Original Research Paper

Application of Elliptic Curve Crypto System to Secure Multi-Signature Bitcoin Block Chain

Mohammed Mujeer Ulla and Deepak S. Sakkari

Department of Computer Science and Engineering, Presidency University Bengaluru, 560064, India

Article history
Received: 16-11-2022
Revised: 10-12-2022
Accepted: 13-12-2022

Corresponding Author:
Mohammed Mujeer Ulla
Department of Computer Science and Engineering, Presidency University Bengaluru, 560064, India
Email:
mohammedmujeerulla@presidencyuniversity.in

Abstract: Blockchain is a technology that enables decentralized digital currencies like bitcoin, ethereum, and ripple. It is widely used in many areas such as healthcare, defense, and industrial domains like internet of things for tracking sensor data and detecting duplicate sensor data. Bitcoin is one of the most popular cryptocurrencies due to its market value and use as a medium of exchange. The trustless secure money exchanges have made bitcoins appealing to people. However, the loose possession of bitcoin keys can lead to monetary loss and a decrease in bitcoin users. In this article, we provide an extensive analysis of major privacy and security issues in the Bitcoin blockchain. First, we discuss the security issues in bitcoin, and second, we discuss countermeasures for these bitcoin threats. Third, we provide an added level of security using elliptic curve cryptography on bitcoin multi-signature accounts. Our research helps bitcoin users understand the risks involved in using non-multi-signature accounts compared to multi-signature accounts. The use of elliptic curve cryptography on bitcoin multi-signature accounts also helps to avoid unnecessary expenses such as bitcoin currency rewards to miners.

Keywords: IoT-Internet of Things, ECC-Elliptic Curve Cryptography, SEC-U.S. Securities and Exchange Commission, The NIST-National Institute of Standards and Technology, NSA-National Security Agency, EdDSA-Edwards Curve Digital Signature Algorithm Nonce-Number only used once, RAG-Random Number Generator

Introduction

The concept of bitcoins was first proposed by Satoshi Nakamoto in 2008 when a peer-to-peer electronic cash-based transaction was presented on a white paper (Patel and Doshi, 2020). The security of digital currency is based on public key cryptography where a public open blockchain contains a history of all the transactions and is secured by consensus algorithm proof-of-work. In bitcoin, we send and receive digital currency without the involvement of a third party and any double spending (Hammi *et al*., 2020). In 2022 it is estimated that bitcoin has 180 million users around the globe and over 2.5 million transactions take place in bitcoins each day (Zhang and Wang, 2018). Some of the enabling technologies for bitcoin are blockchain, the consensus algorithm proof-of-work, and public-key cryptography. As an adverse effect of these technologies bitcoin offers properties like immutable decentralized trustless and permissionless setup. Every bitcoin user uses his own private keys in the wallet to create a bitcoin public address and digital signatures for every transaction on the bitcoin blockchain. The bitcoin wallets make it possible to send, and receive bitcoins and to monitor all the Unspent Transaction Output (UTXOs). Each bitcoin transactions are censorship-resistant, borderless, and pseudonymous due to bitcoin's private keys. The ecosystem of bitcoin is composed of technical, human dynamics, organizational and social components.

Furthermore, bitcoins governance rules are decided, implemented, and enforces when complete consensus is agreed upon among all the bitcoin users (Khan *et al*., 2020). Our research addresses how to monitor, administer and manage bitcoin private keys. There are many forms of wallets available online the choice of selection truly depends on security and usability. A secure bitcoin is said to be sustainable if it relies strongly on the security aspect of private keys (Ouni and Bouallegue, 2016). The protection of private keys includes ensuring confidentiality thereby making unauthorized access to private keys impossible. In general, a bitcoin transaction is a peer-to-peer exchange of bitcoins through the use of asymmetric cryptography using a public bitcoin address derived from a public key. A bitcoin wallet is an application through which the user his private keys, generation of addresses, signs transactions, and tracks inputs and outputs.

Materials

Equipment's used by the researchers to investigate the research is as follows:

Raspberry *Pi* 4 8GB RAM
1. All new raspberry *pi* desktop computer memory size: 8 GB
   RAM memory: 8 GB
   Resolution 3840 × 2160
2. HDMI to VGA cable-for raspberry
3. USB C power supply cable-for raspberry
4. Consumables-bitcoins
   Miscellaneous- external hard disk
5. Seagate expansion 1TB external HDD
   memory card
6. SanDisk extreme 160MB/S U3 A2 micro SDXC memory card(1TB)
   Monitor-lenovo Q24i-1L 60.4 cms (23.8)
7. Full HD natural low blue light monitor
   keyboard mouse HP 230 wireless
8. Optical keyboard and mouse combo

Methods

In this section, we focus on the generation of multi-sig for bitcoins. Due to the enormous amount of fishing and hacking over the internet multi-sigs are one of the best options available right now to store your bitcoins (Park and Park, 2016). Multi-sigs are exponentially more secure. All the bitcoin multi-signature addresses begin with the number 3. Any bitcoin address beginning with a number 3 needs at least 3 out of 3 or 2 out of 3 or 1 out of 3 private keys to release the funds. When it comes to security setting up multi-sig accounts has numerous advantages (Boneh and Venkatesan, 1996). Figure 1 shows the generation of the multi-signature using bitcoin where users can store their private keys separately on their computer machine or in a safe deposit box or even with online wallet providers like block chain.info which is incorporating multi-signatures. Wallet providers do some regular sort of banking analysis is this a usual spend or over the daily limits or preferred merchants, they run different sorts of algorithms to safeguard your bitcoins in user accounts. Even if wallet providers are not in the business or they do not permit your transaction you still have control over transactions because you have 2 private keys out of 3. A lot of coins are being stolen by hackers but hackers find it difficult in doing permutations and combinations of different devices. Most of your security comes in by keeping your private keys separately on different devices or even on a paper wallet or in cold storage. 2 of 3 exponentially increase your chances of keeping your bitcoins secure than keeping in on one single device. The bitcoin environment and cyber security are based on social and technical factors (Breitner and Heninger, 2019). It is required to address both factors to enable people to use the bitcoin network securely. Previously many researchers have worked on improving key management security aspects of wallets. To our knowledge, we are the first to examine the application of elliptic curve cryptography on bitcoin wallets.

A. Generation of Bitcoin Multi-Signature Address

Table 1 shows the top 20 bitcoin rich list since no addresses start with 3. This means in some way the addresses are compromised with the level of security. Tony Gallippi from bitpay states top 100 addresses will be protected by multi-sig where 1 means insecure and 3 means added level of security. It means those addresses were created using multi-sig. Multi-sig facilitates you to have a trustless setup with a wallet provider. The wallet provider can monitor and safeguard your account, but they are too not empowered completely to spend funds as they have access to a single signature and to spend bitcoins you need a minimum of two signatures.

This eliminates them from any sort of liability and they will not be able to go out of business and leave you cold. Multi-signature provides another massive layer of security to any cryptocurrency. The multi-signature also enables to have a three-party escrow setup where there is a credit transaction from Alice to Bob and both are trust-less parties to each other but they trust a centralized transaction like eBay. If there is a transaction dispute between Alice and Bob then eBay would resolve this dispute as eBay possess the third key and now will be able to finish the transaction by sending funds from Alice to Bob or vice versa. Having a multi-signature is also better than having a Shamir's secret which is a kind of system like having one private key on five pieces of paper and any permutation and combination from it will form a complete private key (Breitner and Heninger, 2019). For example, 4 pieces of paper out of five will give you back the original private key. Though its technique is based on permutations and combination it pops out all the flaws we had earlier because any time you bring those four pieces of paper together one of the people who has helped to get those pieces of paper together will have access to the private key and can quickly spend the bitcoins using the obtained private key or it can lead to any form of a man in middle attacks (Shaikh *et al.*, 2017).

Initially, we use a bitcoind client which generates a multi-sig address on a brand new address or it can be an old address as well. As long as you have a private key to a public address you can add it to a multi-signature address. Suppose two brothers each independently can set up their own private keys, and set up their own little wallet with that you generate their multi-signature address than a combination of three can sign off the transactions without having compromised the keys with either one of you. For our research, we generate three brand new address pairs such as a compressed public address (key) of 34 characters, a private key of 52 characters, less compressed public address (key) of 66 characters which we can use to create multi-signature addresses this is shown in Table 1. Then all three less compressed public addresses generated are given as input to the bitcoind /bitcoin-QT client which will spit out multi-signature addresses starting with number 3 as shown in Table 2. Bitcoind creates a multi-sig (number) will create a multi-signature account, where the number stands for the number of private keys you need to unlock the funds, in our case we have the number as 2 which means we need a minimum of two signatures to unlock the funds. If you switch it to 1 any one of these private keys could spend from a multi-sig address. If you switch it to 3 then all their private key is needed to spend from a multi-sig address. Currently, we have worked with a maximum of three keys to generate the multi-sig address while in the future there may be 6 out of 10 keys or 8 out of 12 keys, or any other permutations that can be predicted. To do this we need bitcoind to have calls to your version of the blockchain. A redeem script is generated which is 210 bits which are required to spend the bitcoins from a multi-signature account irrespective of 1 of 3 multi-signature or 2 of 3 multi-signature or 1 of 3 multi-signature account, Table 3 and Fig. 3 shows different multi signatures generated using variable *N*.

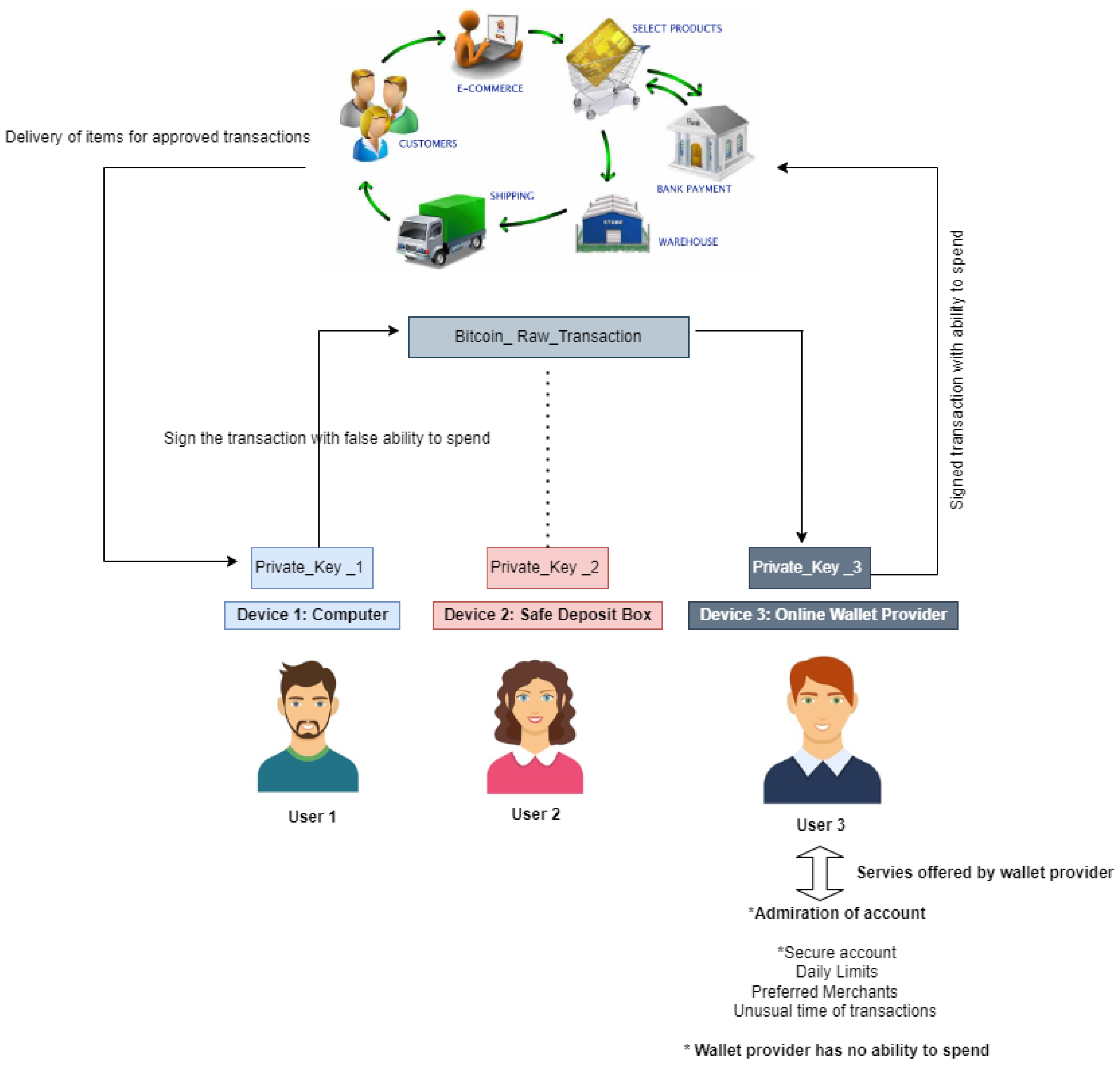

SELECT PRODUCTS
E-COMMERCE
CUSTOMERS
BANK PAYMENT
SHIPPING
WAREHOUSE
Delivery of items for approved transactions
Bitcoin_ Raw_Transaction
Sign the transaction with false ability to spend
Signed transaction with ability to spend
Private_Key _1
Private_Key _2
Private_Key _3
Device 1: Computer
Device 2: Safe Deposit Box
Device 3: Online Wallet Provider
User 1
User 2
User 3
Servies offered by wallet provider
*Admiration of account
*Secure account
Daily Limits
Preferred Merchants
Unusual time of transactions
* Wallet provider has no ability to spend

Fig. 1: Generation of the multi-signature using bitcoin

Table 1: Bitcoin rich list -2022

Public Address
bc1qgdjqv0av3q56jvd82tkdjpy7gdp9ut8 bitcoin balance: 166,510.99 Security assured: Not secure account
Public address: 1P5ZEDWTKTFGxQjZphgWPQUpe554WKDfHQ bitcoin balance: 112,155.62 Security assured: Not secure account
Public address: 1FeexV6bAHb8ybZjqQMjJrcCrHGW9sb6uF bitcoin balance: 79,957.21 Security assured: Not secure account
Public address: 1LdRcdxfbSnmCYYNdeYpUnztiYzVfBEQeC bitcoin balance: 53,880.06 Security assured: Not secure account
Public address: 1AC4fMwgY8j9onSbXEWeH6Zan8QGMSdmtA bitcoin balance: 51,830.36 Security assured: Not secure account
Public address: bc1q5shngj24323nsrmxv99st02na6srekfctt30ch bitcoin balance: 45,558.99 Security assured: Not secure account
Public address: 1LruNZjwamWJXThX2Y8C2d47QqhAkkc5os bitcoin balance: 44,000.09 Security assured: Not secure account
Public address: 1NDyJtNTjmwk5xPNhjgAMu4HDHigtobu1s bitcoin balance: 36,116.16 Security assured: Not secure account
Public address: 17hf5H8D6Yc4B7zHEg3orAtKn7Jhme7Adx bitcoin balance: 36,000.01 Security assured: Not secure account
Public address: bc1qvpgyac88vqtslewxu7yu9dqwp8rd83z bitcoin balance: 35,829.98 Security assured: not secure account

Table 2: Brand new address pairs generated

Address pair: Number 1 compressed public address-34 chars
1L2hyX0KmtRffKBhQBXGyWFVB9ZHJckkbx Private Key - 52 chars: L2MluR9dwgCotoPBprWJYYWz2zWW9sMa9TJwqARG7nFxkpdvBSsm Less compressed Public Key/Address - 66 chars: 020743d44be989540d27blb4bbbcfd1772lc337cb6bc9a 20eb8a3252Ob393532f
Address pair: Number 2 Compressed Public Address - 34 chars: 13XTNTMpsAc4a.EPniAncmnDxWJcY6QcPF Private Key - 52 chars: LlM2ZgjOAtDVU9uemahiZBQPWFA5Tyj4GLdlECkDryv iFrGp6m7k Less compressed Public Key/Address - 66 chars: 02c0120aldda9e51a93Bd39ddd9fe0ebc45ea97eld27a7cbd671d5431 416d3dd87
Address pair: Number 3 Compressed Public Address - 34 chars: lDT8Ki2BKzFYUVnv9h99P91EESG32kmP3Y Private Key - 52 chars: L5PkVBzR4SdQimMsf8nRqRe9JZDJ22sGjSbfp3SsPSnVoBBvRFE Less compressed Public Key/Address - 66 chars: p213820eb3dSf509d743Bc9eeecb4157b2f595105e7cd564b3cdbb9ead3da4leed
For fun, you can paste this into bitcoind to verify multi-sig address bitcoind createmulti-sig 2 '["020743d44be989540d27blb4bbbcfdl772lc337cb6bc9af20eb8a32520b393532f","02c0120aldda9e5la938d39ddd9f e0ebc45ea97eld27a7cbd67ld5431416d3dd87","0213820eb 3d5f509d7438c9eeecb4157b2f595105e7cd564b3cdbb9ead3da4leed"] ' The multi-sig address is 34 chars: 35Z3xG92YkW5Xo4ngQw6w5b3Ce6MDw94A8

Table 3: Different multi-signatures generated using variable N

Trial I Generating the same multi-signature using N=2 bitcoind creates multi-sig 2
("020743d44be9S9540d27b1b4bbbcfd1n21c337cb6bc9af20ebS a32520b393532f, "02c0120a1dda9e51a93Sd39ddd9fe0ebc4Sea 97eld27a7cbd671d5431416d3ddS7","0213S20eb3d5f509d743Sc 9eeecb4157b2f595105e7cd564b3cdbb9ead3da41eed") Multi Sig address:"35Z3xG92YkWSXo4ngQw6w5b3Ce6MDw94A8", RedeemScript:"5221020743d44be989540d27b1b4bbbcf dlm1c337cb6bc9af20eb8a32520b393532f2102c0120a1dda9e51a93Sd3 9ddd9fe0ebc4Sea97e1d27a7cbd671d5431416d3ddS7210213S20eb3d5 f509d743Sc9eeecb4157b2f595105e7cd564b3cdbb9ead3da41eed53ae"
Trial II Generates the same multi-signature using N = 2 and gives the same multi-signature as in trial I bitcoind create multi-sig 2 ("020743d44be9S9540d27b1b4bbbcfd1n21c337cb6bc9af20e

Table 3: Continue

bSa32520b393532f,"02c0120a1dda9e51a93Sd39ddd9fe0ebc
4Sea97eld27a7cbd671d5431416d3ddS7,"0213S20eb3d5f509 d743Sc9eeecb4157b2f595105e7cd564b3cdbb9ead3da41eed")
Multi Sig address:"35Z3xG92YkWSXo4ngQw6w5b3Ce6MDw94A8", Redeem Script: "5221020743d44be989540d2
7b1b4bbbcfdlm1c337cb6bc9af20eb8a32520b393532f2102c012 0a1dda9e51a93Sd39ddd9fe0ebc4Sea97e1d27a7cbd671d543141
6d3ddS7210213S20eb3d5f509d743Sc9eeecb4157b2f595105e7cd 564b3cdbb9ead3da41eed53ae"
Trial III Generating the same multi-signature using N = 1 bitcoind create multi-sig 2
("020743d44be9S9540d27b1b4bbbcfd1n21c337cb6bc9af20ebSa32520b393532f,"02c0120a1dda9e51a93Sd39ddd9fe0ebc4Sea97
eld27a7cbd671d5431416d3ddS7,"0213S20eb3d5f509d743Sc9eee cb4157b2f595105e7cd564b3cdbb9ead3da41eed")
Multi Sig address:"3PjTyDnZy8rjy!-ti00nlpSbled98TBdfJ",
RedeemScript:"5221020743d44be989540d27b1b4bb bcfdlm1c337cb6bc9af20eb8a32520b393532f2102c0120a1dda9e51a93S
d39ddd9fe0ebc4Sea97e1d27a7cbd671d5431416d3ddS7210213S20eb3d5
f509d743Sc9eeecb4157b2f595105e7cd564b3cdbb9ead3da41eed53ae"

Table 4: Public key generation and verification

Input:

Below are the public specifications for bitcoin's curve - The secp256k1
# The proven prime
Pcurve = 2**256 - 2**32 - 2**9 - 2**8 - 2**7 - 2**6 - 2**4 -1
# Number of points in the field
N=0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFF FFEBAAEDCE6AF48A03BBFD25E8CD0364141 # These two define the elliptic curve
Acurve = 0; Bcurve = 7 $y^2 = x^3 + Acurve * x + Bcurve$ Gx= 55066263022277343669578718895
168534326250603453777594175500187360389116729240 *Gy*= 326705100207588169780830851305
07043184471273380659243275938904335757337482424
# This is our generator point. Trillions of dif ones possible *G* Point = (*Gx*, *Gy*) #Individual Transaction/Personal Information privKey= 0xA0DC65FFCA799873CBEA0AC
274015B9526505DAAAED385155425F7337704883E
Extended Euclidean Algorithm/'division' in elliptic curves def modinv(a, n = Pcurve): l m, hm = 1,0
low, high = a % n,n while low> 1:
ratio = high/low
nm, new = hm-lm*ratio, high-low*ratio
lm, low, hm, high = nm, new, lm, low return lm % n
# Not true addition, invented for EC
Could have been called anything
def EC add (a, b):
LamAdd = ((b[1]- a[1]) * modinv(b[0]- a[0],Pcurve)) % Pcurve *x* = (LamAdd*LamAdd-a[0]-b[0]) % Pcurve
y = (LamAdd*(a[0]-x)-a[1]) % Pcurve return (x,y)
# This is called point doubling, also invented for EC. def ECdouble(a):
Lam = ((3*a[0]*a[0]+Acurve) * modinv((2*a[1]),Pcurve)) % Pcurve x = (Lam*Lam-2*a[0])% Pcurve
y = (Lam*(a[0]-x)-a[1]) % Pcurve return (*x*, *y*)
# This is invented EC multiplication
#Double add. Not true
EccMultiply(GenPoint,ScalarHex): if ScalarHex == 0 or ScalarHex >= N: raise Exception
("Invalid Scalar/Private Key")
ScalarBin = str(bin(ScalarHex))[2:]
Q=GenPoint for i in range (1, len(ScalarBin)):
This is invented EC multiplication.
Q=ECdouble(Q); # print "DUB", Q[0]; print if ScalarBin[i] == "1":
Q=ECadd(Q,GenPoint); print "ADD", Q[0]; print return (Q)
"******* Public Key Generation ********"
PublicKey = EccMultiply(GPoint,privKey) print printPublicKey;
"*******SignatureGeneration********" xRandSignPoint, yRandSignPoint =EccMultiply(Gx,Gy,RandNum) r = xRandSignPoint % N; print "r =", r
s = ((HashOfThingToSign + r*privKey)*
(modinv(RandNum,N))) % N; print "s =", s
"******* Signature Verification ********" w = modinv(s,N) xu1, yu1 = EccMultiply(Gx,Gy,(HashOfThingToSign * w)%N)
xu^2, yu^2 = EccMultiply(xPublicKey,yPublicKey,(r*w)%N) x,y = ECadd(xu1,yu1,xu2,yu2) if print r==x;
Signature verified
Signature not verified

Table 5: Your bitcoin-QT/D HAS 5 unspent outputs

Output 1 has 0.00156511 bitcoins, or 156511 Satoshi’s
The transaction ID for output 1 is
Oecbdl0719f54b2ac2df7a267e698bedea3f256386bldabf33e38473917d9aa5
The script pub key
76a9142516e4b2f3458b93d570ff09289ll409463c7b3988ac public address:
14P7Rkr393vWZrfXZrMizTsGP2BE9tfms8
Output 2 has 0.00052340 bitcoins, or 52340 Satoshi’s
The transaction ID for output 2 is
666a23ba57c3407296a7ald7742a62d02033cblfd5ffe64lb8ce2a537defd088
The script pub key
a914fc27b629f7796130b714267d27la8ble0bd4c2bl87 public address:
3QgRn6PsMuszHSMdrB6cmN5bNncyfGQTpi
Output 3 has has 0.00275793 bitcoins, or 275793 Satoshi’s
The transaction ID for output 3 is
6e595al20de49b009cal4207f60b08allee69efb52ac2224b2fefld830aca946 the script pub key
76a9144a3f38a5634057710lb85c748b62e7922aaa5a3b88ac public address:
17majtRM jCjpSCasbXFXWiHWAmzLTSdtga
Output 4 has 0.00155300 bitcoins, or 155300 Satoshi’s
The transaction ID for output 4 is
73926dl2845ecf0dbb5labe0d64alelldeb385dc58df2d7e2c6ed0ab3556lf8d
The script pub key
a9142a5edea39971049a540474c6a99edf0aa4074c5887 public address:
35Z3xG92YkW5Xo4ngQw6w5b3Ce6MDw94A8
Output 5 has 0.00142000 bitcoins, or 142000 Satoshi’s
The transaction ID for output 5 is bd488ce0d9bae299cfb83c418046aldc6lc79b34f07ld66aa491292400675ea3 the script pub key
76a9144a3f38a5634057710lb85c748b62e7922aaa5a3b88ac public address:
17majtRMjCjpSCasbXFXWiHWAmZLTSdtga

Table 6: Bitcoin account selected for debiting

Spend from which output? 4
The public address on that account is
35Z3xG92YkW5Xo4ngQw6w5b3Ce6MDw94A8
The address starts with the number '3'which makes it a multi-signature
All multi signatures transactions Need: txid, script Pub Key, and redeem script fortunately all of this is right there in the bitcoind 'list unspent' the transaction ID is:
73926dl2845ecf0dbb5labe0d64alelldeb385dc58df2d7e2c6ed0ab3556lf8d the script pub key is:
a9142a5edea39971049a540474c6a99edf0aa4074c5887 and only multi signatures have redeem scripts the redeem script is:
522102074Jd44be989540d27blb4bbbcfdl7721c337cb6bc9af20ebBa3
2520b393532f2102c0120aldda9e51a938d39ddd9feOebc45ea97eld27a
7cbd67ld5431416d3ddB7210213820eb3d5f509d743Bc9eoecb4157b2f
595105e7cd564b3cdbb9ead3da4leed5Jae
You have 155300 Satoshi’s in this output
How much do you want to spend? 7777!!!

Table 7: Bitcoin account selected for crediting

Send funds to which bitcoin address?
Nice!!! You chose to send funds to sean's OUT post in pensacola florida this send to 1M72SfpbzlBPpXFffz9m3CdqATR44JVaydd will leave 142062 Satoshi's in your accounts. A transaction fee of 5461 will be sent to the miners creating the raw transaction for User one-private key one

Table 8: Decode, sign and send raw transactions

Decode the raw transaction bitcoind decoderawtransaction
Ol000000018dlf5635abd06e2c7e2ddf58dc85b3dellle4ad6e04b5lb b0dcf5e84126d9273oooooooooottffffff02611eOOOOOOOOOOOOl976 a914dc863734a21Bbfe83ef770ee9d4la27f824a6e568Bacee2a02000
0000000 17a9142a5edea39971049a540474c6a99edf0aa4074c588700000000 and now we’ll sign the raw transaction. The first user gets a ’False’
This makes sense because, in multi-signatures, no single entity can sign alone
Sign the raw transaction
Bitcoind signrawtransaction:
’01000000018dlf5635abd06e2c7e2dd f58dc85b3dellle4ad6e0ab5lbb0dcf5e84l26d92730000000000ffffffff

Table 8: Continue

```
026lle0000000000001976a914dc863734a218bfe83ef770ee9d4la27f824
a6e5688acee2a020000000000l7a9142a5edea39971049a540474c6a99edf
0aa4074c588700000000”’[”txid”:”73926dl2845ecf0dbb5labe0d64a lelldeb385dc58df2d7e2c6edOab3556lf8d”,”vout”:O,”scriptPubKey”:
”a9142a5edea39971049a540474c6a99edfOaa4074c5887”,”redeem5cript”:
”5221020743d44be989540d27blb4bbbcfdl772lc337cb6bc9af20eb8a32520b3
93532f2102c0120aldda9e5la938d39ddd9fe0ebc45ea97eld27a7cbd67ld54314
16d3dd87210213820eb3d5f509d7438c9eeecb4157b2f595105e7cd564b3cdb b9ead3da4leed53ae”] ’
’[”L2MluRgdwgCotoP8prWJYYwz2zWWgsMa9TJwqARG7nFxkpdvBSsm”]’
u’hex’:u”01000000018dlf5635abd06e2c7e2d¡ff58dc85b3dellle4ad6e
0abSlbbOdcf5e84126d927300000000b50048304502100ae3b4e589dfc9d48c
b82d41008dc5fa6a86f94dSc54f9935531924602730ab8002202f88cf464414
c4ed9fallb773c5ee944f66e9b05ccle5ld97abc22ce098937ea014c6952210
243d44be989540d27blb4bbbcfdl772lc337cb6bc9af20eb8a32520b393532f
2102c0120aldda9e5la938d39ddd9fe0ebc45ea9ld27a7cbd67ld5431416d3d
d87210213820eb3dSfS09d7438c9eeecb4lis7b2f595105e7cd564b3cdbb9ea
d3da4leed53aeffffffff026lle0000000000001976a914dc863734a218bfe8
3ef770ee9d4laQ7f824a6e5688acee2ao2000000000011a9142aSedea399710
49a540474c6a99edfOaa4074c588700000000’,u ’complete’: False
Send the Raw Transaction bitcoind sendrawtransaction:
’01000000018dlf5635abd06e2c7e2d¡ff58dc85b3dellle4ad6e0abSlbbOd
cf5e84126d927300000000b50048304502100ae3b4e589dfc9d48cb82d4100
8dc5fa6a86f94dSc54f9935531924602730ab8002202f88cf464414c4ed9fa
llb773c5ee944f66e9b05ccle5ld97abc22ce098937ea014c6952210243d44
be989540d27blb4bbbcfdl772lc337cb6bc9af20eb8a32520b393532f2102c
0120aldda9e5la938d39ddd9fe0ebc45ea9ld27a7cbd67ld5431416d3dd872
10213820eb3dSfS09d7438c9eeecb4lis7b2f595105e7cd564b3cdbb9ead3d
a4leed53aeffffffff026lle0000000000001976a914dc863734a218bfe83ef
770ee9d4laQ7f824a6e5688acee2ao2000000000011a9142aSedea39971049a
540474c6a99edfOaa4074c588700000000’
Transaction ID:8e3730608c3b0bb5df54f09076e196bc292o8e39a78e73b44b6ba08c78f5cbb0
Initiating the Same Transaction leads to Declined Transaction bitcoind sendrawtransaction:
’01000000018dlf5635abd06e2c7e2d¡ff58dc85b3dellle4ad6e0abSlbbOd
cf5e84126d927300000000b50048304502100ae3b4e589dfc9d48cb82d4100
8dc5fa6a86f94dSc54f9935531924602730ab8002202f88cf464414c4ed9fa
llb773c5ee944f66e9b05ccle5ld97abc22ce098937ea014c6952210243d44
be989540d27blb4bbbcfdl772lc337cb6bc9af20eb8a32520b393532f2102c
0120aldda9e5la938d39ddd9fe0ebc45ea9ld27a7cbd67ld5431416d3dd872
10213820eb3dSfS09d7438c9eeecb4lis7b2f595105e7cd564b3cdbb9ead3d
a4leed53aeffffffff026lle0000000000001976a914dc863734a218bfe83ef
770ee9d4laQ7f824a6e5688acee2ao2000000000011a9142aSedea39971049a
540474c6a99edfOaa4074c588700000000’
Error: "code":-22,"message":"TX rejected"
```

Table 9: Features of nodes used in our research

Type of node	Processor	CPU type	CPU speed	RAM	Operating system
Raspberry pi HP	ARM CPU	64 bits	1.20 GHz	8GB	Rasbian 5.10
Laptop	Intel core i3	64 bits	1.99 GHz	4GB	Windows 10

Table 10: Elliptic curve point generation times in milli seconds

	A	B	P	N	p	2p	Time in ms
Creating 2P from P	0	7	37	2	(6, 1)	(18, 17)	0.00013
Creating nP from P	0	7	37	2	(3, 16)	(35, 31)	0.00025
Creating 2P from P with real curves	0	7	37	38	(00, 09) (00, 28) (03, 16) (03, 21) (04, 16) (04, 21) (05, 13) (05, 24) (06, 01) (06, 36) (08, 01) (08, 36) (09, 12) (09, 25) (12, 12) (12, 25) (13, 13) (13, 24) (16, 12) (16, 25) (17, 6) (17, 31) (18, 17) (18, 20) (19, 13) (19, 24) (22, 6) (22, 31) (23, 1) (23, 36)	NA	0.00038

Table 10: Continue

					(24, 17) (24, 20) (30, 16) (30, 21) (32, 17) (32, 20) (35, 6) (35, 31)		
ECC point addition	0	7	37	2	$P_1(6, 1)$ $P_2(8, 1)$ $P_1 + P_2$ (23, 36)	NA	0.00018
ECC point addition with range of points					(4, 16) $P_1 + P_2 = (0, 28)$ $P_1 = (6, 1)$ $P_2 = (5, 13)$ $P_1 + P_2 = (22, 6)$ $P_1 = (6, 1)$ $P_2 =$ (6, 1) $P_1 + P_2 = (18, 17)$ $P_1 = (6, 1)$ $P_2 =$ (8, 1) $P_1 + P_2 = (23, 36)$ $P_1 = (6, 1)$ $P_2 = (9, 12)$ $P_1 + P_2 = (19, 13)$ $P_1 = (6, 1)$ $P_2 = (12, 12)$ $P_1 + P_2 = (9, 12)$ $P_1 = (6, 1)$ $P_2 = (13, 13)$ $P_1 + P_2 = (30, 16)$ $P_1 = (6, 1)$ $P_2 = (16, 12)$ $P_1 + P_2 = (4, 16)$ $P_1 = (6, 1)$ $P_2 = (17, 6)$ $P_1 + P_2 = (35, 6)$ $P_1 = (6, 1)$ $P_2 = (18, 17)$ $P_1 + P_2 = (23, 1)$ $P_1 = (6, 1)$ $P_2 = (19, 13)$ $P_1 + P_2 = (3, 16)$ $P_1 = (6, 1)$ $P_2 = (22, 6)$ $P_1 + P_2 = (13, 13)$ $P_1 = (6, 1)$ $P_2 = (23, 1)$ $P_1 + P_2 = (8, 36)$ $P_1 = (6, 1)$ $P_2 = (24, 17)$ $P_1 + P_2 = (32, 17)$ $P_1 = (6, 1)$ $P_2 = (30, 16)$ $P_1 + P_2 = (17, 6)$ $P_1 = (6, 1)$ $P_2 = (32, 17)$ $P_1 + P_2 = (32, 20)$ $P_1 = (6, 1)$ $P_2 = (35, 6)$ $P_1 + P_2 = (12, 12)$	NA	0.00023

B. Credit-Debit Bitcoin Multi-Signature Address

This section deals with spending bitcoins from 2 of 3 multi-signature accounts live on a bitcoin network. Implementation is done using python 2.7.6, which relies on bitcoind and bitcoin RPC.

We had to rewrite the original python library connection.py to take advantage of multi signatures. Initially, we make an object call to the bitcoind blockchain available on localhost. As bitcoind has all the updated transactions on my local hard disk. The size of the updated bitcoin transactions while writing this study is 30 gigabytes and will be growing day to day. Downloading the entire blockchain takes a couple of days depending on the network bandwidth and disk space. In the previous section, we generated three brand new private keys as shown in Table 1. Any time you generate a new address in bitcoind or bitcoin qt client it stores in a wallet.dat file that is hidden on your computer system. One can always use these private keys. To send from any bitcoin address one needs to have his respective private keys. The best practice is to create all three private keys on three different machines. Even used by three different people. All three keys should never be on the same machine.

We will be sending back the received bitcoins/Sathoshi's to generate a multi-signature account generated in the previous section. We set the minimum transaction amount of 0.00005461 Sathoshi as the transaction fee for miners. Later we query how many bitcoins are available in our user accounts, where users can have multiple accounts. Table 5 displays five unspent user accounts with bitcoins/Sathoshi's in each account, transaction IDs, and script public key on public address. One can notice that user accounts 1,3 and 5 are not multi-signature accounts as they don't start with 1 and account 2 and 4 is a multi-signature account. In our case multi signature account 4 is selected as debiting account which displays the transaction ID, script public key, and redeems the script. The minimum we can spend is 5460 sathoshi's any number higher than that should form a valid transaction. We send 7777 sathoshi's to a charity organization at sean's outpost in pensacola florida will leave the balance in the account and the transaction fee will be sent to the miners.

Table 8 demonstrates decoding, signing, and sending the raw transactions, where user 1 uses his private key to sign the decoded raw transaction with a false ability to spend and sends it to any one of the remaining two users. Instead of our user 1 private key, we have an elliptic curve coded digital signature user 2 or user 3 will not be able to change the amount or who the transaction is destined to. The only thing they can do is decide whether to sign the transaction or not. As well the transactions will get added level of security with online wallet providers who administer your accounts with features like daily limits, and preferred merchants then they will sign the transaction. Once the transaction is signed by any two users then it will have the true ability to spend on it. Finally, one among them can decide to spend the bitcoins. If we ever try to spend from the transaction which is already spent, then the transaction will be rejected. Figure 5 demonstrates the flow chart to generate a multi-signature account. Table 5 displays unspent outputs. Table 6 displays the bitcoin account selected for debiting. Table 7 displays the bitcoin account selected for crediting. Figure 2 demonstrates flow chart spending from multi-signature using bitcoin. Figure 4 demonstrates sending bitcoins to a multi-signature account using a coin base. Figure 5 demonstrates transaction details on the blockchain.

C. Elliptic Curve Cryptography-Generation of Public Key to Sign the Bitcoin Multi-Signature Transaction

In this section, we will be discussing the generation of the public key to sign the transactions generated in section B using elliptic curves. Elliptic curve cryptosystems were first proposed in 1985 by Neil Koblitz and Victor Miller (Sakkari, 2022). The heart of elliptic cryptosystems is based upon a discrete logarithm problem. A discrete logarithm problem is defined by Q $\cancel{\in}$generator point G, find x such that $Q = XP$. An elliptic curve is a curve of form $y^2 = x^3 + ax + b$, over a finite field F_p where p is prime. Several points on an elliptic curve are a finite set of points within the field F. Let us consider an elliptic curve defined over $y^2 = x^3 + ax + b$ where, $a = 2$, $b = 3$, and modulo prime field $p = 5$. The values of a, b, and prime p are random. (Sakkari and Ulla, 2022a-b; Guicheng and Zhen, 2013; Kodali and Naikoti, 2016):

$$\begin{aligned}
x &= 0 \Rightarrow y^2 = 3 \Rightarrow \textit{No Solution}(\text{Mod } 5)\\
x &= 1 \Rightarrow y^2 = 6 \Rightarrow 1y = (1,4)(\text{Mod } 5)\\
x &= 2 \Rightarrow y^2 = 15 \Rightarrow 0y = 0(\text{Mod } 5)\\
x &= 3 \Rightarrow y^2 = 36 \Rightarrow 1y = (1,4)\ (\text{Mod } 5)\\
x &= 4 \Rightarrow y^2 = 75 \Rightarrow 0y = 0(\text{Mod } 5)
\end{aligned}$$

Then points on the elliptic curve are (1,1), (1,4), (2,0), (3,1), (3,4), (4,0). The recommended bitcoin curve by NIST is SECP256K1. Figure 9 demonstrates the flow chart for elliptic curve cryptography for signing and verifying bitcoin for public key generation and verification and the recommended elliptic curve design parameter constants that one needs to follow to use the SECP256K1 curve. Elliptic curve operations used for our study are point addition, point double, point multiply, and extended euclidean algorithm or mod-inverse or division. The two fundamental operations on an elliptic curve are explained below.

Point Addition

Draw a line that crosses P and Q at two places on the elliptic curve P (x_1, x_2) and Q (x_2, y_2), and label the third point on the curve as R. (x_3, $-y_3$). We create a perpendicular line from R (x_3, $-y_3$) that intersects the curve once more at the addition of two points $P + Q$ and is shown as R in Fig. 6. ECC-point addition coordinates:

- Let $X_R = X_3$ and $Y_R = -Y_3$
- $X_R = S^2 - x_1 - x_2 \bmod P$
- $Y_R = S(x_1 - X_R) - y_1 \bmod P$
- $S = (y_2 - y_1)/(x_2 - x_1) \bmod P$
- S= Slope of line through P (x_1, y_1) and Q (x_2, y_2)

Point Doubling

In point doubling, a tangent line is drawn that touches the elliptic curve's point P but does not pass through it, intersecting the curve at a different location. From the point of intersection, which we refer to as $P + P$ or $2.P$, *a* vertical line is drawn. The results are $3.P$, $4.P$, $5.P$, and $6.P$. As seen in Fig. 7, this is also referred to as integer multiplication. An Abilean group is formed by the points on an elliptic curve, and the resulting points can be joined together or multiplied by an integer:

$$\begin{aligned}
&\text{i.e.,} P + Q = Q + P\\
&3.P + P = 2.P + 2.P\\
&5.(7.P) = 7.(5.P)
\end{aligned}$$

Point doubling is very fast, to go from a point P to $100\cdot P$, we need only eight Steps rather than ninety-nine steps as shown below:

$$\begin{aligned}
&i.e, P \to 2.P \quad 12.P \to 24.P\\
&2.P \to 3.P \quad 24.P \to 25.P\\
&3.P \to 6.P \quad 25.P \to 50.P\\
&6.P \to 12.P \quad 50.P \to 100.P
\end{aligned}$$

If you try to multiply integers of 50 digits or more the same procedure is followed.

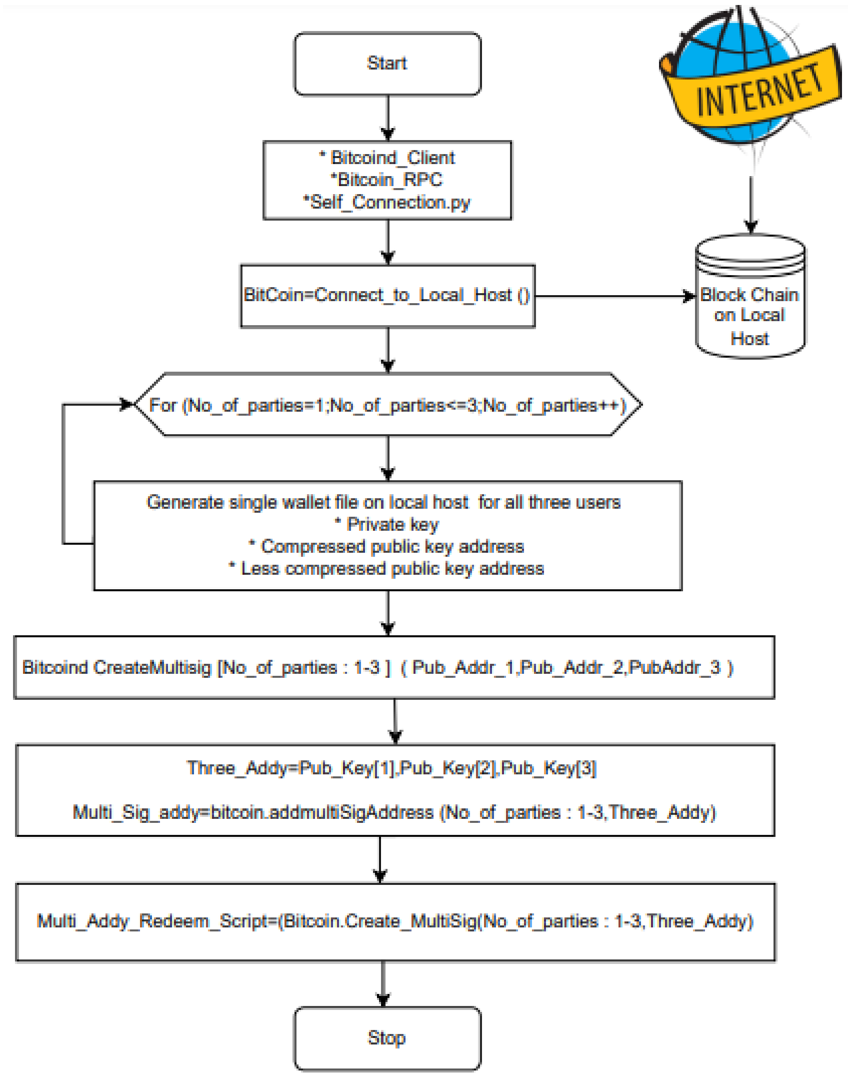

Start
INTERNET
* Bitcoind_Client
*Bitcoin_RPC
*Self_Connection.py
BitCoin=Connect_to_Local_Host ()
Block Chain on Local Host
For (No_of_parties=1;No_of_parties<=3;No_of_parties++)
Generate single wallet file on local host for all three users
* Private key
* Compressed public key address
* Less compressed public key address
Bitcoind CreateMultisig [No_of_parties : 1-3] (Pub_Addr_1,Pub_Addr_2,PubAddr_3)
Three_Addy=Pub_Key[1],Pub_Key[2],Pub_Key[3]
Multi_Sig_addy=bitcoin.addmultiSigAddress (No_of_parties : 1-3,Three_Addy)
Multi_Addy_Redeem_Script=(Bitcoin.Create_MultiSig(No_of_parties : 1-3,Three_Addy)
Stop

Fig. 2: Flow chart-generation of multi-signature

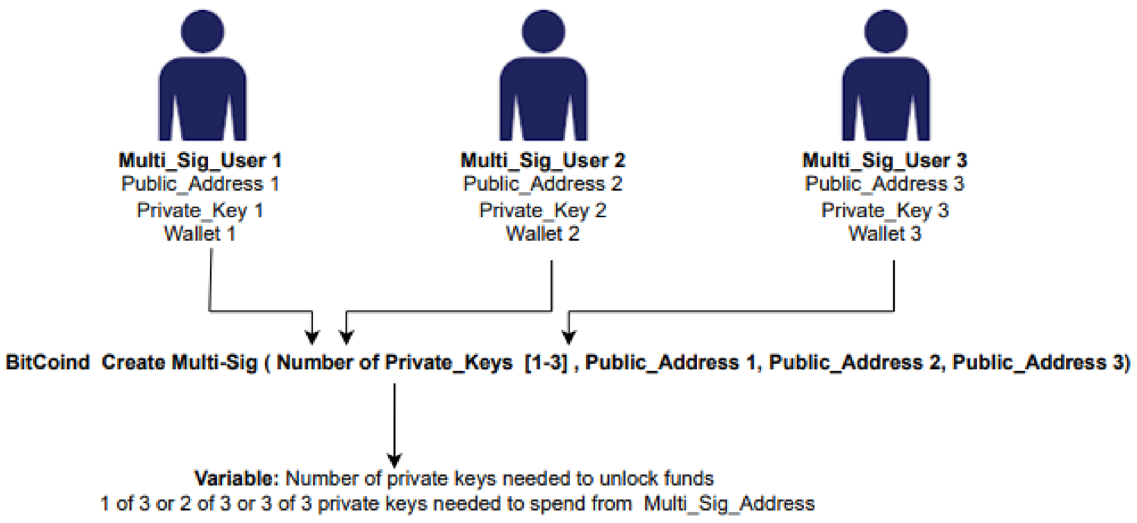
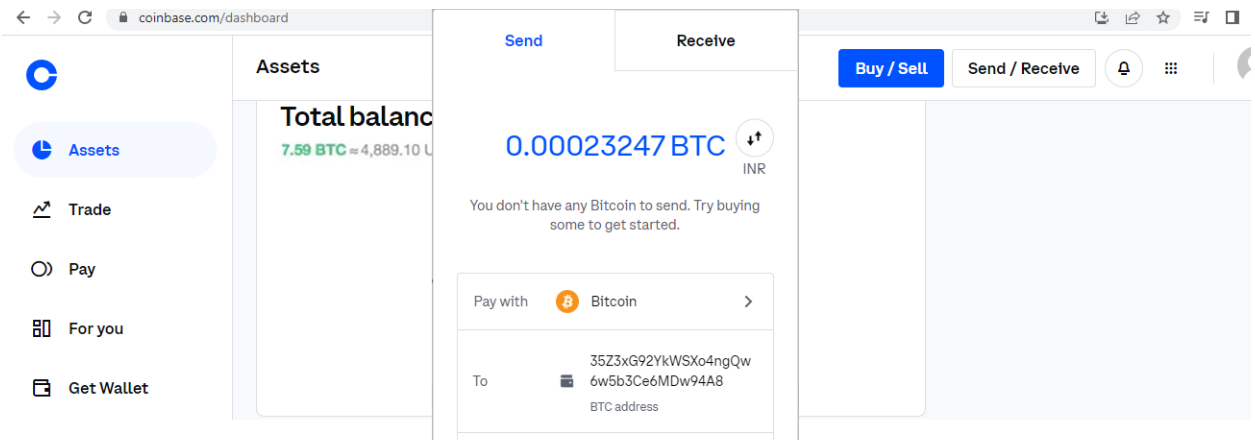

Multi_Sig_User 1
Public_Address 1
Private_Key 1
Wallet 1
Multi_Sig_User 2
Public_Address 2
Private_Key 2
Wallet 2
Multi_Sig_User 3
Public_Address 3
Private_Key 3
Wallet 3
BitCoind Create Multi-Sig (Number of Private_Keys [1-3] , Public_Address 1, Public_Address 2, Public_Address 3)
Variable: Number of private keys needed to unlock funds
1 of 3 or 2 of 3 or 3 of 3 private keys needed to spend from Multi_Sig_Address

Fig. 3: Generation of 2:3 multi-signature account

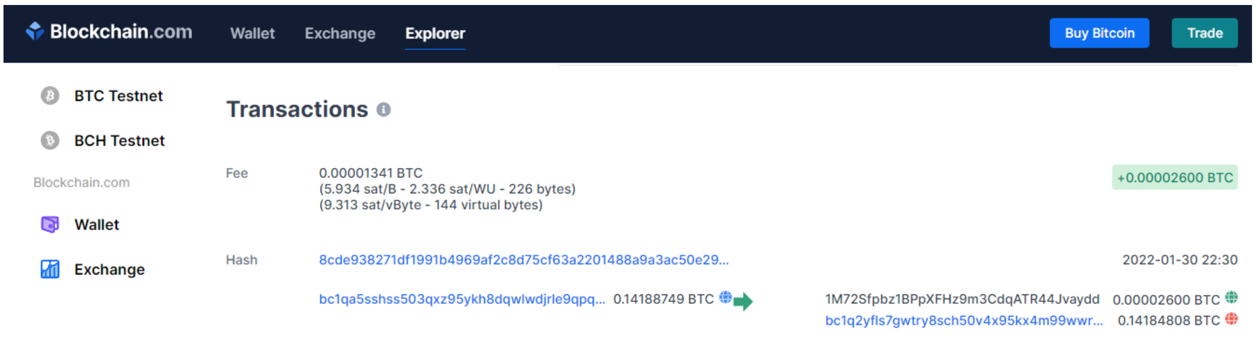


0.00023247 BTC

You don't have any Bitcoin to send. Try buying some to get started.

35Z3xG92YkWSXo4ngQw6w5b3Ce6MDw94A8
BTC address

Fig. 4: Sending bitcoins to generated multi-signature address using coin base (debiting)



Transactions

Fee 0.00001341 BTC
(5.934 sat/B - 2.336 sat/WU - 226 bytes)
(9.313 sat/vByte - 144 virtual bytes)

+0.00002600 BTC

Hash 8cde938271df1991b4969af2c8d75cf63a2201488a9a3ac50e29...

2022-01-30 22:30

bc1qa5sshss503qxz95ykh8dqwlwdjrle9qpq... 0.14188749 BTC

1M72Sfpbz1BPpXFHz9m3CdqATR44Jvaydd 0.00002600 BTC
bc1q2yfls7gwtry8sch50v4x95kx4m99wwr... 0.14184808 BTC

Fig. 5: Transaction details on the block chain

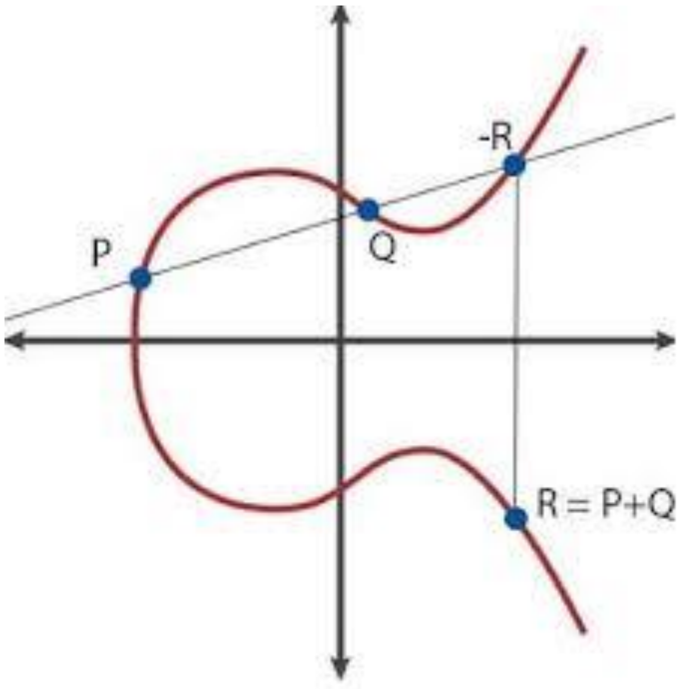


Fig. 6: Point addition

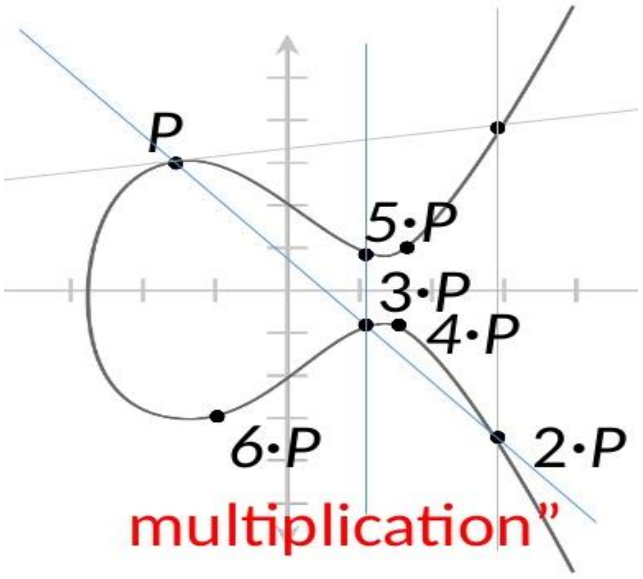
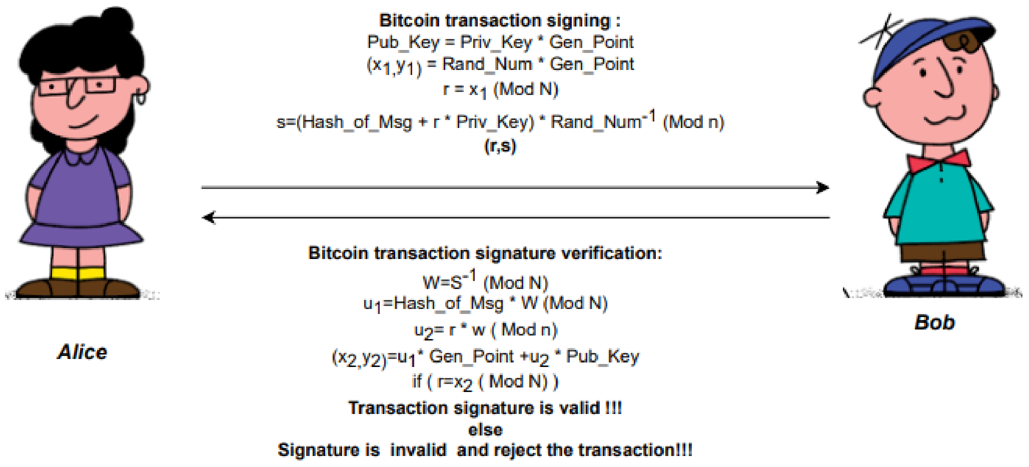


Fig. 7: Point doubling



Bitcoin transaction signing :
Pub_Key = Priv_Key * Gen_Point
(x_1,y_1) = Rand_Num * Gen_Point
r = x_1 (Mod N)
s=(Hash_of_Msg + r * Priv_Key) * Rand_Num^{-1} (Mod n)
(r,s)

Bitcoin transaction signature verification:
W=S^{-1} (Mod N)
u_1=Hash_of_Msg * W (Mod N)
u_2= r * w (Mod n)
(x_2,y_2)=u_1* Gen_Point +u_2 * Pub_Key
if (r=x_2 (Mod N))
Transaction signature is valid !!!
else
Signature is invalid and reject the transaction!!!

Fig. 8: Elliptic curve signing and verifying bitcoin transactions

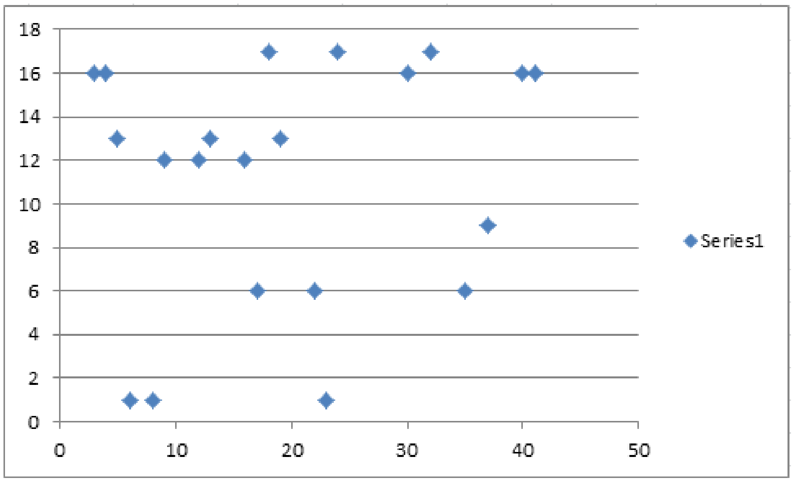


Fig. 9: Creating 2.*P* from *P* on elliptic curve when *A* = 0, *B* = 7, *P* = 37, *N* = 20

ECC-Point Doubling Coordinates

- Let $X_R = X_3$ and $Y_R = -Y_3$
- $X_R = S^2 - 2\,X_1 \bmod P$
- $Y_R = S\,(x_1 - X_R) - y_1 \bmod P$
- $S = (3\,X_1^2 + a)/(2y_1) \bmod P$
- S = Slope of tangent line through $P\,(x_1, y_1)$

Initially, the public key is obtained by multiplying the private key with the generator point without using SHA256. Remember that the elliptic curve public key is not as same as the public address generated in section *B*. For elliptic curve cryptography, you need prime *p*, *a*, *b* which defines the curve, the generator point $G\,(G_x,G_y)$, *n* the number of points in the field, and *h* is the co-factor.

D. Elliptic Curve Cryptography-Signing and Verifying the Bitcoin Transactions

In this section, we focus on signing the transaction and verifying without having the private key. By simply having the public key, a hash of the message, and all the elliptic curve parameters one verifies whether the message was sent with a private key i.e., Only the sender has the private key and the receiving party is unaware of the private key. The public key in ECC is simply an (x, y) coordinate on the elliptic curve which is the result of the multiplication of generator point *G* on the curve with the private key. Table 4 shows the steps involved in public key generation and verification of bitcoin transactions. Each of the steps in Table 4 is explained below:

- Here we have predefined six constants that are specific to SECP256k1 which is the actual curve bitcoin uses. Where Pcurve is the prime number SECP256k1 uses, *N* is the valid number of points on the field. The acyclic property of elliptic curves makes them unique i.e., when we multiply the original generator point *G*, *N* times we reach the infinity point which is also the beginning. We know elliptic curves are defined by the equation of form $y^2 = x^3 + ax + b$ if we have a generator point we don't need to have all the information of the line or the curve. For example, suppose we have point on the line and the slope of the line we do not need to know where the line intersects on the *x* or *y* axis same is true when we are dealing with curves. G_x, G_y is the *x* coordinate and the *y* coordinate is the generator points on the elliptic curves. *H* is the co-factor and another domain coefficient of this curve and will always be one. The private key, random number, and hashofthingstosign vary for each transaction. The hashofthingstosign is the SHA256 hash of the actual message. In terms, of bitcoin hashofthingstosign will always be the transaction itself
- In step two we apply the fundamental operations of elliptic curves like point addition- is not true addition rather invented for elliptic curves here we add point *P* with point *Q*. Point doubling-, point multiplication- which is, in turn, achieved using point addition and point doubling and mod inverse-also known as extended euclidean algorithm or the division in elliptic curves

Steps involved in ECC signature generation are as follows:

1) Pubkey = privKey*genpoint
2) (x_1, y_1) = randnum*genpoint
3) $r = x_1 (\bmod N)$,
4) $s = (Hashofmsg + r * privkey) * rand\,Num^{-1} (\bmod N)$, $\therefore$ *signatureis* (r,s)

Steps involved in ECC signature verification are as follows:

1) $w = s^{-1} (\bmod N)$
2) $u_1 = Hashof\,Msg * W (\bmod N)$
3) $u_2 = r * w (\bmod N)$,
4) $(x_2, y_2) = u_1 *$ gen point + $u_2 *$ pub key
5) if $r = x_2 (\bmod N)$

   $\therefore$ signatureisvalid

Table 4 shows public key generation and verification using elliptic curve cryptography and Fig. 8 shows the process of elliptic curve signing and verifying bitcoin transactions.

Results and Discussion

For a finite field elliptic curve of the form $y^2 = x^3 + ax + b$ with a determined prime number (*p*) we compute 2 *P* if we have a point *P* and use this to find *nP*, where n is the number of times we add *P*. Table 10 demonstrates elliptic curve point generation times in milliseconds to Create 2*P* from *P*, *nP* from *P*, 2*P* from *P* with real curves- Curve25519 Montgomery with *x*-points 9 10 respectively, ECC point addition with a couple and ECC point addition with a range of points. Figure 9 demonstrates creating 2*P* from *P* on an elliptic curve when $A = 0$, $B = 7$, $P = 37$, and $N = 20$. $B = 7$, $P = 37$, $N = 20$. Table 9 shows features of nodes used in our research.

Conclusion

This study aims to investigate the security of bitcoin wallets, how users manage their private keys, and illustrate the application of asymmetric key cryptography using elliptic curves. The results from a survey of the top 100 bitcoin users show that their security is at stake due to the use of non-multi-signature accounts. The results show that only 34.5% of bitcoin users use multi-signature accounts. Many bitcoin users still back up their keys on a piece of paper and remaining who back up their keys do not encrypt their keys. For future work, we could bring out the vulnerabilities with other digital currencies like Ethereum and Ripple by conducting similar surveys by considering the correlation study on the security of wallets, features of wallets, and their adoption methods.

Acknowledgment

The authors would like to acknowledge the support provided by Presidency University- Bengaluru, India.

Funding Information

Authors have received a grant of 50,000 (Fifty Thousand Indian Rupees) from Research and Innovation Council, Presidency University- Bangalore 56006.

Author's Contributions

Mohammed Mujeer Ulla: Contributed in all experiments, coordinated the data-analysis and contributed to the writing of the manuscript.

Deepak S. Sakkari: Designed the research plan and organized the study.

Ethics

This article does not contain any studies with human participants or animals performed by any of the authors.

References

Boneh, D., & Venkatesan, R. (1996, August). The hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In Annual International Cryptology Conference (pp. 129-142). *Springer, Berlin, Heidelberg*. https://doi.org/10.1007/3-540-68697-5_11

Breitner, J., & Heninger, N. (2019, February). Biased nonce sense: Lattice attacks against weak ECDSA signatures in cryptocurrencies. *In International Conference on Financial Cryptography and Data Security* (pp. 3-20). *Springer, Cham*. https://doi.org/10.1007/978-3-030-32101-7_1

Guicheng, S., & Zhen, Y. (2013, October). Application of elliptic curve cryptography in node authentication of internet of things. *In 2013 Ninth International Conference on Intelligent Information Hiding and Multimedia Signal Processing* (pp. 452-455). IEEE. https://doi.org/10.1109/IIH-MSP.2013.118

Hammi, B., Fayad, A., Khatoun, R., Zeadally, S., & Begriche, Y. (2020). A lightweight ECC-based authentication scheme for Internet of Things (IoT). *IEEE Systems Journal*, 14(3), 3440-3450. https://doi.org/10.1109/JSYST.2020.2970167

Khan, M. A., Quasim, M. T., Alghamdi, N. S., & Khan, M. Y. (2020). A secure framework for authentication and encryption using improved ECC for IoT-based medical sensor data. *IEEE Access*, 8, 52018-52027.

Kodali, R. K., & Naikoti, A. (2016, December). The ECDH-based security model for IoT using ESP8266. *In 2016 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT*) (pp. 629-633). *IEEE*. https://doi.org/10.1109/ICCICCT.2016.7988026l0.30l0.30

Ouni, N., & Bouallegue, R. (2016). Performance and complexity analysis of a reduced iterations LLL algorithm. arXiv preprint arXiv:1607.03272.

Park, Y., & Park, J. (2016). Analysis of the upper bound on the complexity of the LLL algorithm. *Journal of the Korean Society for Industrial and Applied Mathematics*, 20(2), 107-121. https://doi.org/10.12941/jksiam.2016.20.107

Patel, C., & Doshi, N. (2020). Secure lightweight key exchange using ECC for user-gateway paradigm. IEEE Transactions on Computers, 70(11), 1789-1803. https://doi.org/10.1109/TC.2020.3026027

Sakkari, D. S. (2022). Review on Insight into Elliptic Curve Cryptography. In Modern Approaches in Machine Learning & Cognitive Science: A Walkthrough (pp. 81-93). *Springer, Cham*. https://doi.org/10.1007/978-3-030-96634-8_8

Sakkari, D. S., & Ulla, M. M. (2022a). Design and Implementation of Identifying Points on Elliptic Curve Efficiently Using Java. In Modern Approaches in Machine Learning & Cognitive Science: A Walkthrough (pp. 95-105). *Springer*, Cham. https://doi.org/ 10.1007/978-3-030-96634-88."

Sakkari, D. S., & Ulla, M. M. (2022b). Design and Implementation of Elliptic Curve Digital Signature Using BitCoin Curves Secp256K1 and Secp384R1 for Base10 and Base16 Using Java. *In Innovation in Electrical Power Engineering, Communication, and Computing Technology* (pp. 323-337). Springer, *Singapore*. https://doi.org/10.1007/978-981-16-7076-328

Shaikh, J. R., Nenova, M., Iliev, G., & Valkova-Jarvis, Z. (2017, November). Analysis of standard elliptic curves for the implementation of elliptic curve cryptography in resource-constrained E-commerce applications. *In 2017 IEE International Conference on Microwaves, Antennas, Communications and Electronic Systems* (COMCAS) (pp. 1-4). IEEE. https://doi.org/10.1109/COMCAS.2017.8244805

Zhang, X., & Wang, X. (2018). Digital image encryption algorithm based on elliptic curve public cryptosystem. *IEEE Access*, 6, 70025-70034. https://doi.org/10.1109/ACCESS.2018.2879844