

Primains

A proposal to make blockchain technology
more accessible

CONTENTS

Abstract.....	3
1. Basic Functionality	3
2. Pointer based Primains Implementation	3
2.1 Theory of pointer based Primain generation:.....	3
2.2 The wordlist	4
2.3 Wordlist interchangeability	4
2.4 Primain generation code	5
2.5 Theory of converting a Primain to a public key	5
2.6 Primain conversion code	6
2.7 Implementing Primain systems on existing chains	6
2.8 Scalability of pointer based systems.....	6
3.Hash based Primains Implementation	7
3.1 Theory of hash based Primain generation	7
3.2 The Wordlist	7
3.3 Theory of hash based Primain to key conversion	7
3.4 Hash based Primain generation code	8
3.5 Hash based Primain to key conversion code	8
3.5 Scalability of hash based Primain systems.....	9
4. Differences between the systems	9
5. Primain number usage within a chain.....	9
5.1 Scalability Issues	9
5.2 Breaking points	10
5.3 Architectural changes	10
5.4 Funding transaction based validation within a Primain chain	11
6. Possible attack vectors and their solutions.....	11
6.1 Full to light node/wallet communication	11
6.2 Malicious wordlists	12
7. Conclusion	12
References	12
Table of figures.....	12

ABSTRACT

With a growing user base blockchain technology is on the verge of receiving mass adoption. One of the biggest challenges regarding broad adoption is a current lack of user friendliness. Compared to centralized finance applications with smaller, easier to manually verify addresses most blockchains require the input of either a 324 character long base 16 number¹ or a 30 character long base 58 number². Although easily copied to the clipboard, these (to the human eye) random seeming strings are very hard to memorize or manually check. The aim of this paper is to propose an easy to implement, fast and lightweight method of generating easy to check, memorize and share addresses for any blockchain using either RSA or ECC. These new addresses consist of a combination of 3 words connected by dots (e.g. junior.rule.urban) or a small integer. Utilizing the integer version of the Primain within the chain could offer an opportunity to save big amounts of space with only a minimal increase in computational cost.

1. BASIC FUNCTIONALITY

With two different Primain systems (hash and pointer based) scalability and practical applications vary, whilst the core premise stays the same. This very core premise lays within improved user friendliness as RSA/ECC public keys or classic crypto addresses no longer need to be copy pasted into a transaction recipient field. They can be replaced by 3 words interconnected by dots, which in turn enables manual checking and even remembering of own “addresses”. These much shorter keys can also be utilized in the blockchain itself to save on chain space.

2. POINTER BASED PRIMAINS IMPLEMENTATION

The following section will firstly explain the theory of the two needed algorithms, then provide a python code implementation of the previously elaborated algorithms.

2.1 THEORY OF POINTER BASED PRIMAIN GENERATION:

Taking a hex key as an input, the Primain generation function starts by retrieving the users first received transaction, takes the transactions index as well as the block index in which the transaction is stored. These two numbers can then be connected to form one big Primain number, defined by a standard template. These templates enable flexible limits on block number and size. Whilst an Ethereum like blockchain might need only a maximum of 99 transactions in a block, a blockchain with a slower block time (e.g. bitcoin) could benefit from an upper limit of 9999 transactions per block.

Visualizing a template with a limit of 999 transactions per block as t and a limit of 999999 blocks as b, the Primain number would be interpreted as follows:

bbbb bbbb tttt

1641 8033 0045

Block index: 16418033

Transaction index: 45

This big number then gets broken down into three smaller numbers (bbbb.bbbb.tttt) which then get replaced by their corresponding wordlist word (abominable.insert.accent) (see Figure 1).

¹ (Kaliski, 2006)

² (Nakamoto, 2008)

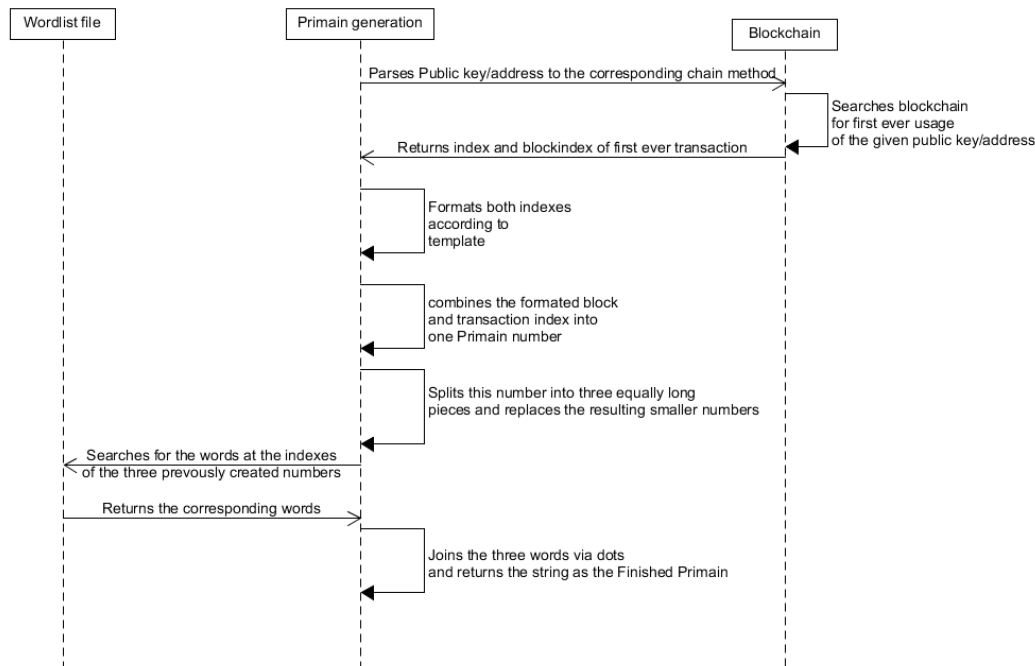


FIGURE 1

2.2 THE WORDLIST

To ensure user-friendly Primain generation, the list of possible words should only contain as easy to memorize, write and speak words as possible. As pointer based Primains rely on combining the indexes of the Primain words into one big number, the length of the wordlist should be of the power of 10. For the below demonstration a wordlist of the 10.000 most common English words is used.

2.3 WORDLIST INTERCHANGEABILITY

As the wordlist words only represent their indexes, actual wordlists do not need to be identical as long as the list length is equal. This allows for the construction of a multilingual or even non lingual Primain system.

Multilingual Primain systems can greatly improve accessibility to non-native English speakers without causing any major computational expenses. As Primains are always generated on the users node (or the chosen relay node) changing the Primains language would be as simple as changing the wordlist file for your own chosen version. This process however should be heavily standardized to ensure safe usage as every node needs to possess every Wordlist for every language which should not significantly raise hardware requirements as a 10k Wordlist only equates to around 100 kb of disk space. Operating on a multilingual system would be as simple as checking in which wordlist the first word is present and then proceeding same as before from that point on. Cross list duplicates should however be avoided at all cost as everything over three cross list duplicates could cause a Primain to be interpreted from the wrong list, resulting in wrong indexes and ultimately a wrong recipient.

As the wordlists actual items are not of any real importance other than not being duplicates of each other, even a non-lingual Primain system could be established and connected to a lingual one. Taking a Unicode based system as an example, words could be replaced by 2 emoticons per word or for systems with a “wordlist” length of 1k a whole word could even be replaced by a single emoticon. With Unicode 15.0 consisting of over 300 different Unicode Blocks non lingual Primain systems are only constraint by their lack in practical user-friendliness.

2.4 PRIMAIN GENERATION CODE

```
def generate_primain(tx_index,block_index):
    #Adding zeros in front of numbers smaller than 1000
    # to ensure proper formatting (4&8 being the our template)
    while len(str(tx_index)) < 4:
        tx_index=f'{0}{tx_index}'
    while len(str(block_index)) < 8:
        block_index=f'{0}{block_index}'
    #Creating the primain number
    primain_number=f'{block_index}{tx_index}'
    #Splitting the number in 4 digits long parts
    parts = [int(primain_number[i:i+4])
              for i in range(0, len(primain_number), 4)]
    #Replacing the numbers with the Wordlist words
    # (removing any capitalisations)
    for c,number in enumerate(parts):
        parts[c]=WORDS[int(parts[c])].casefold()
    #Combining the list of 3 words into a Primain
    primain='.'.join(parts)
    return primain
```

2.5 THEORY OF CONVERTING A PRIMAIN TO A PUBLIC KEY

The converting algorithm takes a Primain as input and returns the transaction index and the block index. This is achieved by firstly converting the three words into their representative wordlist indexes and then joining these together, forming the Primain number, which can then be interpreted depending on the template.

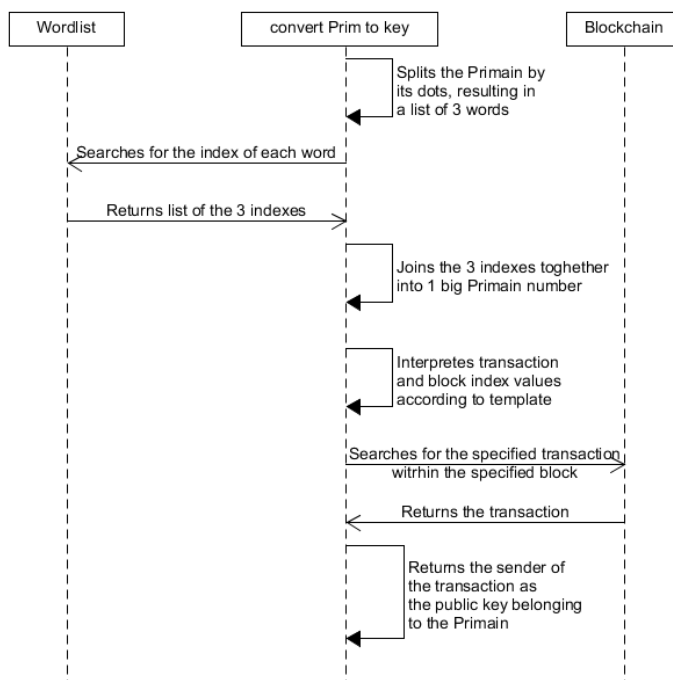


FIGURE 2

2.6 PRIMAIN CONVERSION CODE

```
def get_index_from_prim_(prim):  
    #Replacing each word in the primain by its  
    # wordlist index and joining them into the Primain number  
    primnumber=''.join([WORDS.index(i.upper())  
                        for i in prim.split('.')])  
    #Taking the transaction and block index values  
    # from the Primain number (4&8) being our template  
    txindex=primnumber[-4:]  
    blocknumber=primnumber[0:-4]  
    return blocknumber,txindex
```

2.7 IMPLEMENTING PRIMAIN SYSTEMS ON EXISTING CHAINS

Due to the pointer-based nature of Primains, the implementation of such systems onto an existing blockchain network merely require the addition of a few searching functions into the protocol. These could however even be outsourced to an external program as long as this program has access to a maintained chain. This allows even small minorities to utilize the benefits of Primains without having to fork the actual blockchain protocol. The only threat regarding such smaller sub systems could be a lack of wordlist standardization (see 2.1.1 Wordlist interchangeability) resulting in poor connectability between said subnets.

2.8 SCALABILITY OF POINTER BASED SYSTEMS

Considering a wordlist length of 10.000, the limit of possible Primains in the system would be 9999.9999.9999, this limit however also includes regular transactions, putting the practical maximum amount of Primains far below 9999.9999.9999. To combat this inherent scalability weakness, an overflow system can be established, in which all Primains get reset after reaching the limit of transactions. When implemented accordingly such a system would be infinitely scalable.

Should extreme numbers of transactions make reset times of the above described overflow method impractical, other measures of scalability can still be enforced. These include the addition of a single digit number in front of the three Primain words (decreasing reset time by a factor of 9), increasing the number of words from 3 to 4 (decreasing reset times by a factor of 999) and the usage of a 100k word list (decreasing reset times by a factor of 999). These scalability improvements can be combined and stacked as needed, it should, however, be considered that their usage also represents a significant impact on user friendliness.

The above-mentioned numbers and principles however only apply to systems which have not had architectural changes to accommodate better Primain scalability³. Should the proposed block header attribute however be successfully implemented, scalability issues regarding maximum user numbers will be easily resolved as 100% of the possible word combinations can be utilized for wallets instead of also including transactions. This takes makes user numbers of just under 1 trillion possible, each one having a unique Primain

³ See 5.3 Architectural changes

3.HASH BASED PRIMAINS IMPLEMENTATION

The following section will firstly explain the theory of the two needed algorithms, then provide a python code implementation of the previously elaborated algorithms.

3.1 THEORY OF HASH BASED PRIMAIN GENERATION

Taking a hexadecimal public key as an input, the Primain generation algorithm firstly converts this hex key into a decimal integer. After the conversion the algorithm starts at the 68th index of the integer (skipping all identifier digits) and checks if the joined numbers from index 68 to index 72 are smaller than the length of the proposed wordlist (3000 words). If this is not the case the algorithm will move on to the next 4 digits and check their value, this cycle is repeated until 3 numbers smaller than the length of the wordlist have been yielded. These number will then be replaced by the words on the corresponding wordlist index. The resulting list of 3 words is then finally connected with 2 dots (leaving no spaces) and returned.

Due to taking a variable input length (different key types), whilst outputting a constant number of words, this type of Primain generation is comparable to a primitive hash function, therefore the system will be referred to as “hash-based”. This however results in the possibility of hash collisions, which in turn would result in wrongful transaction recipients or could even break a backend Primain system⁴. To combat this breaking point either a strictly direct conversion direction or a no duplicate policy could be utilized. The latter is however favorable as just favoring earlier Primains could result in wallets without Primain capabilities or even manipulation attacks⁵.

3.2 THE WORDLIST

Due to very similar requirements regarding the selection of words⁶ and the management of multi- or non-lingual Primain systems⁷ hash based Wordlist are quite comparable to their pointer based counter parts. As such, all overlapping requirements will not be elaborated on again in this section.

The only key difference between the two Wordlist lies within their lengths, as hash-based Primains skip over the numbers they cannot represent with their Wordlist, whilst pointer based addresses are forced to accurately depict a single big integer. This trade enables hash-based systems to strike a better balance between scalability and user friendliness as they do not have to deal with increments to the power of 10.

3.3 THEORY OF HASH BASED PRIMAIN TO KEY CONVERSION

The converting algorithm takes a Primain and a list of all possible keys as an input. Starting by converting every hex key to a decimal number and appending the new number to a list of all the decimal keys, the algorithm continues by looping through every key in the decimal list and checks if the Primain generated for the key matches the inputted Primain. If the two Primains are in fact identical, the corresponding decimal key will get converted to its previous hexadecimal encoding and returned.

⁴ See 5.2 Breaking points

⁵ See 6 Attack vectors and their solutions

⁶ See 2.1.2 The Wordlist

⁷ See 2.1.3 Wordlist interchangeability

3.4 HASH BASED PRIMAIN GENERATION CODE

```
def generate_primain(public_key):
    '''Turns an inputted key into a valid Primain'''
    #Words are present in the form of a list constant
    words=WORDS
    #Converting the key to decimal
    k = int(public_key, 16)
    #Setting the starting indexes
    v1=68
    v2=72
    numberlist=[]
    #Starting the loop which constrcuts the number,
    while True:
        num=str(k)[v1:v2]
        #checks if its smaller then the length of the wordlist
        if int(num) < len(words):
            #if its smaller it gets appended and the loop moves on
            #to the next number
            numberlist.append(int(num))
            v1+=4
            v2+=4
            #After accumulating 3 numbers the loop is broken
            if len(numberlist) > 2:
                break
        else:
            #else the loop moves on by 4 indexes
            v1+=4
            v2+=4
    #replacing the numbers with the corresponding words
    for c,number in enumerate(numberlist):
        numberlist[c]=words[numberlist[c]]
    #Building the final string by jojing the words of the numberlist
    primain='.'.join(numberlist)
    return primain
```

3.5 HASH BASED PRIMAIN TO KEY CONVERSION CODE

```
def get_key_from_prim(prim,keys):
    '''If inputted a valid Primain this function returns the corresponding key
    (inputs=primain,keys)'''

    #Looping through the hex keys
    for k in keys:
        #Generating a primain from each key and comparing it to the inputted
        prim2=generate_primain1(k)
        if prim == prim2:
            #Returning the hex version of the decimal key
            return k
```

3.5 SCALABILITY OF HASH BASED PRIMAIN SYSTEMS

Although scalability is still a very real issue, there are a few ways to greatly improve a systems maximum user amount, size and speed. The biggest scalability concern is the computationally expensive conversion from a Primain to a hex key, this however could be battled with the already quite common practice of sharding to reduce the size of keys to loop through. Another method could be probabilistic searching, starting from recipients and senders of all to the user connected transactions, then moving through the chain from newer to older blocks. The least concerning scalability limit would be the occurrence of duplicates as this issue can easily be solved by increasing the wordlist length. Adding only one more word to the 3000 words long list would result in 27 million more possible Primains whilst increasing the Primain length from 3 to 4 words would increase the possible number of Primains to 81 trillion.

4. DIFFERENCES BETWEEN THE SYSTEMS

With speed and scalability regarding a maximum amount of Primain users, pointer based Primains are very versatile and practically viable. They are the perfect fit for most average blockchain networks with a high number of individual wallets. Their only inherent flaw is their scalability issues regarding bigger amounts of transactions, this however is where hash based Primains come into place as their scalability is not affected by transaction numbers. With them being only confined by the maximum number of keys in the system and the resulting increase in computational expense., hash based Primains are perfect for sub 1 million user systems with extreme numbers of transactions. These might be found in layer 2 solutions or private or heavily automated blockchains.

What should also be noted is that hash based Primains do not need Wordlist lengths to the power of 10, this can aid in improving user friendliness by tailing the wordlist length to the exact requirements of the chain. Due to this versatility, hash based wordlists can also be official standards such as BIP-,39,32,44⁸ and others.

5. PRIMAIN NUMBER USAGE WITHIN A CHAIN

Due to a lack of cryptographic properties, utilizing Primains or their respective Primain numbers within a blockchain brings on numerous challenges. If implemented correctly however implementation could greatly improve blockchain space efficiency.

Accessing the cryptographic properties of a Primain number would have to be delayed to after the Primain to key transformation has taken place (no matter if hash or pointer based.) The protocol should however take into account that no matter the method (hash or pointer based conversion), the first mention of a public key cannot occur as a Primain as this would inhibit the Prim- to key conversion process.

5.1 SCALABILITY ISSUES

The following analysis of Scalability Issues will mostly refer to pointer based systems as hash based systems would struggle with astronomical computational costs if implemented in a system with a very high amount of users.

Whilst front end usage of Primains faces minimal risk from hitting scalability limits, as those Primains are only used in the seconds between the input of a Primain and the conversion to its corresponding key, Primains utilized within the immutable blockchain itself have to connect to their corresponding key for an indefinite amount of time.

⁸ (Bitcoin developer team, 2022)

This can lead to catastrophic consequences should an overflow system not be implemented properly as wrongful Prim- to key bindings could result in double spending and invalid transactions, effectively breaking the whole chain. These disastrous consequences can be avoided by ensuring that Primains are only transformed to keys within the bounds of their scalability limit⁹. This forces protocols to basically restart the Primain system with each overflow epoch, as cross epoch referencing would not be possible without increasing the Primain numbers length, which could either be done in accordance with the front end Primain system, effectively merging the linked epochs or the increased Primain number length could only be backend based, enabling a longer epoch time with the goal of saving even more blockchain space.

5.2 BREAKING POINTS

Assuming all security measurements¹⁰ are met, there should not be any additional risks arising with implementing a backend Primain system. What should, however, be considered is the fact that the outcome of a successful attack would be far more devastating compared to a front end system. As a successful attack would most likely result in huge amount of double spending, draining of uninvolved wallets (no matter if hot or cold) and general loss of trust in the network. Whereas front end systems are mainly concerned with the manipulation of single transactions.

5.3 ARCHITECTURAL CHANGES

Although both pointer as well as hash based systems do not require core architectural changes, besides the implementation of the proposed safety measures¹¹, pointer based systems could potentially profit from the implementation of a new block header attribute. This new attribute would contain all new keys, which have never been previously used in the chain, this would allow for unlimited amounts of transactions as only active wallets would take up Primains.

This however comes at the cost of extra space, as the additional block header attribute would take up considerable space, especially due to the fact that first transactions would still have to contain the full original key, as the key owner will not be able to know their own index within the new attribute of the header, which makes signing the transaction without knowledge of the sender Primain impossible. This would, however mean that Primains could reach their full scalability potential regarding maximum number of users, practically eliminating the need for further scalability measurements.

For average systems, implementation of such a new block header attribute could enable extremely user-friendly Primains with wordlist lengths of only 1000 (resulting in 1 billion possible users) or even only 100-word long wordlists (resulting in 1 million possible users).

The actual make up of such an attribute could either be a simple list with all new keys or, to further save blockchain space, a list of transaction indexes referring to the key already saved in the transaction. It should however be mentioned that with every extra refer (such as presented in the header attribute) validation becomes more computationally expensive and as such also slower.

⁹ See 2.8 Scalability of pointer-based Systems

¹⁰ See 6. Possible attack vectors and their solutions

¹¹ See 6. Possible attack vectors and their solutions

5.4 FUNDING TRANSACTION BASED VALIDATION WITHIN A PRIMAIN CHAIN

For systems which rely on the referencing of funding transactions to validate transactions on chain usage of pointer based Primains could greatly improve space efficiency of the chain as long hashes could be replaced by shorter Primain numbers. As the Primain system works very similar to a hash map in the sense that accessing of the value (Primain to key conversion) can be done in $O(1)$ time.

This however is not directly compatible with the in 5.3 mentioned architectural changes, as should those be implemented in the described manner, transactions will no longer be linkable to Primain numbers, which makes referencing funding transactions via Primains impossible. To combat this, a second, completely independent, only numeral Primain system could be established, which would be structurally similar to the pointer based Primain system proposed before. Its dimensions should however be tailored toward expected transaction numbers, as longer Primain numbers take up more precious space.

Should the amount of transactions reach extreme highs, an overflow system similar to the one described in 2.8 could be implemented in which old transactions can only be referenced by their hash. Old transactions could still be validated as their Primains can be interpreted according to the cycle which was present at the time of publishing. This means results in the fact that although multiple transactions are represented by the same Primain number, only one can be referenced as a funding transaction by its Primain. As it is clearly defined after how many blocks the Primains start at 0 again, each past Primain can be interpreted within its corresponding epoch, but outside of the epoch the transaction can only be referenced by hash.

6. POSSIBLE ATTACK VECTORS AND THEIR SOLUTIONS

As with every decentralized system attacks should not only be considered but rather expected. As both previously described Primain systems could suffer from massive vulnerabilities if implemented incorrectly, the subchapters below will elaborate on possible attack vectors and secure solutions to those vectors.

6.1 FULL TO LIGHT NODE/WALLET COMMUNICATION

As the majority of blockchain users do not run full nodes, a big number of participants rely on full nodes to publish their transactions and receive on chain data. As every Primain conversion method requires access to the chain, Primain conversions have to be outsourced to full nodes as well. This however enables the risk for manipulating transaction recipients by returning a wrongful Primain conversion. As the non-full node would need the blockchain to verify the conversion, funds could easily be redirected to another wallet.

For pointer based Systems this vulnerability can be easily combated, by implementing the transmission of a Merkle (or vercle) proof of the transaction the Primain refers to. This Merkle tree could prove the inclusion of the referred transaction within a certain block, whilst the block index could be compared to the index encoded in the Primain. For full security the same metric can be applied to the transaction index within the block. Combined with the inherent cost of filling a new wallet each block, this attack vector can be effectively neutralized by the above-mentioned security measurements.

Hash based systems could be regarded as the cryptographically safer Primain system as their hash like nature enables even light nodes to check their validity with minimal computational cost. Every received Primain to key conversion can be easily checked by converting the key back to the Primain and comparing the result to the original Primain. The security of this system is derived from a protocol wide no duplicate policy regarding Primains as duplicates could be used to target specific wallets and redirect their transactions.

6.2 MALICIOUS WORDLISTS

As the words contained within a Primain only represent their indexes, manipulation of the wordlist could result in the ability to directly influence the recipient of a transaction. This attack vector is especially dangerous as it cannot be combated by above mentioned security measurements. However, the threat can easily be mitigated, even for multilingual Primain systems, by providing reference hashes of all official wordlists. Even light nodes/wallets can easily hash their own wordlist and compare it to one of the official hashes. This attack vector and the derived security measurements apply to pointer based as well as hash based Primains.

7. CONCLUSION

Adding a user-friendly way to sending and receiving of funds, which is also available for non-English native speakers, could revolutionize blockchain accessibility. Due to the two different proposed Primain systems even huge networks could implement Primain systems tailored to their specific needs, without facing inherent scalability issues. Especially the construction of multilingual Primain systems could greatly improve intercultural collaboration and wealth transfer on blockchain mediums. Combining this with the easy to implement nature of this technology, even regarding already existing chains, Primains can be expected to gain in popularity over the coming crypto cycles, but they can more importantly also be expected to help bridge the gap between new high-tech blockchain innovations and the average user.

REFERENCES

- Bitcoin developer team. (2022, July 26). *Github bitcoin repository*. Retrieved from Github: <https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki>
- Danziger, P. (2010). Big O Notation. *Google Scholar*.
- Kaliski, B. (2006). The Mathematics of the RSA Public-Key Cryptosystem. *Google scholar*.
- L. Luu, V. N. (2016). A Secure Sharding Protocol for Open Blockchains. *ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*.
- Nakamoto, S. (2008, October 31). Bitcoin: A Peer-to-Peer Electronic Cash System.

TABLE OF FIGURES

Figure 1	4
Figure 2	5