Definitions:

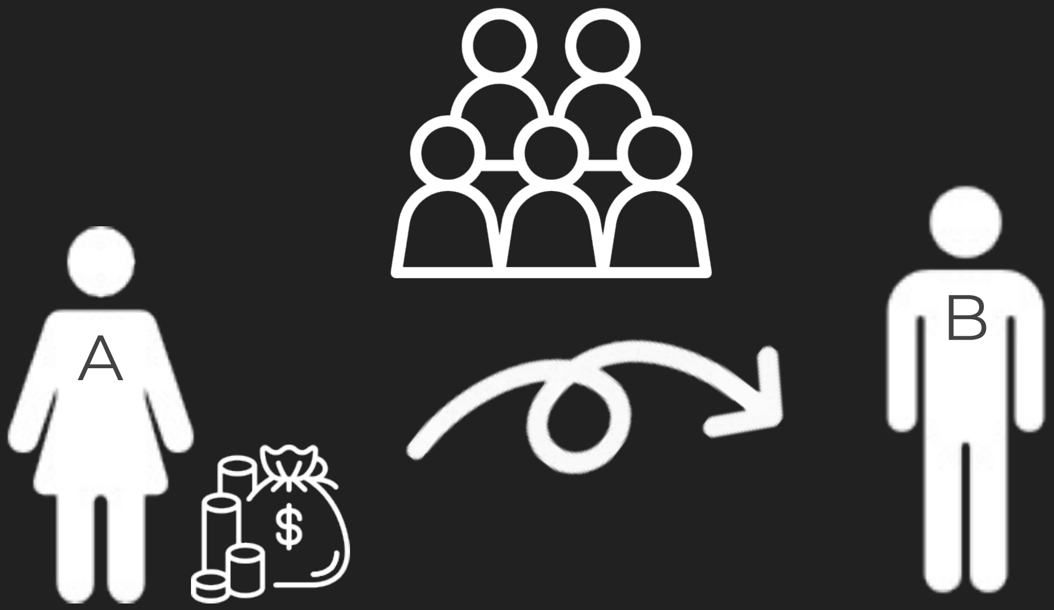
- A "stealth meta-address" is a set of one or two public keys that can be used to compute a stealth address for a given recipient.
- A "spending key" is a private key that can be used to spend funds sent to a stealth address. A "spending public key" is the corresponding

Problem definition…

Problem definition…

Problem definition…

Problem definition…

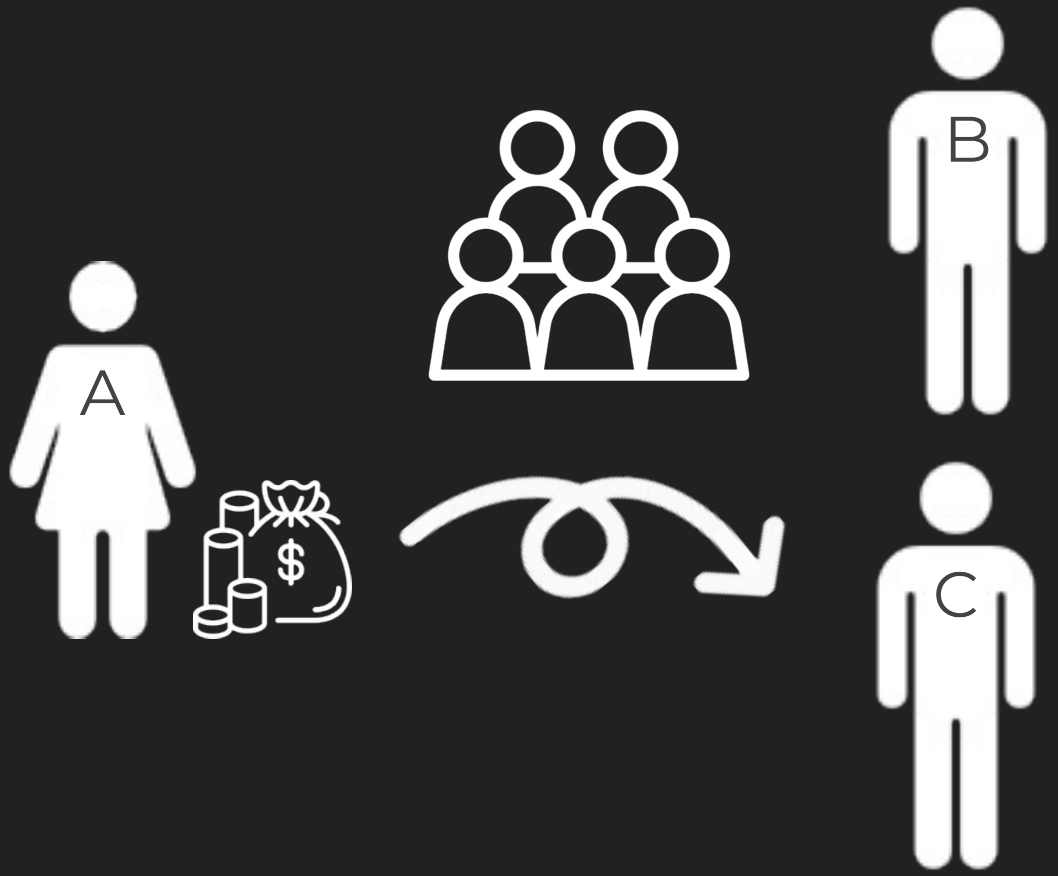


Problem definition…

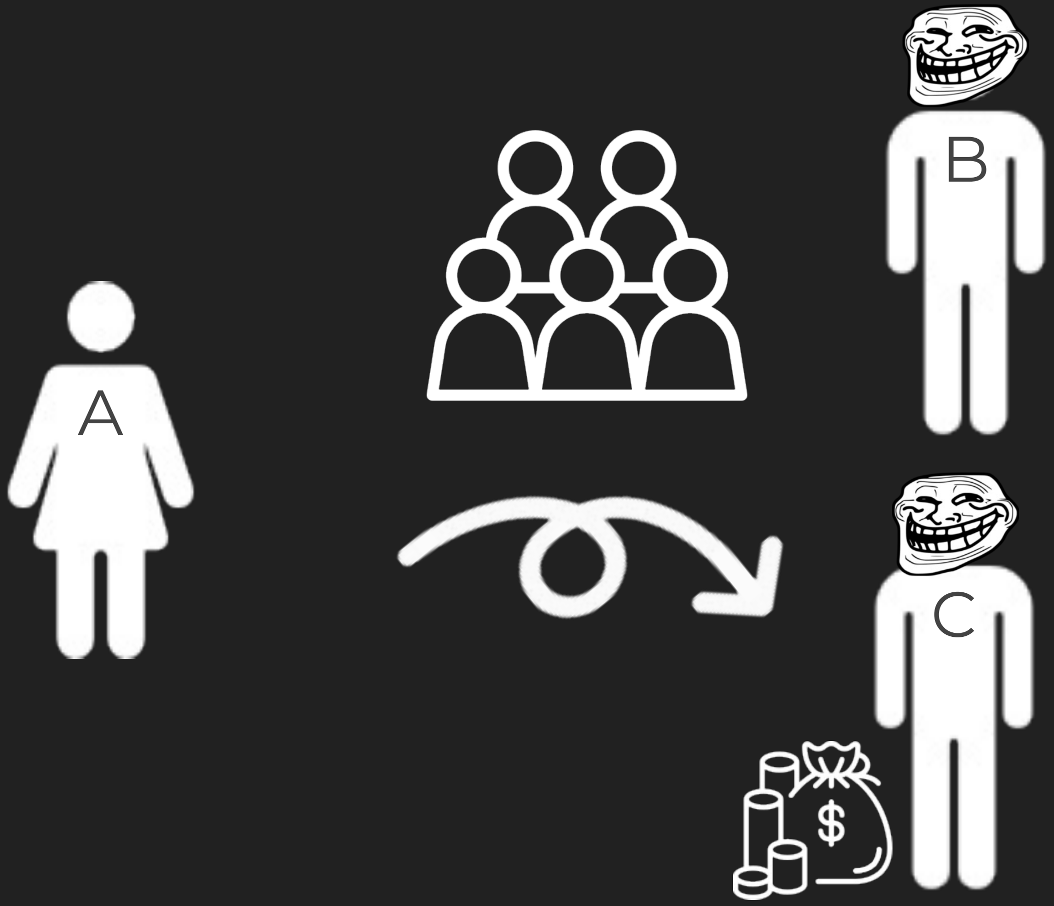
Problem definition…

Problem definition…

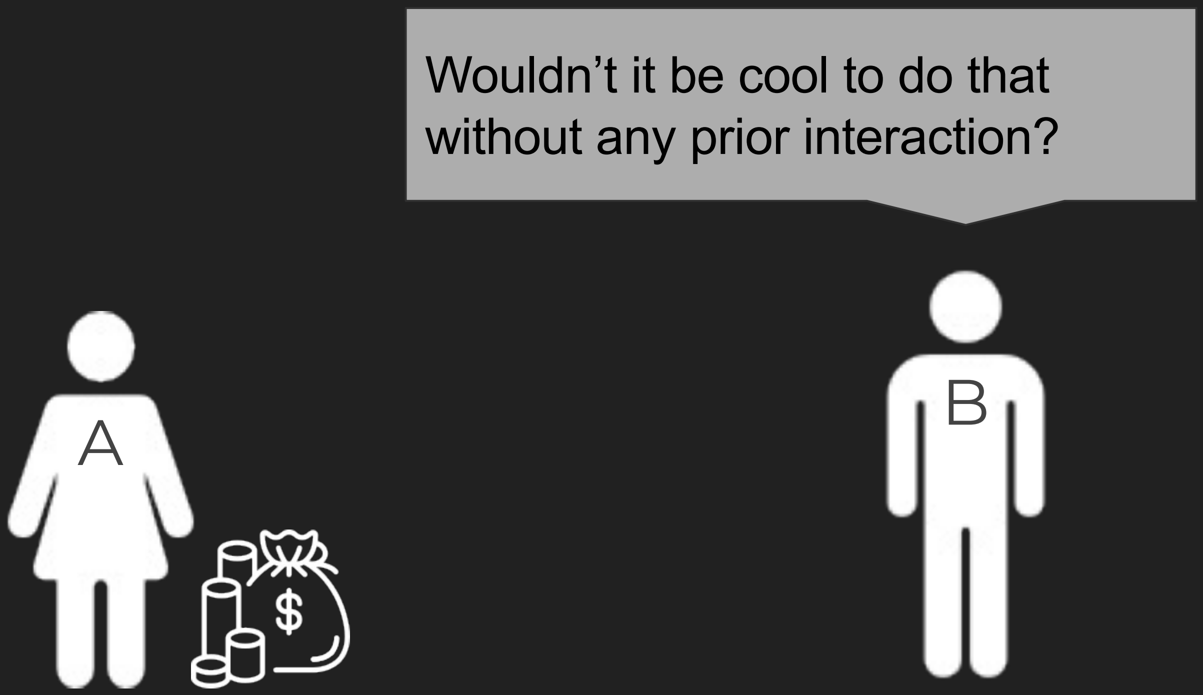
Problem definition…



Problem definition…

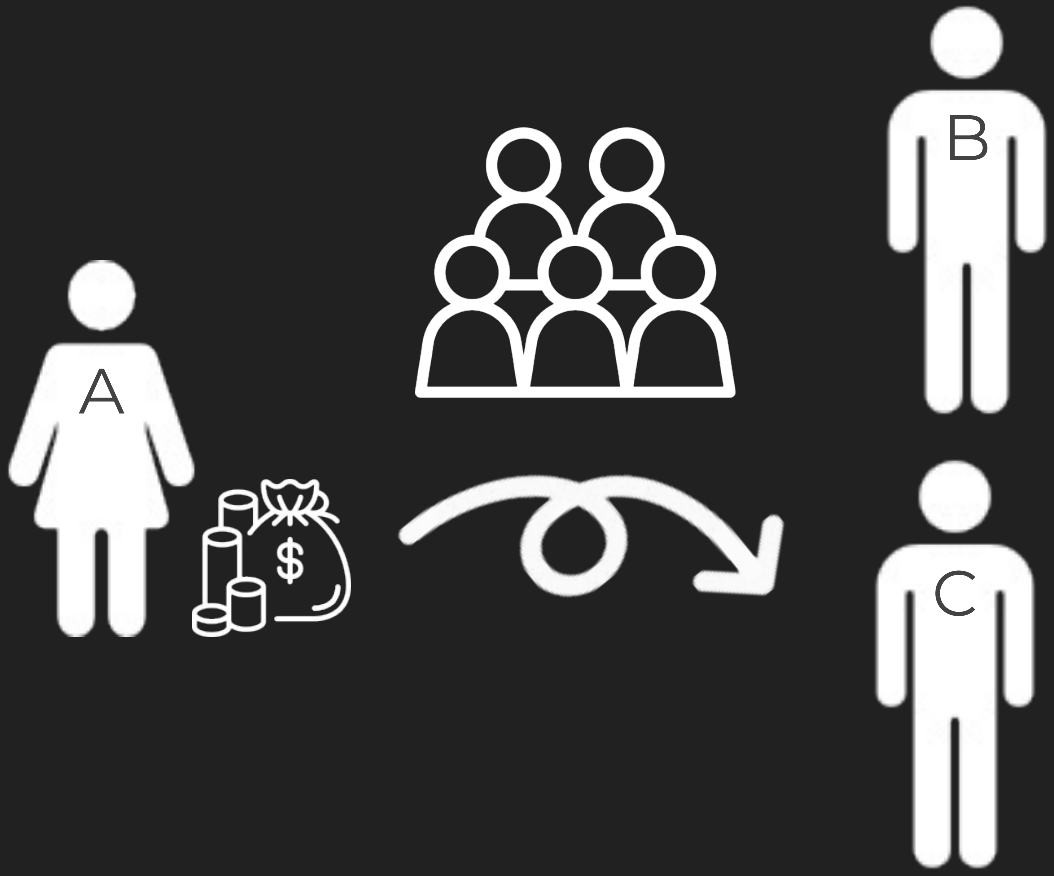


But wait…


Wouldn't it be cool to do that without any prior interaction?
A
B

Stealth Addresses

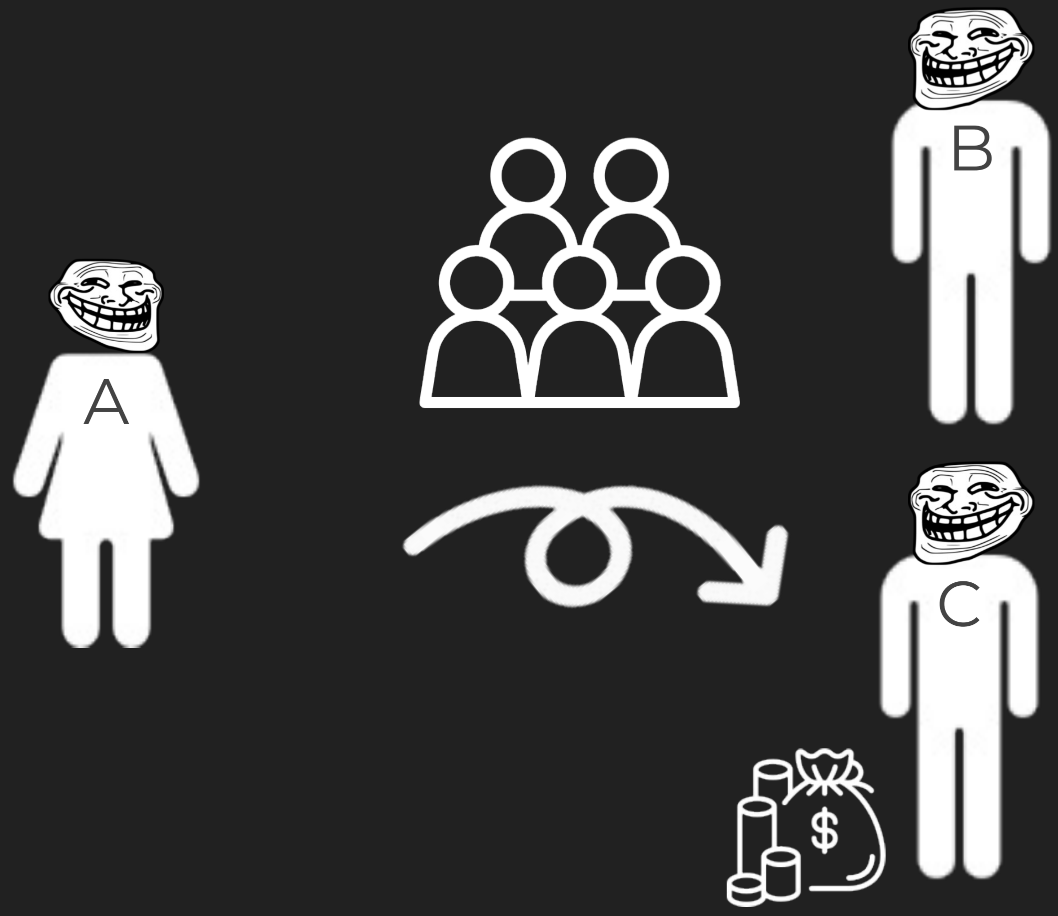
- Alice (non-interactively) generates a stealth address for Bob.
- Alice sends to that stealth address.
- Bob can access the stealth address.
- Looks like Alice sent to some random address.



Stealth Addresses

- Alice (non-interactively) generates a stealth address for Bob.
- Alice sends to that stealth address.
- Bob can access the stealth address.
- Looks like Alice sent to some random address.

We get unlikability, but NOT untraceability.



Stealth Addresses

Where are we today and how we got here?

From the beginning…

2008

- Bitcoin Whitepaper (pseudonymous accounts)

2011

Apr. 2011

- Launch of first mixers:
  - BitcoinFog
  - BitLaundry
  - Blockchain.info' Shared Send,

  Bitansky et al.: SNARKS
- User `Bytecoin` introduces Stealth Addresses

2013

- Peter Todd: Bitcoin CoinJoins, Parno et al.: Pinocchio

Oct. 2013

- Nicolas van Saberhagen: CryptoNote v. 2.0

2014

- Monero Launch, Ben-Sasson et al.: Zerocash

2015

- CoinJoins:
  - Dark Wallet
  - Samourai Whirlpool
  - JoinMarket
  - Tumblebit

2016

- Electric Coin Company: ZCash

2017

- Adam Ficsor: ZeroLink protocol, Bünz et al.: Bulletproofs

2018

- Wasabi CoinJoin Wallet

2019

- Semenov, Pertsev & Storm: Tornado Cash, Bünz et al.: Zether, Starkware: zk-STARKs, Bowe et al.: HALO

2020

- Aztech 1, ScopeLift's Umbra, Secret Network

2021

- Aztech Connect, zkSync, Railgun

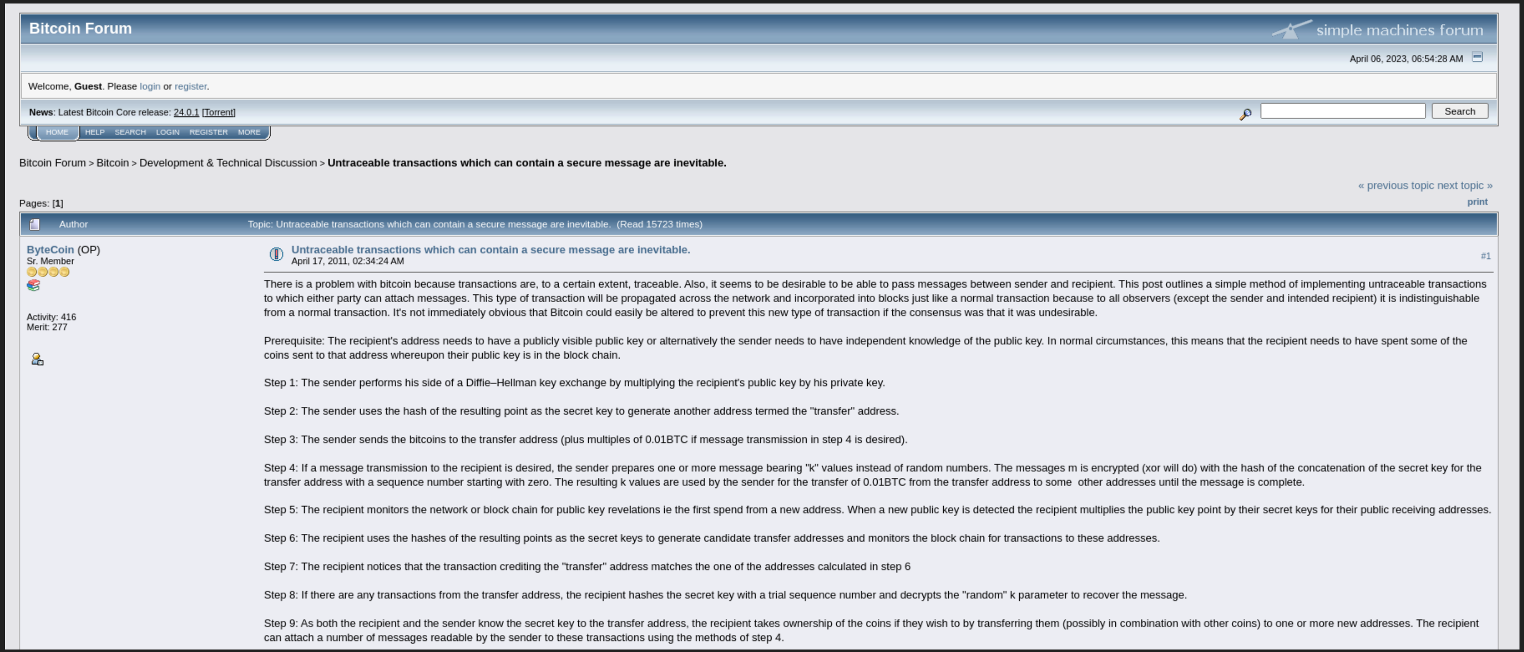
2022

- zk-EVMs, EIP-5564: Stealth Addresses

2023

- Privacy Pools, HOPR Protocol

From the beginning…



Bitcoin Forum — simple machines forum

April 06, 2023, 06:54:28 AM

Welcome, Guest. Please login or register.

News: Latest Bitcoin Core release: 24.0.1 [Torrent]

HOME HELP SEARCH LOGIN REGISTER MORE

Bitcoin Forum > Bitcoin > Development & Technical Discussion > **Untraceable transactions which can contain a secure message are inevitable.**

« previous topic next topic »

Pages: [1] print

Author | Topic: Untraceable transactions which can contain a secure message are inevitable. (Read 15723 times)

ByteCoin (OP)
Sr. Member
Activity: 416
Merit: 277

Untraceable transactions which can contain a secure message are inevitable.
April 17, 2011, 02:34:24 AM #1

There is a problem with bitcoin because transactions are, to a certain extent, traceable. Also, it seems to be desirable to be able to pass messages between sender and recipient. This post outlines a simple method of implementing untraceable transactions to which either party can attach messages. This type of transaction will be propagated across the network and incorporated into blocks just like a normal transaction because to all observers (except the sender and intended recipient) it is indistinguishable from a normal transaction. It's not immediately obvious that Bitcoin could easily be altered to prevent this new type of transaction if the consensus was that it was undesirable.

Prerequisite: The recipient's address needs to have a publicly visible public key or alternatively the sender needs to have independent knowledge of the public key. In normal circumstances, this means that the recipient needs to have spent some of the coins sent to that address whereupon their public key is in the block chain.

Step 1: The sender performs his side of a Diffie–Hellman key exchange by multiplying the recipient's public key by his private key.

Step 2: The sender uses the hash of the resulting point as the secret key to generate another address termed the "transfer" address.

Step 3: The sender sends the bitcoins to the transfer address (plus multiples of 0.01BTC if message transmission in step 4 is desired).

Step 4: If a message transmission to the recipient is desired, the sender prepares one or more message bearing "k" values instead of random numbers. The messages m is encrypted (xor will do) with the hash of the concatenation of the secret key for the transfer address with a sequence number starting with zero. The resulting k values are used by the sender for the transfer of 0.01BTC from the transfer address to some other addresses until the message is complete.

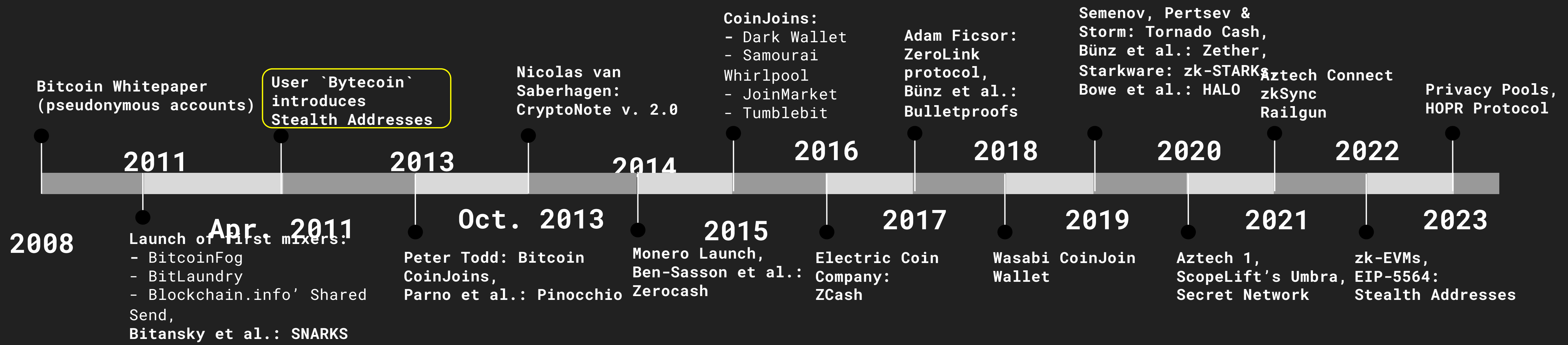
Step 5: The recipient monitors the network or block chain for public key revelations ie the first spend from a new address. When a new public key is detected the recipient multiplies the public key point by their secret keys for their public receiving addresses.

Step 6: The recipient uses the hashes of the resulting points as the secret keys to generate candidate transfer addresses and monitors the block chain for transactions to these addresses.

Step 7: The recipient notices that the transaction crediting the "transfer" address matches the one of the addresses calculated in step 6

Step 8: If there are any transactions from the transfer address, the recipient hashes the secret key with a trial sequence number and decrypts the "random" k parameter to recover the message.

Step 9: As both the recipient and the sender know the secret key to the transfer address, the recipient takes ownership of the coins if they wish to by transferring them (possibly in combination with other coins) to one or more new addresses. The recipient can attach a number of messages readable by the sender to these transactions using the methods of step 4.



From the beginning…

[Bitcoin-development] Stealth Addresses

Peter Todd | Mon, 06 Jan 2014 04:06:05 -0800

```
* Abstract

A Stealth Address is a new type of Bitcoin address and related
scriptPubKey/transaction generation scheme that allowers payees to
publish a single, fixed, address that payors can send funds efficiently,
privately, reliably and non-interactively. Payors do not learn what
other payments have been made to the stealth address, and third-parties
learn nothing at all. (both subject to an adjustable anonymity set)


* Acknowledgments

Credit goes to ByteCoin for the original idea.(1) Gregory Maxwell, Adam
Back, and others on #bitcoin-wizards contributed valuable input on the
implementation. Finally thanks goes to Amir Taaki for input on the
general idea of stealth addresses and use-cases.
```

The Mail Archive

Search bitcoin-development

- The Mail Archive home
- bitcoin-development - all messages
- bitcoin-development - about the list
- Expand
- Previous message
- Next message

2008 — Bitcoin Whitepaper (pseudonymous accounts)

2011 — Launch of first mixers:
- BitcoinFog
- BitLaundry
- Blockchain.info' Shared Send,

Bitansky et al.: SNARKS

Apr. 2011 — User `Bytecoin` introduces Stealth Addresses

2013 — Peter Todd: Bitcoin CoinJoins, Parno et al.: Pinocchio

Oct. 2013 — Nicolas van Saberhagen: CryptoNote v. 2.0

2014 — Monero Launch, Ben-Sasson et al.: Zerocash

2015 — CoinJoins:
- Dark Wallet
- Samourai Whirlpool
- JoinMarket
- Tumblebit

2016 — Electric Coin Company: ZCash

2017 — Adam Ficsor: ZeroLink protocol, Bünz et al.: Bulletproofs

2018 — Wasabi CoinJoin Wallet

2019 — Semenov, Pertsev & Storm: Tornado Cash, Bünz et al.: Zether, Starkware: zk-STARKs, Bowe et al.: HALO

2020 — Aztech 1, ScopeLift's Umbra, Secret Network

2021 — Aztech Connect, zkSync, Railgun

2022 — zk-EVMs, EIP-5564: Stealth Addresses

2023 — Privacy Pools, HOPR Protocol

From the beginning…

CryptoNote v 2.0

Nicolas van Saberhagen

October 17, 2013

1 Introduction

"Bitcoin" [1] has been a successful implementation of the concept of p2p electronic cash. Both professionals and the general public have come to appreciate the convenient combination of public transactions and proof-of-work as a trust model. Today, the user base of electronic cash is growing at a steady pace; customers are attracted to low fees and the anonymity provided by electronic cash and merchants value its predicted and decentralized emission. Bitcoin has effectively proved that electronic cash can be as simple as paper money and as convenient as credit cards.

Unfortunately, Bitcoin suffers from several deficiencies. For example, the system's distributed nature is inflexible, preventing the implementation of new features until almost all of the network users update their clients. Some critical flaws that cannot be fixed rapidly deter Bitcoin's widespread propagation. In such inflexible models, it is more efficient to roll-out a new project rather than perpetually fix the original project.

In this paper, we study and propose solutions to the main deficiencies of Bitcoin. We believe that a system taking into account the solutions we propose will lead to a healthy competition among different electronic cash systems. We also propose our own electronic cash, "CryptoNote", a name emphasizing the next breakthrough in electronic cash.

2008 — Bitcoin Whitepaper (pseudonymous accounts)

2011 — Launch of first mixers:
- BitcoinFog
- BitLaundry
- Blockchain.info' Shared Send,

Bitansky et al.: SNARKS

Apr. 2011 — User `Bytecoin` introduces Stealth Addresses

2013 — Peter Todd: Bitcoin CoinJoins, Parno et al.: Pinocchio

Oct. 2013 — Nicolas van Saberhagen: CryptoNote v. 2.0

2014 — Monero Launch, Ben-Sasson et al.: Zerocash

2015 — CoinJoins:
- Dark Wallet
- Samourai Whirlpool
- JoinMarket
- Tumblebit

2016 — Electric Coin Company: ZCash

2017 — Adam Ficsor: ZeroLink protocol, Bünz et al.: Bulletproofs

2018 — Wasabi CoinJoin Wallet

2019 — Semenov, Pertsev & Storm: Tornado Cash, Bünz et al.: Zether, Starkware: zk-STARKs, Bowe et al.: HALO

2020 — Aztech 1, ScopeLift's Umbra, Secret Network

2021 — Aztech Connect, zkSync, Railgun

2022 — zk-EVMs, EIP-5564: Stealth Addresses

2023 — Privacy Pools, HOPR Protocol

From the beginning…

2008 — Bitcoin Whitepaper (pseudonymous accounts)

2011 — Launch of first mixers:
- BitcoinFog
- BitLaundry
- Blockchain.info' Shared Send,

Bitansky et al.: SNARKS

Apr. 2011 — User `Bytecoin` introduces Stealth Addresses

2013 — Peter Todd: Bitcoin CoinJoins, Parno et al.: Pinocchio

Oct. 2013 — Nicolas van Saberhagen: CryptoNote v. 2.0

2014 — Monero Launch, Ben-Sasson et al.: Zerocash

2015 — CoinJoins:
- Dark Wallet
- Samourai Whirlpool
- JoinMarket
- Tumblebit

2016 — Electric Coin Company: ZCash

2017 — Adam Ficsor: ZeroLink protocol, Bünz et al.: Bulletproofs

2018 — Wasabi CoinJoin Wallet

2019 — Semenov, Pertsev & Storm: Tornado Cash, Bünz et al.: Zether, Starkware: zk-STARKs, Bowe et al.: HALO

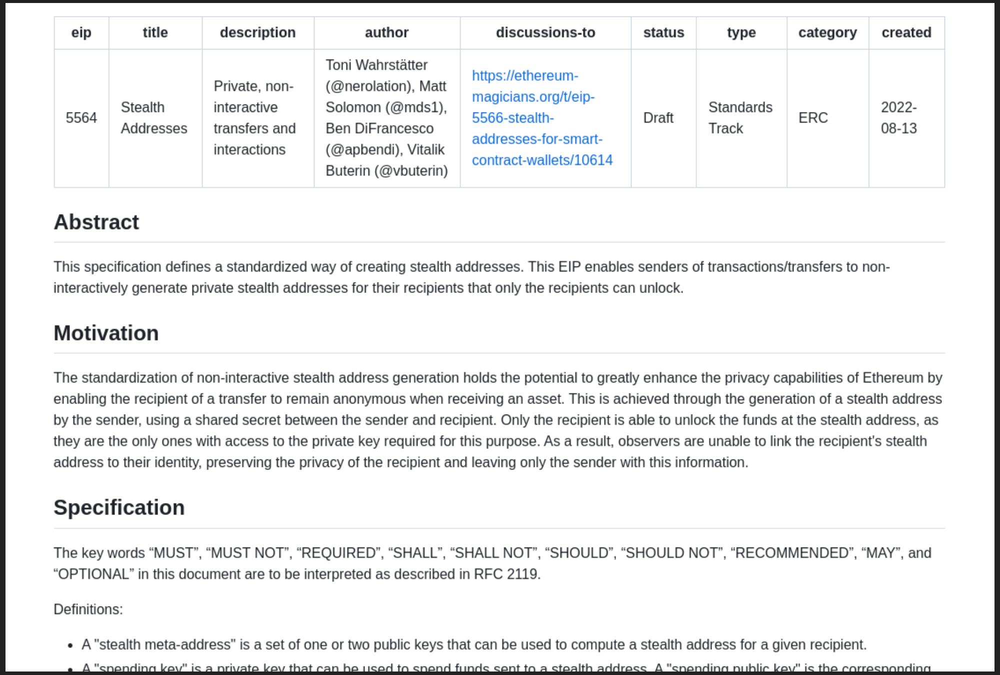
2020 — Aztech 1, ScopeLift's Umbra, Secret Network

2021 — Aztech Connect, zkSync, Railgun

2022 — zk-EVMs, EIP-5564: Stealth Addresses

2023 — Privacy Pools, HOPR Protocol

From the beginning…



eip	title	description	author	discussions-to	status	type	category	created
5564	Stealth Addresses	Private, non-interactive transfers and interactions	Toni Wahrstätter (@nerolation), Matt Solomon (@mds1), Ben DiFrancesco (@apbendi), Vitalik Buterin (@vbuterin)	https://ethereum-magicians.org/t/eip-5566-stealth-addresses-for-smart-contract-wallets/10614	Draft	Standards Track	ERC	2022-08-13

Abstract

This specification defines a standardized way of creating stealth addresses. This EIP enables senders of transactions/transfers to non-interactively generate private stealth addresses for their recipients that only the recipients can unlock.

Motivation

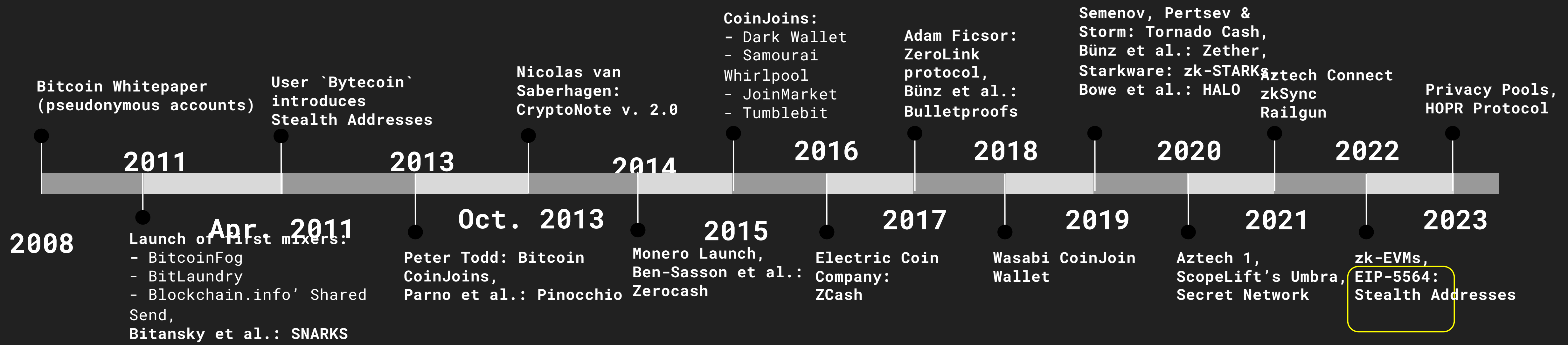
The standardization of non-interactive stealth address generation holds the potential to greatly enhance the privacy capabilities of Ethereum by enabling the recipient of a transfer to remain anonymous when receiving an asset. This is achieved through the generation of a stealth address by the sender, using a shared secret between the sender and recipient. Only the recipient is able to unlock the funds at the stealth address, as they are the only ones with access to the private key required for this purpose. As a result, observers are unable to link the recipient's stealth address to their identity, preserving the privacy of the recipient and leaving only the sender with this information.

Specification

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

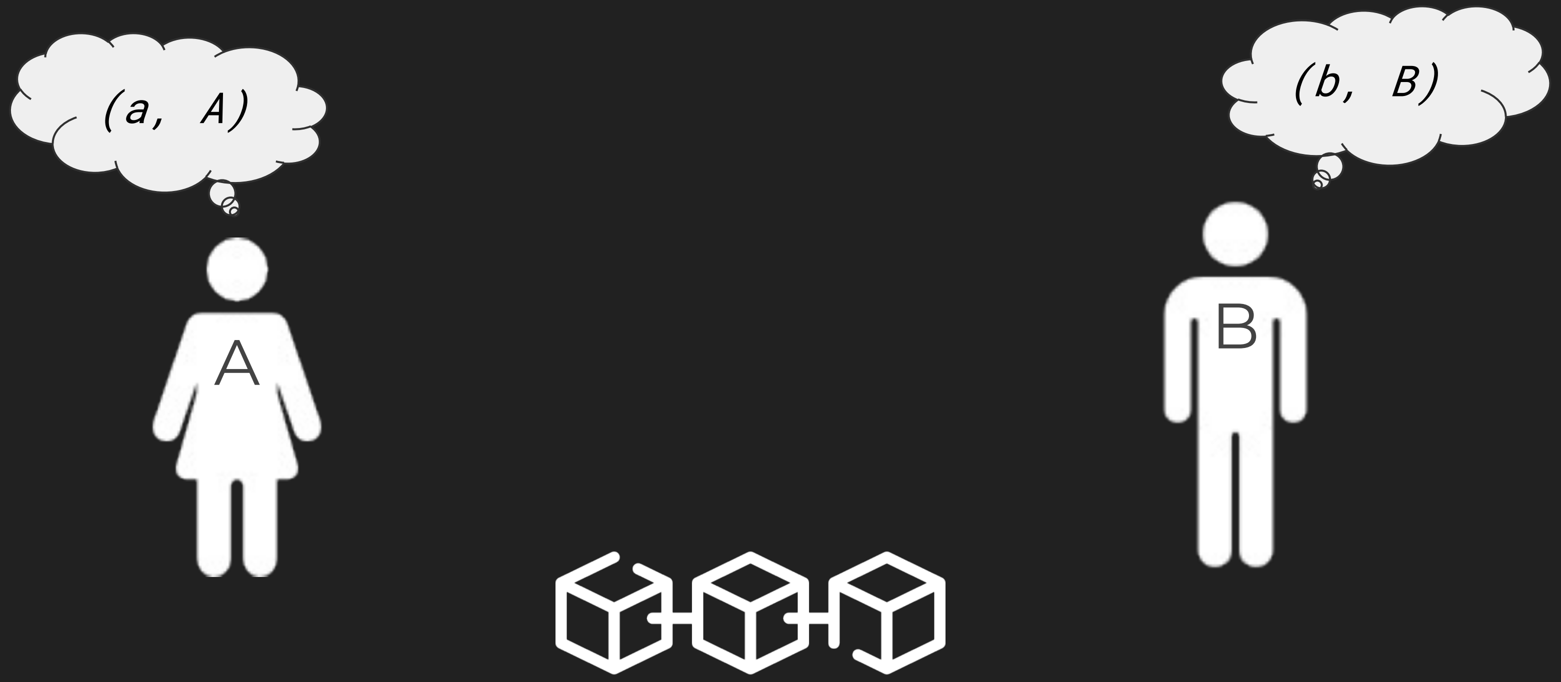
Definitions:

- A "stealth meta-address" is a set of one or two public keys that can be used to compute a stealth address for a given recipient.

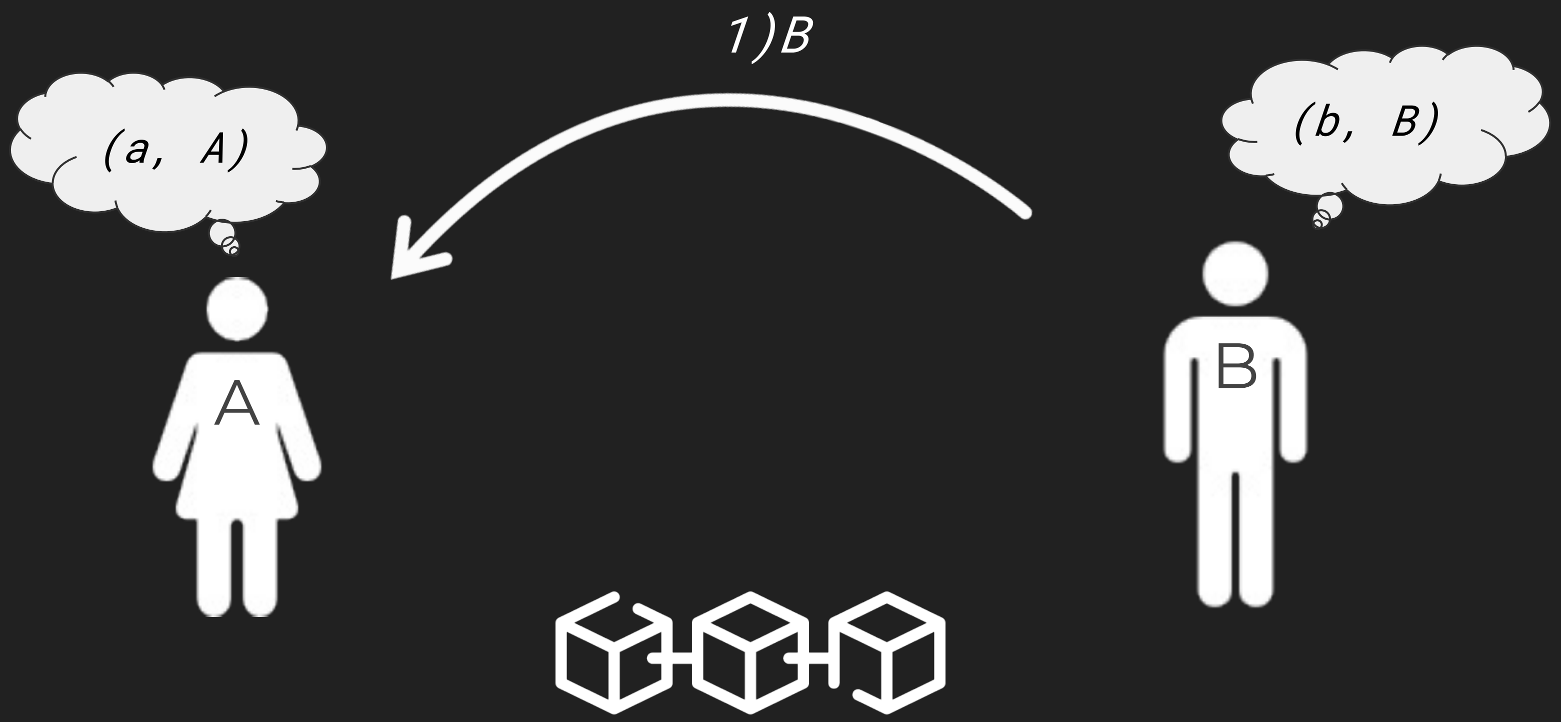


How does it work?

How does it work?



How does it work?

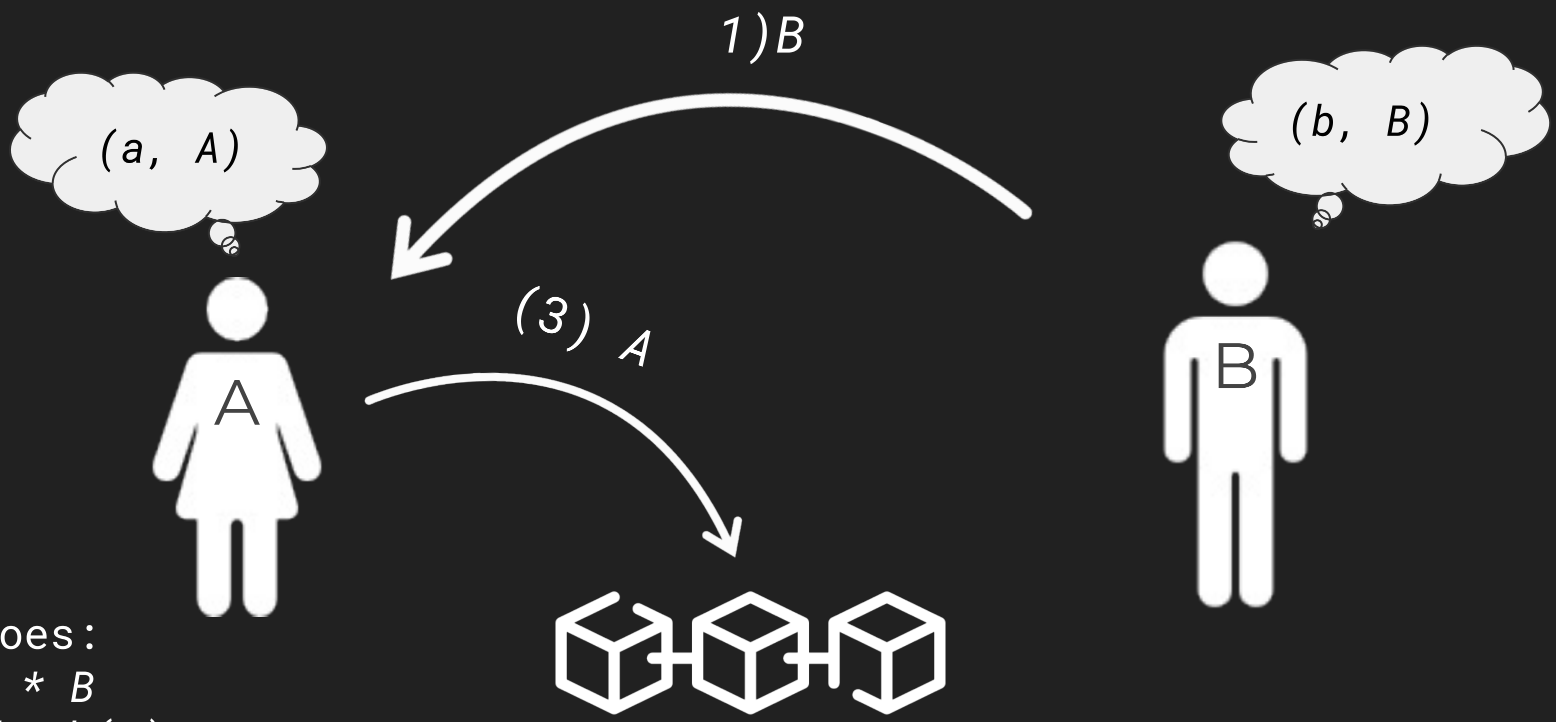


How does it work?

(2) Alice does:

```
s = a * B
sh = hash(s)
stPk = sh * G + B
stAddr = address(stPk)
```

How does it work?

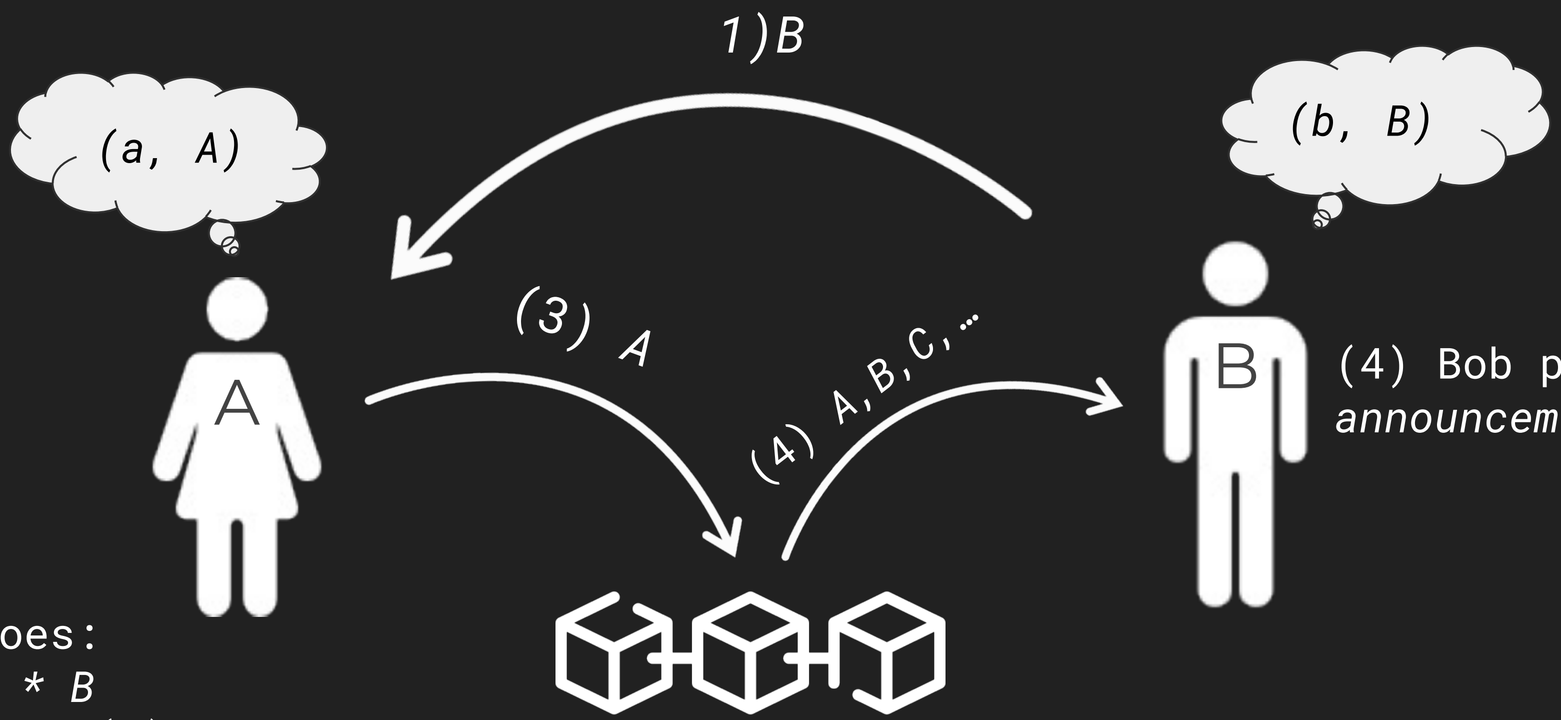


(2) Alice does:

```
      s = a * B
      sh = hash(s)
      stPk = sh * G + B
      stAddr = address(stPk)
```

(3) Alice sends funds to stAddr which is different from address(B) and publishes A.

How does it work?



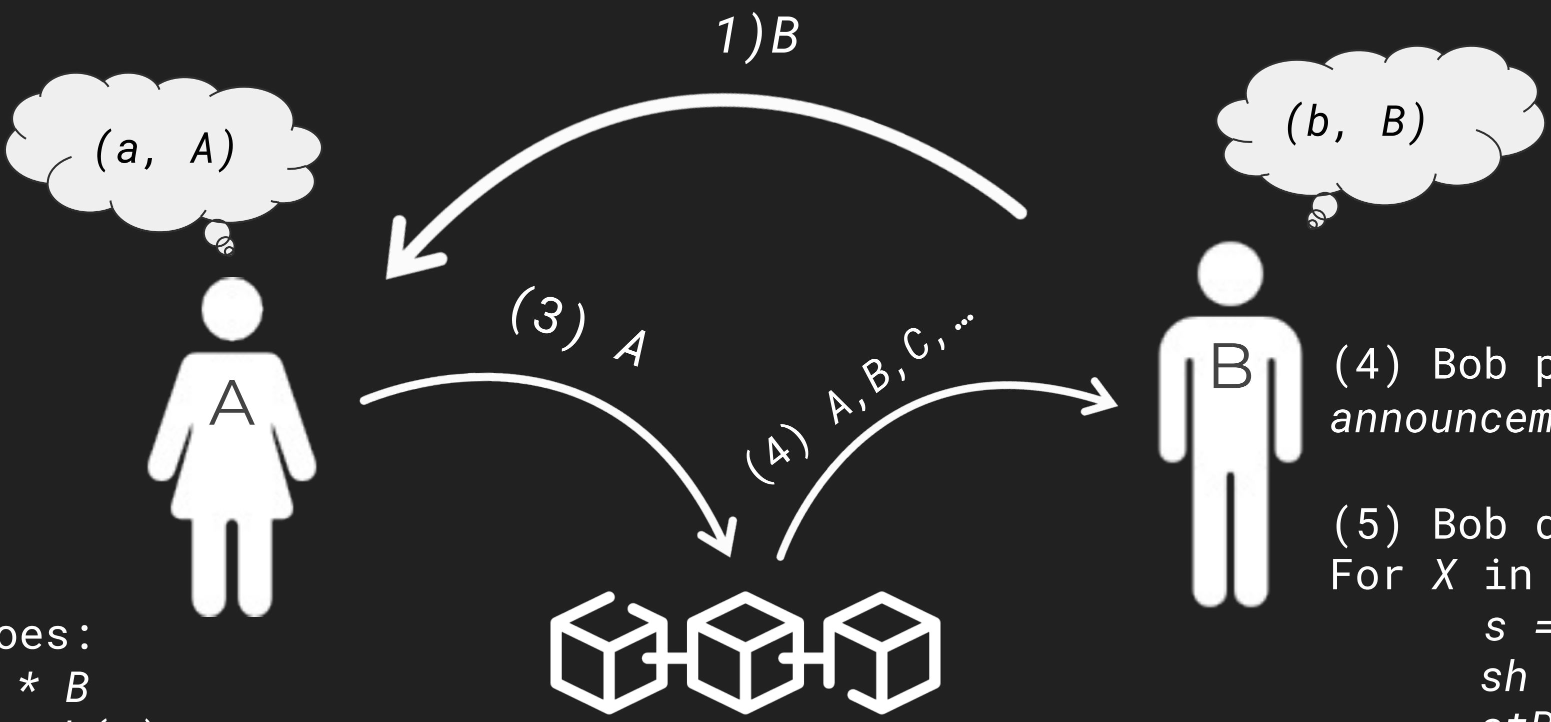
(4) Bob parses all *announcements.*

(2) Alice does:

*s = a * B*
sh = hash(s)
*stPk = sh * G + B*
stAddr = address(stPk)

(3) *Alice sends funds to stAddr which is different from address(B) and publishes A.*

How does it work?



(4) Bob parses all *announcements.*

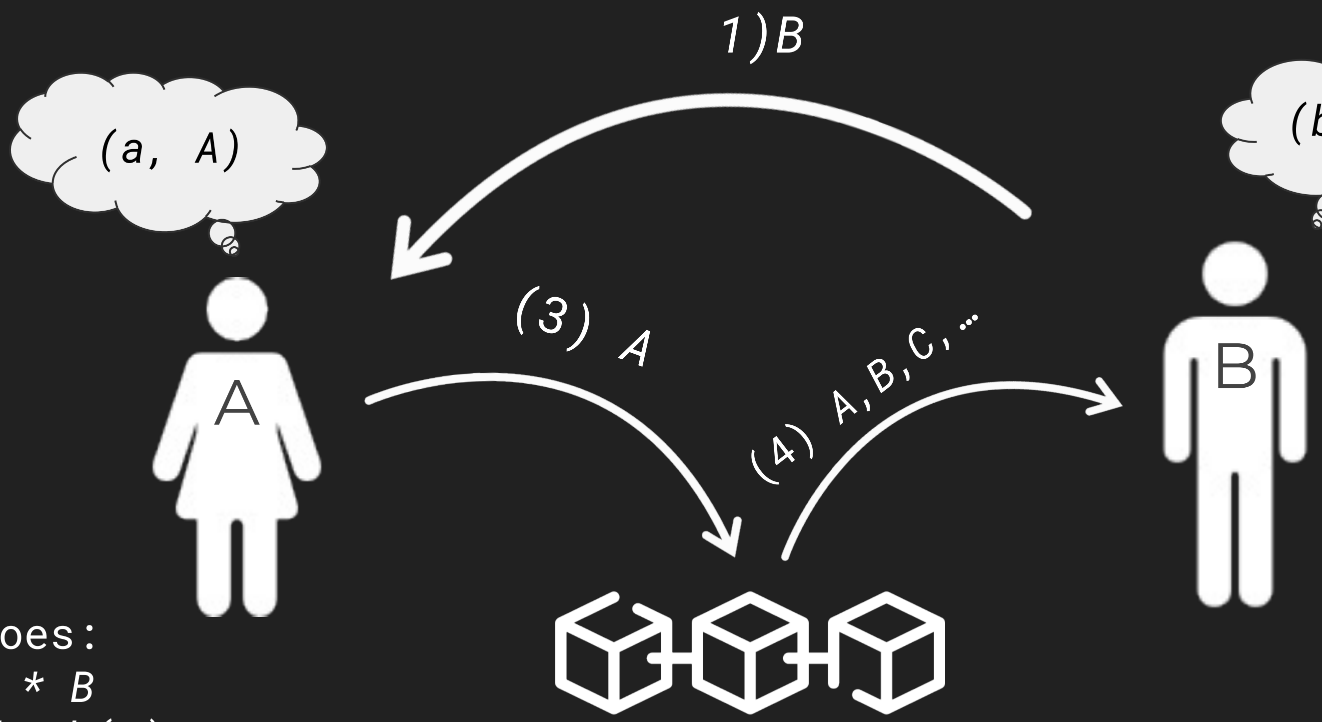
(5) Bob does:
For *X* in { *A*, *B*, *C*, … }:
*s = X * b*
sh = hash(s)
*stPk = sh * G + B*
stAddr = address(stPk)

Bob can compute the private key to the stealth address: *b + sh*

(2) Alice does:
*s = a * B*
sh = hash(s)
*stPk = sh * G + B*
stAddr = address(stPk)

(3) *Alice sends funds to stAddr which is different from address(B) and publishes A.*

How does it work?



(2) Alice does:

    s = a * B
    sh = hash(s)
    stPk = sh * G + B
    stAddr = address(stPk)

(3) *Alice sends funds to stAddr which is different from address(B) and publishes A.*

$sh * G + B = (b+sh) * G$

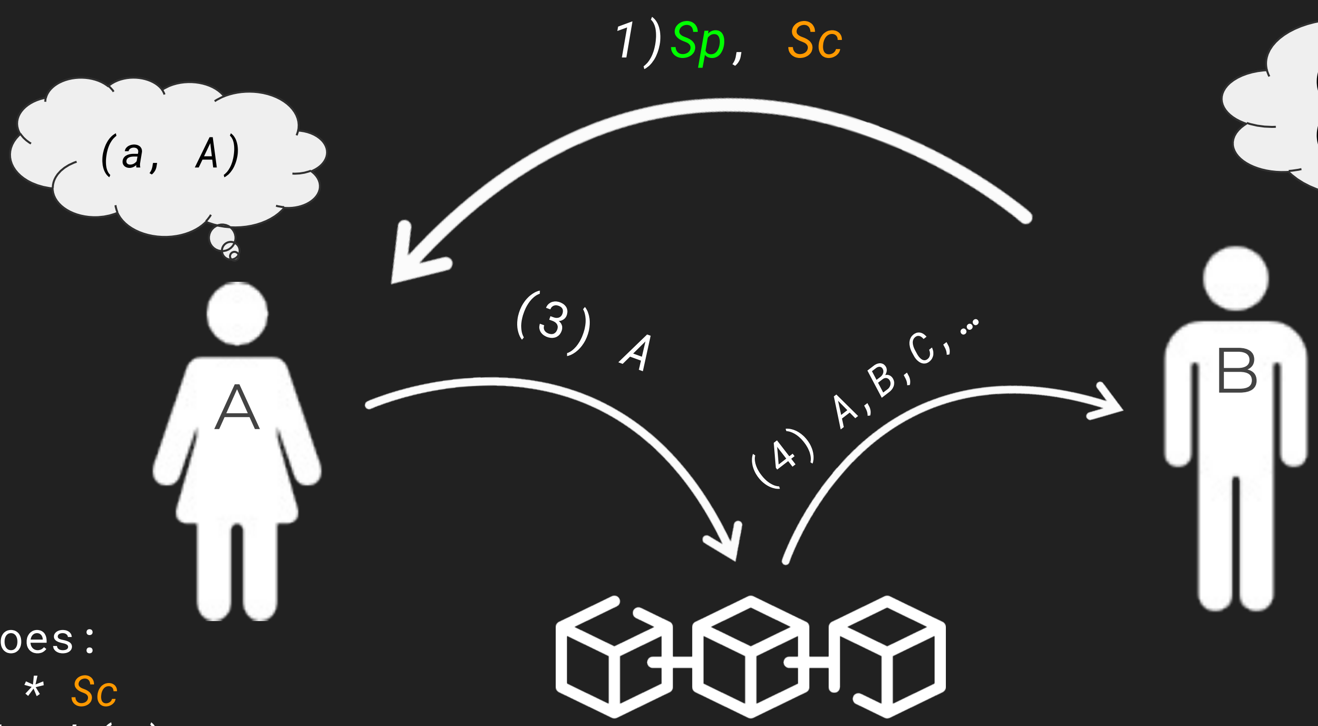
(4) Bob parses all *announcements.*

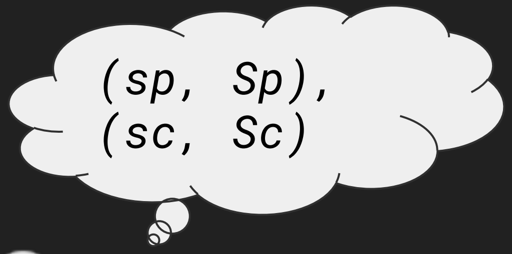
(5) Bob does:
For X in { A, B, C, … }:

    s = X * b
    sh = hash(s)
    stPk = sh * G + B
    stAddr = address(stPk)

Bob can compute the private key for the stealth address: *b + sh*

How does it work?





(4) Bob parses all *announcements.*

(5) Bob does:
For *X* in { *A*, *B*, *C*, … }:
    s = *X* * *sc*
    sh = *hash*(*s*)
    stPk = *sh* * G + *Sp*
    stAddr = *address*(*stPk*)

Bob can compute the private key for the stealth address: *sp* + *sh*

(2) Alice does:
    s = *a* * *Sc*
    sh = *hash*(*s*)
    stPk = *sh* * G + *Sp*
    stAddr = *address*(*stPk*)

(3) *Alice sends funds to stAddr which is different from address(Sp) and publishes A.*

So what about EIP-5564?

EIP-5564

Stealth Addresses ≠ Stealth Addresses

EIP-5564

Stealth Addresses ≠ Stealth Addresses

- Many different possibilities for Stealth Address protocols:
    - Different Elliptic Curves
        - Secp256k1
        - Secp256r1
        - …
    - Elliptic Curve Pairings
    - Lattice-based

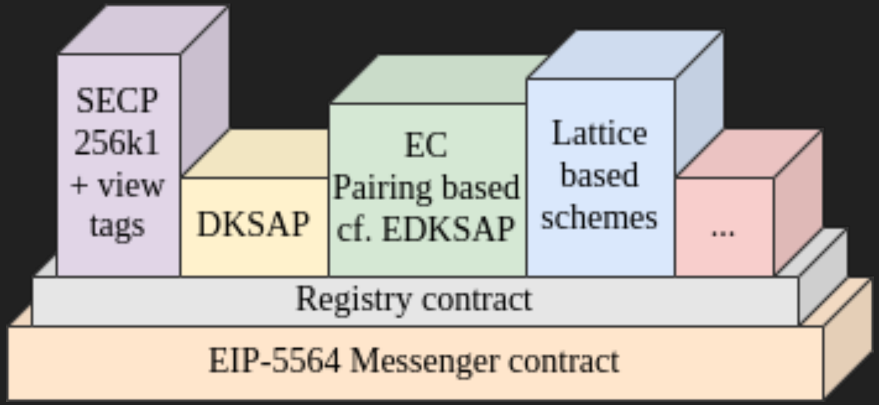
EIP-5564

- “Stealth address and key management techniques in blockchain systems” - Courtois & Mercer (2017)
- “Faster dual-key stealth address for blockchain-based internet of things systems” - Fan (2018)
- “A new stealth address scheme for blockchain” - Fan *et al.* (2019)
- “A lattice-based linkable ring signature supporting stealth addresses” - Liu *et al.* (2019)
- “Blockchain Stealth Address Schemes” - Yu (2020)
- “PDKSAP: Perfected double-key stealth address protocol without temporary key leakage in blockchain” - Feng *et al.* (2020)
- “EDKSAP: Efficient Double-Key Stealth Address Protocol in Blockchain” - Feng *et al.* (2021)
- “A privacy-preserving data transfer in a blockchain-based commercial real estate platform using random address generation mechanism“ - AbdulKadar & Kumar (2022)
- “A Hybrid Design of Linkable Ring Signature Scheme with Stealth Addresses” - Li *et al.* (2022)

EIP-5564

Stealth Addresses ≠ Stealth Addresses

- Many different possibilities for Stealth Address protocols:
  - Different Elliptic Curves
    - Secp256k1
    - Secp256r1
    - …
  - Elliptic Curve Pairings
  - Lattice-based
- Standardization is key



EIP-5564

```
contract IERC5564Messenger {
  /// @dev Emitted when sending something to a stealth address.
  event Announcement (
    uint256 indexed schemeId,
    address indexed stealthAddress,
    bytes ephemeralPubKey,
    bytes metadata
  );

  /// @dev Called by integrators to emit an `Announcement` event.
  function announce (
    uint256 schemeId,
    address stealthAddress,
    bytes memory ephemeralPubKey,
    bytes memory metadata
  )
    external
  {
    emit Announcement(schemeId, stealthAddress, ephemeralPubKey, metadata);
  }
}
```

Scheme ID to indicate the specific stealth address protocol.

Stealth Address to enable recipient to quickly discover their stealth addresses without RPC calls.

Ephemeral Public Key to enable recipients finding their stealth address and compute the stealth private key.

Metadata for additional information to improve UX.

EIP-5564

Stealth meta-address format:

```
st:eth:0x<spendingKey><viewingKey>
```

st:eth:0x0385b15e0d16672bbe2b215b86742ee6ba0b1f89
b01f35e4dc30ef4dd2eec967770387de997ce72ad74be3072
e02ca5a31f187c6306101047f9f81ecc626c4abebc3

EIP-5564

Stealth meta-address format:

```
st:eth:0x<spendingKey><viewingKey>
```

st:eth:0x0385b15e0d16672bbe2b215b86742ee6ba0b1f89
b01f35e4dc30ef4dd2eec967770387de997ce72ad74be3072
e02ca5a31f187c6306101047f9f81ecc626c4abebc3

EIP-5564

Stealth meta-address format:

```
st:eth:0x<spendingKey><viewingKey>
```

st:eth:0x0385b15e0d16672bbe2b215b86742ee6ba0b1f89
b01f35e4dc30ef4dd2eec967770387de997ce72ad74be3072
e02ca5a31f187c6306101047f9f81ecc626c4abebc3

Prefix: st:eth:0x

Compressed PubKey I:
0385b15e0d16672bbe2b215b86742ee6ba0b
1f89b01f35e4dc30ef4dd2eec96777

Compressed PubKey II:
0387de997ce72ad74be3072e02ca5a31f1
87c6306101047f9f81ecc626c4abebc3

EIP-5564

Stealth meta-address format:

```
st:eth:0x<spendingKey><viewingKey>
```

st:eth:0x0385b15e0d16672bbe2b215b86742ee6ba0b1f89
b01f35e4dc30ef4dd2eec967770387de997ce72ad74be3072
e02ca5a31f187c6306101047f9f81ecc626c4abebc3

EIP-6538:

Ethereum Improvement Proposals

All Core Networking Interface ERC Meta Informational

Draft Standards Track: ERC

ERC-6538: Stealth Meta-Address Registry

A registry to map addresses to stealth meta-addresses

Authors	Matt Solomon (@mds1), Toni Wahrstätter (@nerolation), Ben DiFrancesco (@apbendi), Vitalik Buterin (@vbuterin)
Created	2023-01-24
Discussion Link	https://ethereum-magicians.org/t/stealth-meta-address-registry/12888

Find more…

- PoC: stealth-wallet.xyz
- Tutorial: nerolation.github.io/stealth-utils/
- EIP-5564: eips.ethereum.org/EIPS/eip-5564
- EIP-6538: eips.ethereum.org/EIPS/eip-6538

Behavioural Threats in Decentralized Federated Learning: A Dynamic Assessment Approach

Authors: Khan, Sajjad; Gomes Jr., Jorao; Rehman, Muhammad Habib ur; Svetinovic, Davor

Under review in IEEE Transactions on Dependable and Secure Computing.

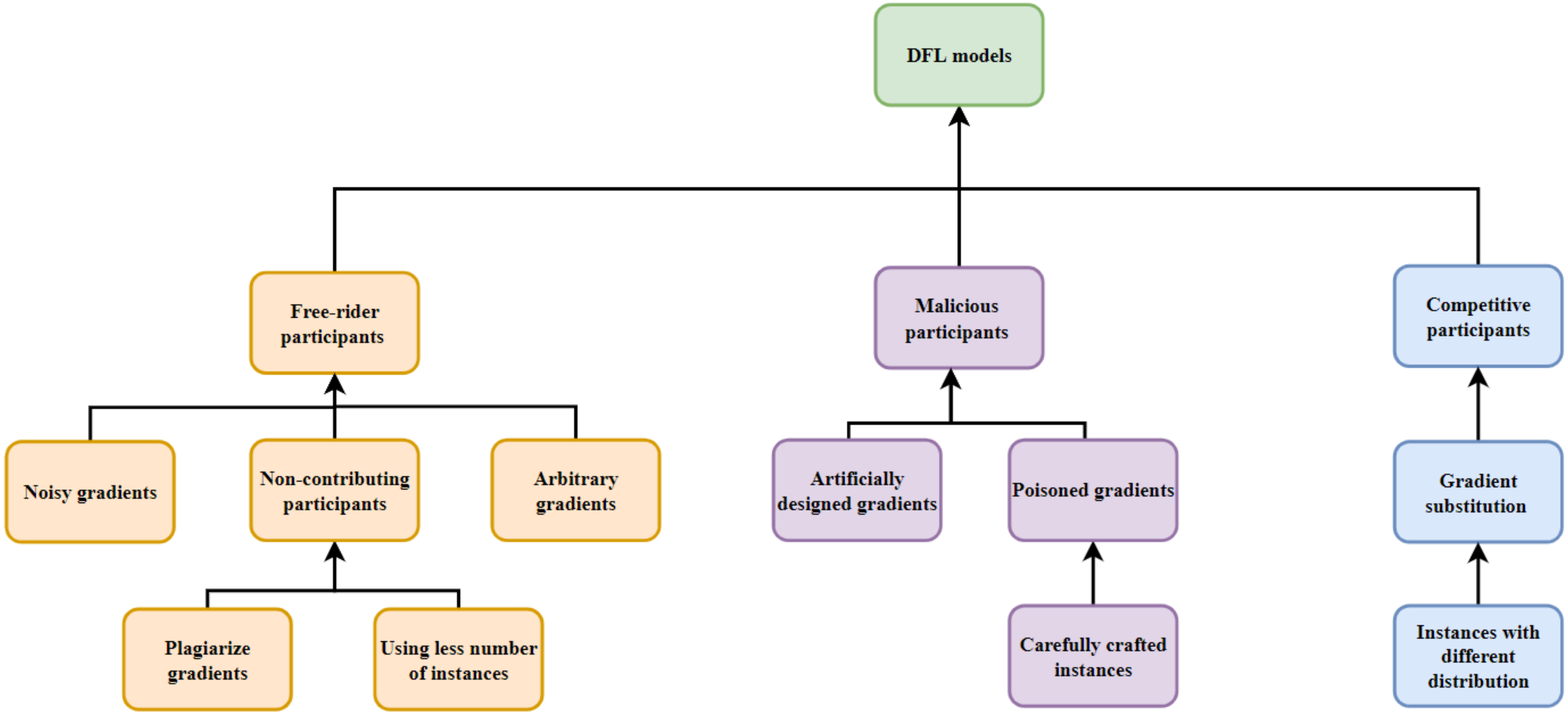
Introduction & Background

- Traditional Machine Learning (Data to Code)
- Federated Learning (Code to Data)
  1. Single point of failure/bottlenecks
  2. Curious Server
  3. Lacks the capability to detect adaptive behaviour
- Decentralized Federated Learning

Research Questions

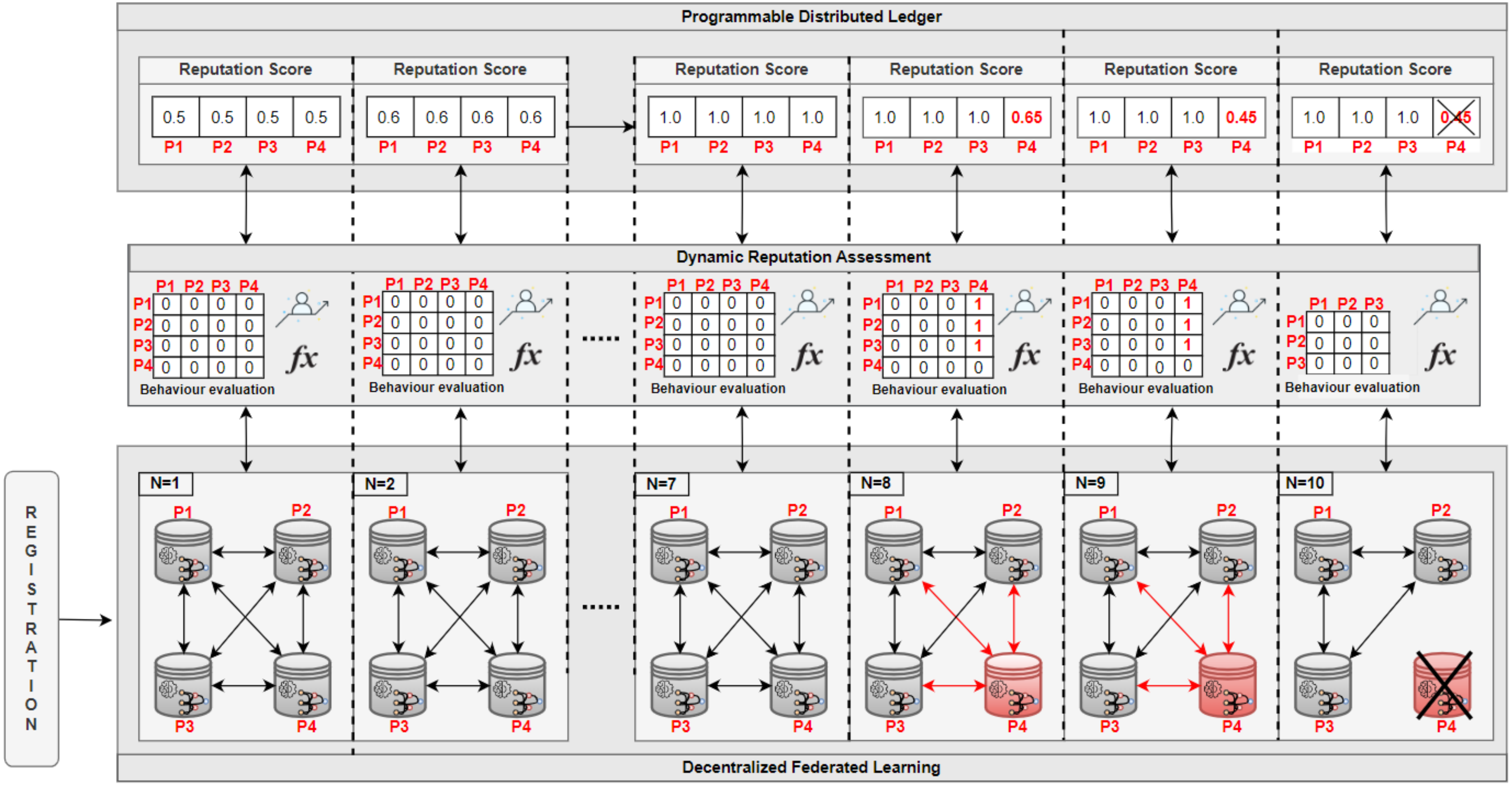
- What are the common issues that hinder the efficiency of the DFL?
- Adaptive behaviour is not recognized (only free rider)
- How do the existing DFL systems tackle the adaptive behaviour of participants?
- Assuming honest behaviour
- incentivizing to behave honest.
- How to ensure the correct performance of DFL systems?
- Ability to detect/mitigate and quick elimination procedure to avoid confidence degradation.

Taxonomy of behaviour threats



It is assumed that Free-riders are only non-contributing participants.

Proposed architecture

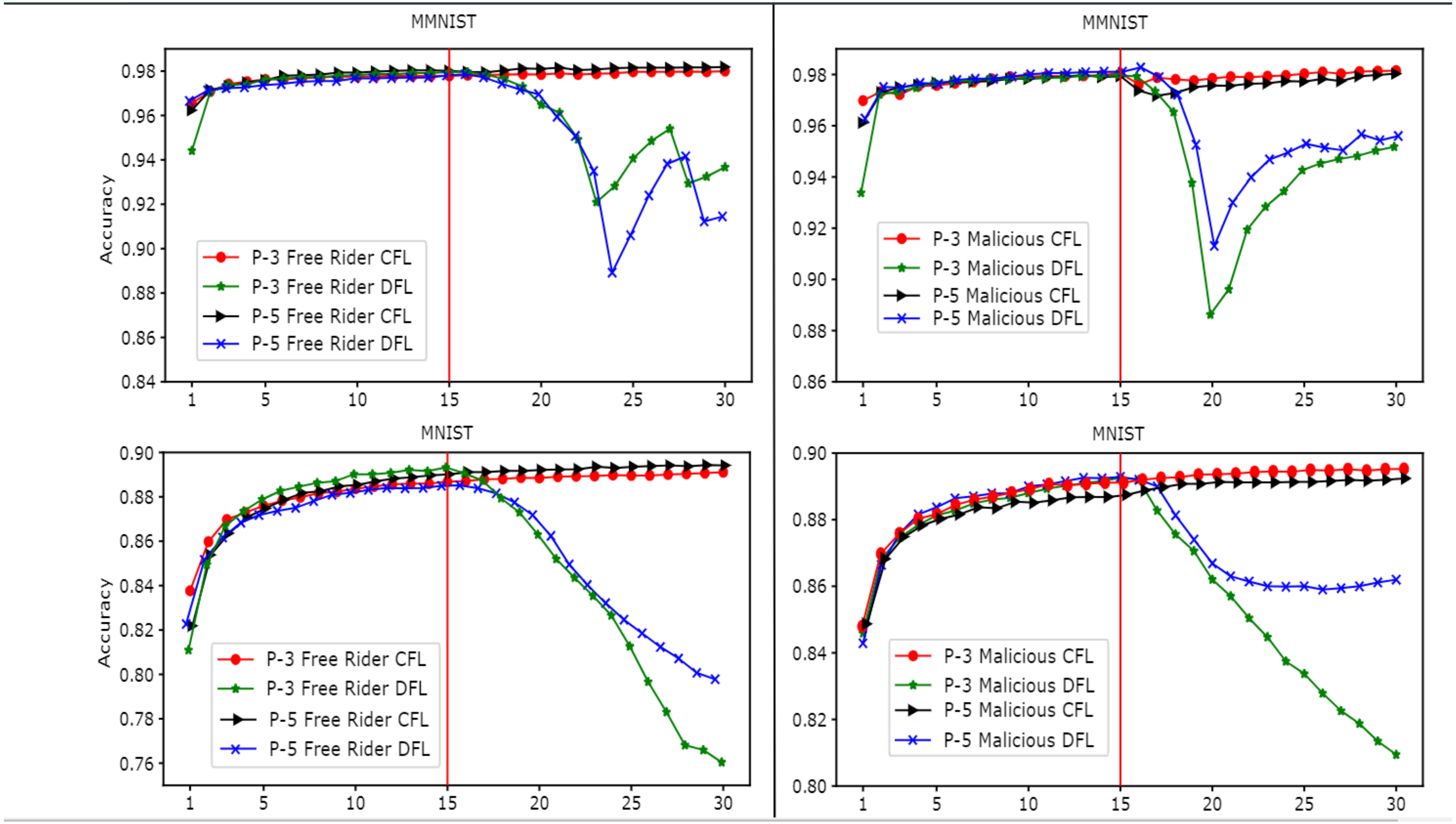


Behavioural evaluation algorithm

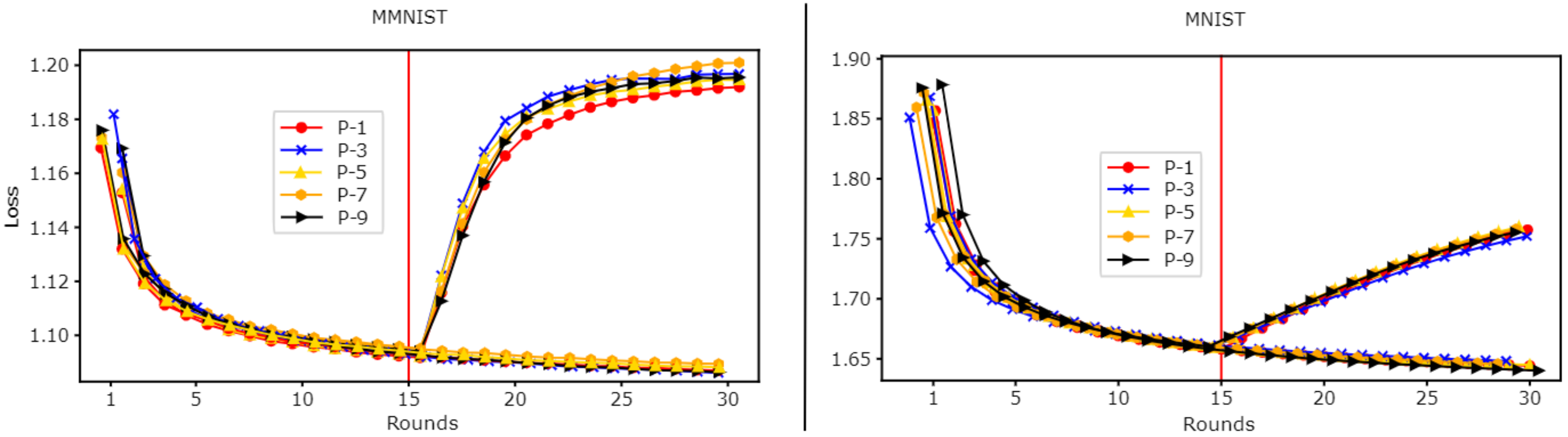
Algorithm 1 Behaviour evaluation algorithm

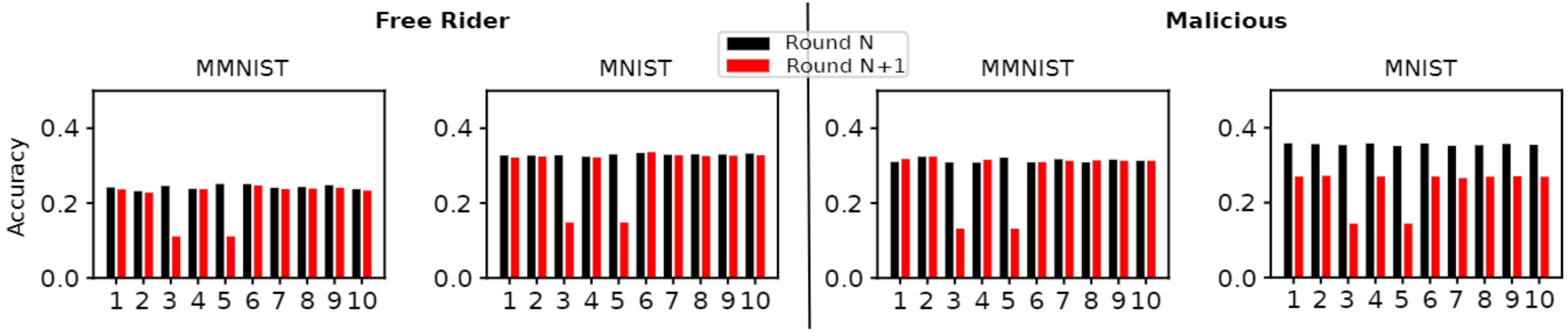
```
Input: accuracy_results, α
Output: malicious_index
malicious_report ← ∅
mean ← mean(accuracy_results)
std ← std(accuracy_results)
lower_limit ← mean − α × std
for i ∈ (0, len(accuracy_results)) do
    if accuracy_results[i] < lower_limit then
        malicious_index ← malicious_index ∪ i
    end if
end for
return malicious_index
```

Experiments



Experiments



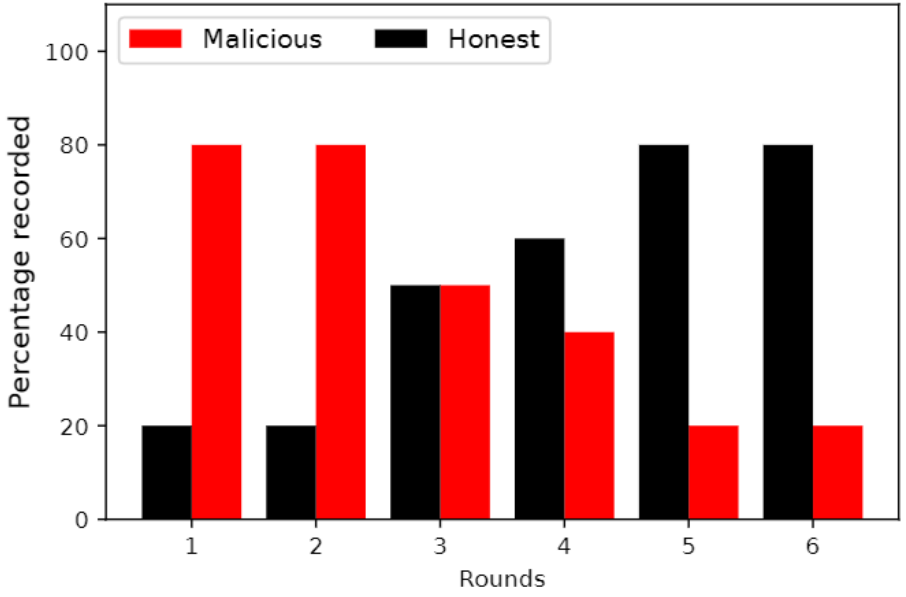


Why CFL fails.

TABLE 1
Example of the impact of malicious gradients on the users' confidence.
Read UID as a User ID and R as Round

	User 6			User 9		
UID	R1	R2	R3	R1	R2	R3
1	0.299	0.164	0.110	0.335	0.234	0.190
2	0.296	0.163	0.110	0.330	0.232	0.189
3	*0.198*	*0.119*	0.109	*0.261*	*0.207*	*0.184*
4	0.299	0.159	0.110	0.334	0.228	0.188
5	*0.206*	*0.123*	0.109	*0.274*	*0.209*	*0.186*
6	0.296	0.166	0.110	0.334	0.234	0.193
7	0.298	0.161	0.110	0.333	0.227	0.188
8	0.298	0.169	0.110	0.337	0.237	0.191
9	0.299	0.162	0.110	0.333	0.231	0.191
10	0.304	0.179	0.110	0.335	0.239	0.196

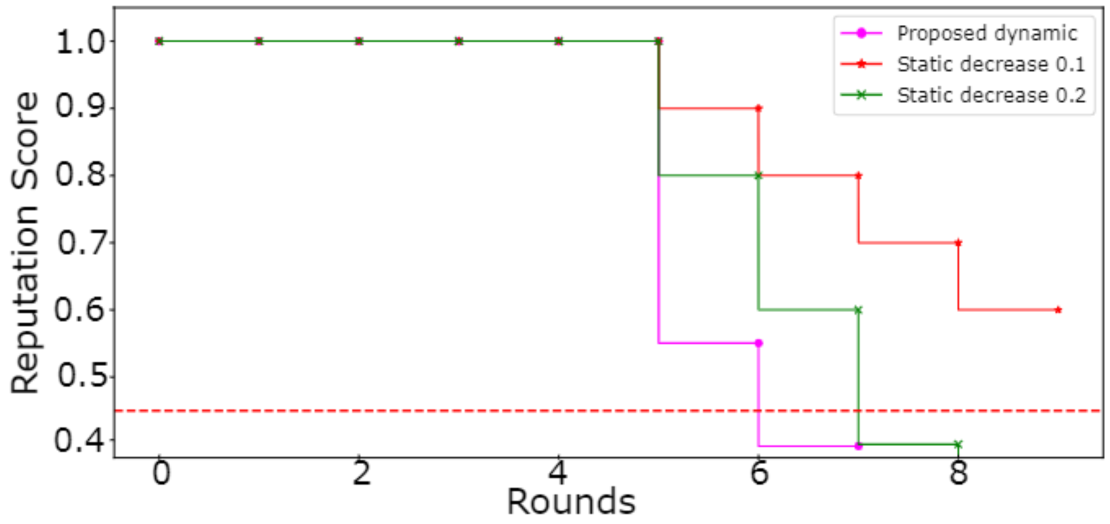
How confidence level degrades



How it works.

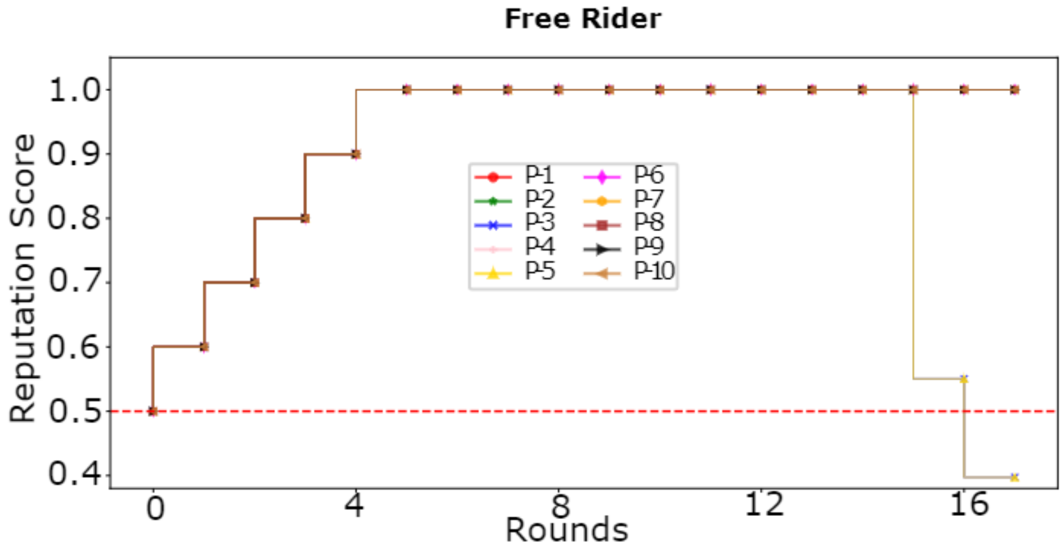
Input: $score\_metrics, malicious\_reports$
Require: $n\_participants, reputations$
Output: Updated reputation scores

```
P_factor ← 1
report_votes ← [0] * n_participants
for report ∈ malicious_reports do
    for id ∈ report do
        report_votes[id] ← report_votes[id] + 1
    end for
end for
for id ∈ report_votes do
    votes ← report_votes[id] / n_participants
    if votes > 0.5 then
        for results ∈ score_metrics do
            mean ← mean(results)
            value ← results[id]
            tmp_P_factor ← value / mean
            if P_factor > tmp_P_factor then
                P_factor ← tmp_P_factor
            end if
        end for
    end if
end for
for id ∈ report_votes do
    votes ← report_votes[id] / n_participants
    if votes > 0.5 then
        reputations[id] ← reputations[id] × P_factor
    else
        reputations[id] ← min(1, reputations[id] + 0.1)
    end if
end for
```

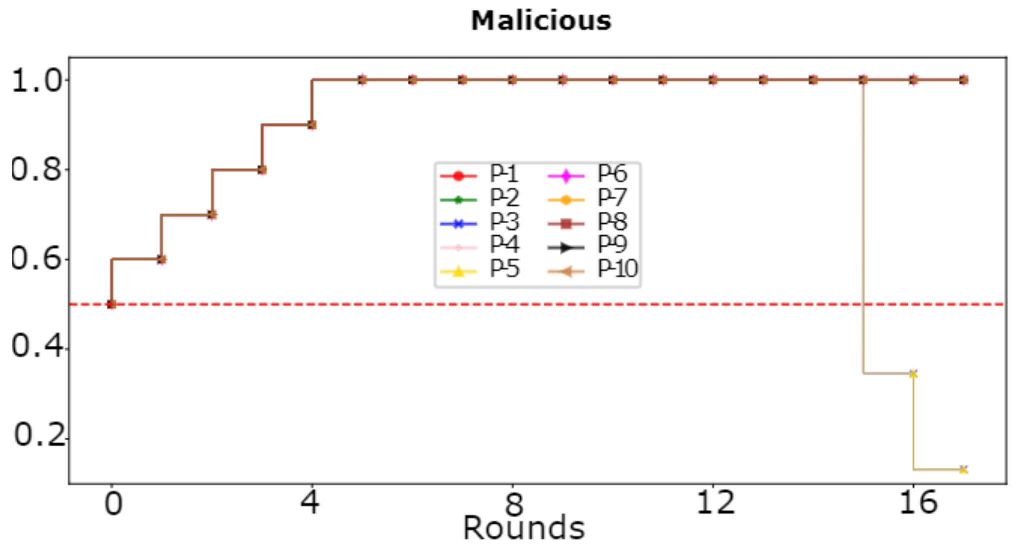


Comparison to common ways of reputation decreasing

Evaluating Honest/Adaptive behaviour.



How it worked.



And 3 more recent works under review

- OpenFL: Ethereum-Based Decentralized Federated Learning for Trustworthy Collaboration
- Crypto-economic Blockchains Under Geopolitical Stress: Analyzing User Behavior During the Acute Stage of Russia–Ukraine Conflict
- Fortifying the Blockchain: A Systematic Review and Classification of Post-Quantum Consensus Solutions for Enhanced Security and Resilience

Now to overall summary, insights, and directions…

Blockchain and AI: A Promising Direction

- AI-driven smart contracts and decision-making
- Enhanced security and fraud detection
- Autonomous organizations and decentralized applications
- Federated learning for improved privacy
- Challenges: trust, ethics, and system manipulations

Blockchain and IoT: Unlocking New Possibilities

- Decentralized IoT device management and data sharing
- Enhanced security and data privacy
- Supply chain traceability and transparency
- Smart homes, cities, and infrastructure
- Challenges: scalability, interoperability, and standards

Engineering Trust in Blockchain Systems

- Secure and privacy-preserving protocols
- Transparent governance and decentralized decision-making
- Regulatory compliance and legal frameworks
- Reputation systems and identity management
- Interoperability and standardization

Censorship-Resistant Federated Learning

- Decentralized machine learning on permissionless blockchains
- Privacy-preserving data sharing and collaboration
- Resilience against data poisoning and sybil attacks
- Incentivizing honest participation and contribution
- Challenges: scalability, efficiency, and incentives

Blockchain System Turbulences

- Market fluctuations and speculation
- Illegal activities and system manipulations
- Geopolitical conflicts and regulatory changes
- Technological innovations and breakthroughs
- Public perception and media influence

Engineering Mechanisms to Mitigate Turbulences

- Market mechanisms for stability and price discovery
- Machine learning for prediction and early warning systems
- Adaptive governance and responsive regulation
- Security and privacy enhancements
- Community engagement and education

Increasing Trustworthiness and Reliability

- Building on secure and privacy-preserving foundations
- Transparent governance and decision-making
- Collaboration with regulators and legal frameworks
- Enhancing system resilience and adaptability
- Encouraging research and development in critical areas

Conclusion

- Blockchain, AI, and IoT offer significant potential for innovation
- Addressing engineering challenges to build trustworthy systems
- Understanding and mitigating blockchain turbulences
- Increasing public acceptability and maximizing benefits
- Ongoing research and collaboration for a more resilient future

Thank you! See you at the panel later!

Here are three jokes that combine blockchain, AI, and quantum computing themes:

1. Why did the AI-powered quantum computer invest in cryptocurrencies?

   Because it calculated that the odds of success were both one and zero at the same time!

2. What do you get when you cross a blockchain enthusiast, an AI researcher, and a quantum computing expert?

   A decentralized, self-learning, and superpositioned party!

3. Why was the AI-driven quantum blockchain so popular?

   Because it could simultaneously verify transactions, learn from its mistakes, and exist in multiple states, all while keeping a cryptic sense of humor!