

**ALTERNATIVE DIGITAL SIGNATURE SCHEMES
IN BLOCKCHAIN**

**BLOK ZİNCİRDE
ALTERNATİF DİJİTAL İMZA ŞEMALARI**

FAHRETTİN YAVUZYİĞİT

ASSOC. PROF. DR. OĞUZ YAYLA

Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Mathematics

2019

This work named "**ALTERNATIVE DIGITAL SIGNATURE SCHEMES IN BLOCKCHAIN**"
by **Fahrettin YAVUZYIĞİT** has been approved as a thesis for the Degree of **MASTER OF
SCIENCE IN MATHEMATICS** by the below mentioned Examining Committee Members.


Prof. Dr. Evrim AKALAN

Head



Assoc. Prof. Dr. Oğuz YAYLA

Supervisor



Assoc. Prof. Dr. Murat CENK

Member



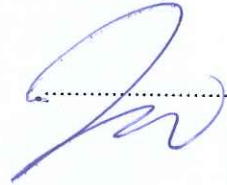
Assoc. Prof. Dr. Sedat AKLEYLEK

Member



Assist. Prof. Dr. Adnan ÖZSOY

Member



This thesis has been approved as a thesis for the Degree of **MASTER OF SCIENCE IN
MATHEMATICS** by Board of Directors of the Institute for Graduate School of Science
and Engineering.

Prof. Dr. Menemşe GÜMÜŞDERELİOĞLU

Director of the Institute of
Graduate School of Science and Engineering

To the memory of my mother Kamile YAVUZYIĞİT

ETHICS

In this thesis study, prepared in accordance with the spelling rules of Institute of Graduate Studies in Science of Hacettepe University.

I declare that

- all the information and documents have been obtained in the base of the academic rules
- all audio-visual and written information and results have been presented according to the rules of scientific ethics
- in case of using other Works, related studies have been cited in accordance with the scientific standards
- all cited studies have been fully referenced
- I did not do any distortion in the data set
- and any part of this thesis has not been presented as another thesis study at this or any other university.

27/06/2019



Fahrettin YAVUZYİĞİT

YAYINLAMA VE FİKRİ MÜLKİYET HAKLARI BEYANI

Enstitü tarafından onaylanan lisansüstü tezimin/raporumun tamamını veya herhangi bir kısmını, basılı (kağıt) ve elektronik formatta arşivleme ve aşağıda verilen koşullarla kullanıma açma iznini Hacettepe üniversitesine verdiğimi bildiririm. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanım hakları bana ait olacaktır.

Tezin kendi orijinal çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanması zorunlu metinlerin yazılı izin alarak kullanıldığını ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.

Yükseköğretim Kurulu tarafından yayınlanan **“Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge”** kapsamında tezim aşağıda belirtilen koşullar haricince YÖK Ulusal Tez Merkezi / H. Ü. Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

- Enstitü / Fakülte yönetim kurulu kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren 2 yıl ertelenmiştir.
- Enstitü / Fakülte yönetim kurulu gerekçeli kararı ile tezimin erişime açılması mezuniyet tarihimden itibaren ay ertelenmiştir.
- Tezim ile ilgili gizlilik kararı verilmiştir.

27/06/2019



Fahrettin YAVUZYİĞİT

ABSTRACT

ALTERNATIVE DIGITAL SIGNATURE SCHEMES IN BLOCKCHAIN

Fahrettin YAVUZYIĞİT

Master of Science, Department of Mathematics

Supervisor: Assoc. Prof. Dr. Oğuz YAYLA

June 2019, 51 pages

This thesis is a study on authentication methods of Bitcoin ecosystem which is an application of blockchain. Firstly, the facilities given by Bitcoin along with its security requirements are examined from a perspective that can be used in place of today's banking system. Bitcoin ecosystem is a use case of blockchain and its current transaction authentication methods are well studied in the literature, and based on this, the questions are raised, "Could the common account and the proxy concepts of banking services also be possible in Bitcoin?". Many researches say that there are solutions based on bilinear pairings commonly, accountable subgroup multi-signature and delectable credentials schemes. These schemas are studied in this thesis. As a first alternative, accountable subgroup multi-signature (ASM) constructed from Boneh – Lynn – Shacham (BLS) signature schemes gives us the opportunity of the public key aggregation mechanism. In this way, it can be possible that more than one user sign the same message jointly and only one public key is needed to verify the signature. This approach allows lots of savings in storage of public keys in transaction scripts and that is quite convenient to implement in Bitcoin. For the second alternative, delectable credentials signature schema renders possibility that someone can give her signing authority to another. Delegatable credentials can do this with a structure that is built on Groth and Schnorr signature schemes. Furthermore, the suggested delectable credential schema in this thesis is able to store in secret or disclosed within the signature. This property is included in the usage scenarios about need of delegation for individuals.

Keywords: Block Chain, Bitcoin, transaction, authentication, multi signature, accountable subgroup multi signature, delegatable credentials.

ÖZET

BLOK ZİNCİRDE ALTERNATİF DİJİTAL İMZA ŞEMALARI

Fahrettin YAVUZYİĞİT

Yüksek Lisans, Matematik Bölümü

Tez Danışmanı: Doç. Dr. Oğuz Yayla

Haziran 2019, 51 sayfa

Bu tez, bir blok zincir uygulaması olan Bitcoin ekosisteminin kimlik doğrulama yöntemleri üzerine bir araştırmadır. İlk olarak, bugünün bankacılık sistemi yerine kullanılabilirliği bakış açısıyla, güvenlik gereksinimlerinin yanı sıra Bitcoinin sağladığı olanaklar sorgulanmıştır. Bir blok zincir uygulaması örneği olan Blockchain ve kendisine ait mevcut kimlik doğrulama metotları araştırılmış, elde edilen bilgiler üzerinden "Bankacılık hizmetlerinde bulunan ortak hesap ve vekalet kavramları aynı zamanda Bitcoin de mümkün müdür?" sorusu ortaya atılmıştır. Cevap olabilecek birçok araştırma arasından, en yeni çalışmalardan olması ve ortak şekilde bilinear parings kullanıyor olmaları sebebiyle hesap verebilir altgrup çoklu imza ve devredilebilir yetki şemaları detaylı şekilde araştırılmıştır. İlk öneri olarak, hesap verebilir altgrup çoklu imza yöntemi, açık anahtarların birleştirilmesine olanak sağlayan Boneh – Lynn – Shacham (BLS) imzalama şeması üzerine inşa edilmiştir. Bu sayede birden fazla kullanıcının aynı mesajı ortak şekilde imzalaması ve bu imzanın yalnız bir açık anahtar ile doğrulanması mümkün olabilmektedir. Bu yaklaşım, işlemler içinde açık anahtarların saklanması tasarruf sağlamaya ve Bitcoin'e uyarlanmasının oldukça kolay olmasını sağlamaktadır. İkincisi için çözüm olarak, delege edilebilir kimlikli imzalama yöntemi birinin sahip olduğu imzalama yetkisini bir başkasına devredebilmesine olanak sağlamaktadır. Delege edilebilir kimlik bu olanğı Groth ve Schnorr imzalama yapıları üzerine kurulan bir yapı ile gerçekleştirilebilmektedir. Aynı zamanda bu tezde yer alan delege edilebilir kimlik şeması kullanıcı özbilgilerini açıkta veya gizli şekilde saklayabilmektedir. Bu özellik, tezimizin bireysel delegasyon ihtiyacı üzerindeki kullanım örneğine dahil edilmiştir.

Anahtar Kelimeler: Blok zincir, Bitcoin, işlem, kimlik doğrulama, çoklu imza, hesap verebilir altgrup çoklu imzalama, delege edilebilir kimlik.

ACKNOWLEDGEMENT

First of all, I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Oğuz YAYLA for his invaluable suggestions, motivation, patience, and continuous support in this thesis. His guidance helped me in all the time of research and writing of this thesis.

Besides my supervisor, I would like to thank the rest of my thesis committee: Prof. Dr. Evrim AKALAN, Assoc. Prof. Dr. Murat CENK, Assoc. Prof. Dr. Sedat AKLEYLEK and Assist. Prof. Dr. Adnan ÖZSOY for their encouragement, insightful comments.

I would like to my special thank to my teacher dear Prof. Dr. Haşmet GÜRÇAY for always believing and supporting me from bachelor to master degree.

Also, I'm grateful that my wife and daughter are in my life and with me in this process. Thank you.

Fahrettin YAVUZYİĞİT

June 2019, Ankara

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT	i
ÖZET	iii
ACKNOWLEDGEMENT	v
TABLE OF CONTENTS	vi
NOTATIONS	x
1 INTRODUCTION	1
1.1 Motivation for Writing This Thesis	1
1.2 Previous Works	1
1.3 An Application of Blockchain, Bitcoin	2
1.4 Suggestions for Bitcoin Transaction Authentication	3
1.5 Outline of The Thesis	4
2 BLOCKCHAIN	5
2.1 Bitcoin	6
2.2 Block Structure	7
2.3 Transaction Structure	8
2.4 Transaction Scripts	9
2.4.1 Pay To Public Key (P2PK)	10
2.4.2 Pay To Public Key Hash (P2PKH)	12
2.4.3 Multisig (M-of-N Multi-Signature)	14
3 ACCOUNTABLE SUBGROUP MULTI-SIGNATURE	16
3.1 Bilinear Groups and Pairing Based Cryptography	16
3.2 Construction of ASM Schema	18
3.3 Proof-of-Possession	21
3.4 Accountable-Subgroup Scheme with PoPs	22
3.5 Application ASM to Bitcoin	23
4 DELEGATABLE CREDENTIALS WITH HIDEABLE ATTRIBUTES	25
4.1 Practical Delegatable Anonymous Credentials System	26
4.2 Signature Schemes	26
4.2.1 Groth Structure-Preserving Signature Schema	27
4.2.2 Sibling Signatures	28

4.2.3 Constructing Sibling Signatures.	29
4.3 Construction for Delegatable Credentials	30
4.4 Application of Delegatable Credentials to Bitcoin	33
5 CONCLUSION	35
REFERENCES	37
APPENDIX	41
CURRICULUM VITAE	51

FIGURES

Figure 2.1. Converting public key to Bitcoin address	7
Figure 2.2. Structure of a block	7
Figure 2.3. Structure of the header	8
Figure 2.4. Structure of a transaction	8
Figure 2.5. Structure of a transaction output	9
Figure 2.6. The transaction input structure	10
Figure 2.7. The unlocking and locking script combination	10
Figure 2.8. The P2PK script flow	11
Figure 2.9. The P2PKH script flow	13
Figure 2.10. The m-of-n multisig script flow	15
Figure 4.1. The delegation chart	30

LISTINGS

1 The Magma Codes of ASM Example	41
2 The Magma Codes of ASM with PoPs Example	45
3 The pseudocode of presenting	49
4 The pseudocode of verifying	50

NOTATIONS AND ACRONYMS

Notations

\mathbb{Z}	Integers
\mathcal{G}	Bilinear group generator
\mathbb{G}	Cyclic group
e	Bilinear map
\mathbb{F}	Finite field
H	Hash algorithm
$\$$	Randomly generated

Acronyms

SHA-256	Secure Hash Algorithm with 256 bit version
RIPMD-160	RACE Integrity Primitives Evaluation Message Diges with 160 bit
OP_CHECKSIG	Operator that checks the signature validity
OP_DUP	Operator that duplicates the top stack element
OP_EQUAL	Operator that checks either top two stack elements are equal
OP_HASH160	Operator that calculates SHA-256 + RIPMD-160 hash
OP_0	Operator that pushes an empty array to the stack
OP_CHECKMULTISIG	Operator that checks the signatures validity
ASM	Accountable Subgroup Multi-Signatures

1 INTRODUCTION

1.1 Motivation for Writing This Thesis

Commercial relations, started with barter economy, have been continuing through banking system that is most commonly used today. But in a globalized world, Bitcoin, a new payment method brought by developing techniques and technologies in addition to different needs, has a potential to replace the banking system we know. Although Bitcoin is currently only a payment instrument, it may assume other functions in the banking system or come other similar systems in the future.

The current Bitcoin system, used as a payment method, hosts some of the opportunities that are accepted in the banking system. For example, Multi-signature, which allows Bitcoin transactions to be signed by multiple users, corresponds to the common account in banking system. In this thesis, studies on using Multisig, that is a type of transaction scripts, more effectively and efficiently are examined. Also, the delegatable credential, which may be a candidate as a signing method in Bitcoin system because of being equivalent to the transactions carried out by proxy in banking, is studied.

1.2 Previous Works

Most used one of transaction scripts in Bitcoin ecosystem, even if Multisig can compress the signature value, storing all public keys still take the major size in transactions. Multi-signatures have been worked on different mathematical bases e.g. on RSA [19, 20], discrete logarithm problem [21, 22, 5, 23, 6], pairings [24, 25, 3], and lattices [26]. To reduce transaction sizes, the aggregating public-key algorithm has been clearly handled only in [6] and [3]. The "public-key aggregation" is named firstly in the work of Maxwell [6]. With the public-key aggregation, it can be enough that only the aggregated value of all public keys is stored in transaction instead of them separately. According to [6], in signing stage, there is a need for two round of transmission between all signers. But one round of transmission can be enough by [3]. With the name of "Accountable subgroup multi-signatures (ASM)", this approach was firstly worked by Micali, Ohta and Reyzin [5]. In addition to not having key

aggregation algorithm, it bases on discrete logarithm problem and constructed from [18]. Boneh, Drijvers and Neven [3] take it to the next level via adding key aggregation and using parings further.

There are some works on delegatable anonymous credentials [27, 28, 29, 30]. Chase and Lysyanskaya [29] uses generic zero-knowledge proofs. By this approach, the size must grow exponentially by the number of delegators. Therefore, this complexity limits the delegation process and using this method practically is not possible. "Hierarchical group signatures" by Trolin and Wikström [31] are an addition to the group signatures. This methods gives the opportunity that includes a hierarchy of group managers. Users can sign a message for the group with the credential of any managers. This property can be considered as a "delegatable credentials". But, it differs from the suggested one in this thesis, because users are able to do as a manager or as a user, not both of them, as only one. And, the managers can disclose the signers in contrary to the anonymity.

1.3 An Application of Blockchain, Bitcoin

Blockchain is a kind of data structure. It can store data and connects one to another by linking. To link the blocks, hash algorithm SHA-256 which is a cryptographic, namely mathematical, method is used. Informally, lets think three number of blocks (Block-X, Block-Y, Block-Z). The hash value of whole Block-X is stored in the Block-Y, and using the same, whole Block-Y's hash value is stored in the Block-Z. In this way, Block-X is linked to Block-Y and Block-Y is linked to Block-Z. If there is an any change in Block-X, because of that it results in change of Block-X's hash value, Block-Y, which holds the hash of Block-X, will also be altered. Likewise, any change in Block-Y affects Block-Z. In a situation where there is a huge number of blocks, altering must be done from first changed block to the last one and this process may be almost impossible in some specific hashing conditions.

Bitcoin ecosystem bases on this limitation. The blocks store data about the information that an amount of Bitcoin have been transferred someone to another by the name of "transaction". With the unchangeable data structure property, Bitcoin ecosystem gives an assurance to users about keeping their assets. But there is another issue, which is authenticating transactions. It may be more important that an amount of transaction is spent or transferred by real owner

than storing it safely. To authenticate transactions, another cryptographic method, digital signing, is used. Unlike the general usage of digital sign, the ownership of a digital asset in Bitcoin is not proved via signed by owner of asset, merely it must be signed by the previous owner and new owner's public key must be stored in it. To illustrate this, the paper check example can be given. Using paper check payment method, signer is not owner of the money currently, owner is whose name has been written on "Pay to the order of" field. In Bitcoin system, to spend amount of Bitcoin, owner must give a valid signature that is signed using the private key matching the public key which is stored in previous transaction. This approach allows two benefits, privacy and decentralization of authentication process. Because of that no-one knows whose transaction specific public key is, privacy of Bitcoin ownership is provided. And, there is not any need for certificate authority, so Bitcoin transactions are decentralized. There is an only assurance for reliance the Bitcoin system. That is strong and efficient transaction signing and validation mechanism. Bitcoin ecosystem has many kinds of transaction authentication standards. Some of those are "Pay To Public Key (P2PK)", "Pay To Public Key Hash (P2PKH)" and "Multisig".

1.4 Suggestions for Bitcoin Transaction Authentication

Multisig is used for multi-ownership of Bitcoin amount or extra security needs by adding another factor of authentication like Bitcoin wallet etc. in case of stolen individuals' private keys. Multisig has a bulky structure. It is described in Section 2.4.3 in detail. From this point, "Accountable Subgroup Multi-Signature (ASM)" is a potential solution for improving multi-signature need of Bitcoin. Rather than having necessity of keeping all public keys in Multisig, ASM gives the opportunity that only one aggregated key which is the combination of all signers' public key can be enough to store in transaction. This gives a lot of data and computational saving. In addition, accountability property of ASM provides another security need. Owing to this property, all group members are responsible for the transactions signed by a subgroup. In this way, the possibility of other group members repudiating the transactions, committed by the subgroup of users, is eliminated.

The suggested ASM schema in [3] is constructed from "Boneh – Lynn – Shacham (BLS)" signature scheme [17] which is based on bilinear pairings. ASM schema has additional two

stages "Key Aggregation" and "Group Setup" to common signing protocols. With Key Aggregation stage, the aggregated public key " apk " is generated. In Group Setup, for all group members, the membership keys " mk_i " obtained from all members' public and secure key are generated. So, the accountability is provided because the membership keys are needed in Signing stage.

Delegatable credentials is the suggestion that is not currently in Bitcoin ecosystem but can be a solution to diversity needs of Bitcoin transaction authentications. As an example, to visualize a need, let a firm make their payments through the Bitcoin. If the firm's payments is needed to be done by more than one person, payment processes must be divided to departments or employers. To do this, the firm either should share the Bitcoin confidential information like secret key and/or wallet ID and password or authorize the responsible persons via delegating the owner's credential.

The suggested schema [16] has four stages "Setup", "Delegate", "Present" and "Verify". Setup is used for parameter generation, Delegate stage is for delegate the credential owned to another user, Present is signing a message or etc. by a delegate and generates the signature by the name "attribute token" in this schema, Verify is matching the signature and message with the attribute token value and only first delegator's public key, described in detail in Section 4.3. To the schema, any wanted attributes - e.g. name, gender, age, working position, authority level etc. - of all delegates can be stored into attribute token or they can be hidden. In this work, Groth and Schonorr signatures are combined and this combination is used with sibling signature method, a new method first given in [16]

1.5 Outline of The Thesis

The rest of the thesis is organized as follows. In Chapter 2, the concept of blockchain is introduced and Bitcoin ecosystem, a case study of blockchain, is briefly described up to the transactions stage in a view of how its transaction authentication mechanisms work. In Chapter 3, accountable subgroup multi-signature schema, which is a suggestion instead of Bitcoin multi-signature, is given in detail. In Chapter 4, as a new authentication mechanism that is likely to be included in Bitcoin ecosystem, delegatable credentials with concealable attributes is suggested. Finally, the conclusions are included in Chapter 5.

2 BLOCKCHAIN

The blockchain is a data structure that is ordered and linked list of blocks. Blocks are linked back each one referring to the previous one. In this way, the linked blocks create a chain, that is why we call it blockchain. The blockchain data can be stored in a file like a simple text file and also in a database. The blockchain can be considered as a stack that is left to right, with blocks linked to top of next one and the initial block being the base of chain. In blockchain terminology, “height” means the number of blocks from the initial block, and “top” (or “tip”) is the most recent block.

In blockchain structure, the identifier of each block is its hash value, generated using cryptographic hash algorithm SHA256 and stored in the header of the block. Besides, every block refers the previous one, with the name of parent block, and the block header also has the parent block hash. Connecting every block to its predecessor with hash values creates a chain, that goes back the initial block.

It is possible that there can be more than one block whose parent’s hash value is same. Namely, a block can possess more than one child. However, having only one parent hash value field, there can be only one parent block. If a block has multiple children, that is known as blockchain “fork,” which comes accidentally when different miners connects one each new block to the same almost meanwhile. Finally, only one block is accepted as a part of the chain and hence the fork situation is eliminated.

Since the predecessor block hash is stored in the block header, it affects the hash value of current block. This means that if predecessor’s hash value changes, the child’s hash value also changes. Altering the any value of parent block causes to change the predecessor’s hash. The predecessor’s changed hash requires to change the previous block link of the successor which is the hash value of it. This also results in the successor’s hash to alter, which requires a change in the link of the two-next successor, which in order altering it, and so on. This cascade effect ensures that when a block has many followers, it cannot be altered without recalculation of all successor blocks. This needs huge computational power, so being a long chain of blocks gives us the unreversibility property of the blockchain, that is one of the most important properties of blockchain’s security [1].

2.1 Bitcoin

Having the most common usage in this area, Bitcoin is an application of blockchain. Blocks are used for storing Bitcoin transactions that contains the information about sent amount of Bitcoin from someone to someone. Bitcoin is a cryptocurrency ecosystem on its own. It gives much opportunity to use money in a different way that we did ever. To illustrate, lets see the similarity of Bitcoin transactions to banking checks. The Bitcoin address is for the receiver identification, the beneficiary in banking terms, that is written on “Pay to the order of”. The checks do not require a specific account, it can be the name of a bank account holder, corporations, institutions, or even cash. The Bitcoin transaction usage is similar, the Bitcoin address gives the bitcoin ecosystem flexibility. A Bitcoin address represents whose the public key is.

Bitcoin Addresses

The Bitcoin address is a string value which is a character set, and it is used for receiving money. Bitcoin addresses are generated from public keys. A Bitcoin address looks like:

19WVyUJ75rboXZymdgQdx3HENYWK7fsLpE.

Bitcoin address is generated from the signer’s public key using cryptographic hash functions. In many stage, the hash functions are also needed. For example, they are also in script addresses, and mining process etc. "Secure Hash Algorithm (SHA)" and "RACE Integrity Primitives Evaluation Message Digest (RIPEMD)", especially SHA-256 and RIPEMD-160, are used to generate the Bitcoin addresses.

With public key pk , first it is calculated using the SHA-256 and then the RIPEMD-160 of the first result:

$$A = \text{RIPEMD-160}(\text{SHA-256}(pk))$$

Bitcoin addresses are generally encoded the format Base58Check (“Base58 and Base58Check Encoding”), that contains 58 number of characters and a check-sum to guarantee ease of spelling. Figure 2.1 gives the conversion in a chart.

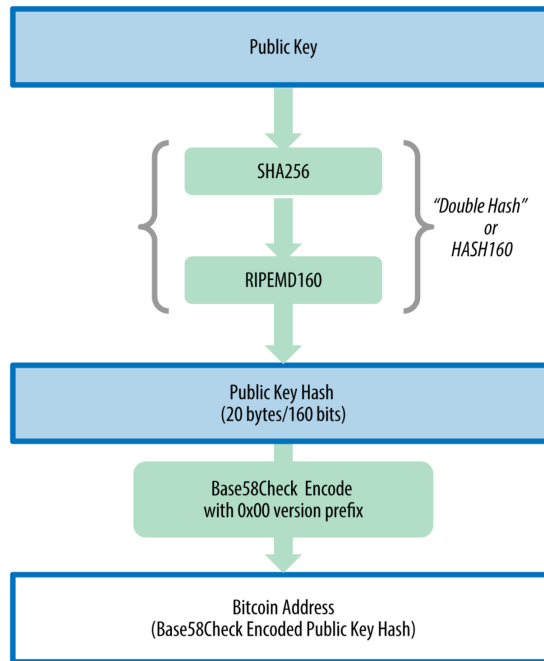


Figure 2.1: Converting public key to Bitcoin address

Example 2.1. *bitcoin's Base58 alphabet*

123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz

2.2 Block Structure

In the blockchain, a block contains data structure of collected transactions as a ledger, which is kept public. Specifically, a block consists of a header including metadata about the block, and a transactions list that occupies the majority of block size. Figure 2.2 shows the block structure.

Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1-9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

Figure 2.2: Structure of a block

Block Header

The block header has three kinds of metadata. First, a previous block hash field, which connects the block to the predecessor. The second is for difficulty, timestamp, and nonce fields which are required in the mining process. The third one is "the Merkle tree root", that is a structure and is to digest the transactions in a block. Figure 2.3 shows the content of a block header.

Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The proof-of-work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the proof-of-work algorithm

Figure 2.3: Structure of the header

2.3 Transaction Structure

A transaction is an information which records a transfer of Bitcoin amount from the source, named input, to the destination, named output. Any kind of account is not used in generating the input or output, only Bitcoin address is required personally. Bitcoins are locked with a Bitcoin address that only person, owner or who knows it, be able to unlock. Figure 2.4 shows the fields in a transaction.

Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1–9 bytes (VarInt)	Input Counter	How many inputs are included
Variable	Inputs	One or more transaction inputs
1–9 bytes (VarInt)	Output Counter	How many outputs are included
Variable	Outputs	One or more transaction outputs
4 bytes	Locktime	A Unix timestamp or block number

Figure 2.4: Structure of a transaction

Transaction Outputs

Bitcoin transactions has outputs and they are stored in Bitcoin ecosystem. Pretty much of all outputs create spendable Bitcoin called unspent transaction outputs (UTXO). The whole network recognizes them and the owner can spend in later transactions. When transferring Bitcoin to someone, the "unspent transaction output (UTXO)" is registered to receivers Bitcoin address and ready to a next transfer.

A transaction output consists of two parts listed below.

- An amount of transferred bitcoin.
- A locking script that locks this amount and concern us fundamentally in this thesis.

Size	Field	Description
8 bytes	Amount	Bitcoin value in satoshis (10^{-8} bitcoin)
1-9 bytes (VarInt)	Locking-Script Size	Locking-Script length in bytes, to follow
Variable	Locking-Script	A script defining the conditions needed to spend the output

Figure 2.5: Structure of a transaction output

Transaction Inputs

Transaction input is the pointer which links between the current transaction and previous UTXO. It points to UTXO by reference to the transaction hash and sequence number where the UTXO is recorded in the blockchain. In addition, the transaction input includes unlocking script that meets the locking script. The unlocking script is generally a proof of the signature ownership. Figure 2.3 shows the structure of a transaction input.

2.4 Transaction Scripts

When a new transaction is broadcast to the network to be included in the blockchain, each node, aware of the transaction, validates it by evaluating the challenge and response scripts. Most common three types of transaction scripts are below.

Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent; first one is 0
1-9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking script.
4 bytes	Sequence Number	Currently disabled Tx-replacement feature, set to 0xFFFFFFFF

Figure 2.6: The transaction input structure

Script Construction (Lock + Unlock)

The validation of Bitcoin transaction process depends on two types of scripts, locking and unlocking. A locking script is a hypothec on output, and it sets the conditions that must be met to spend the output when spending an amount of bitcoin.

An unlocking script is to meet the conditions placed on an output by a locking script and allows the output to be spent. Unlocking scripts are part of every transaction input, and mostly they include a digital signature generated with the user's private key.

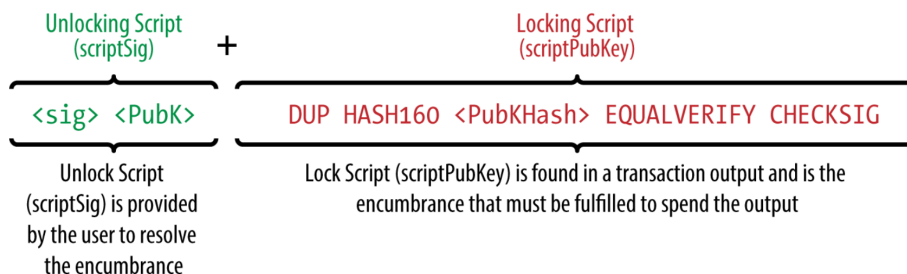


Figure 2.7: The unlocking and locking script combination

2.4.1 Pay To Public Key (P2PK)

Pay-to-public-key is a simpler form of a Bitcoin payment than other methods. A pay-to-public-key locking script is handled as:

<Public Key of A> OP_CHECKSIG

OP_CHECKSIG is an ECDSA signature verification procedure to check whether the signature of a user A is valid for a given public key in a transaction locking script.

The unlocking script is a simple signature:

<Signature by Private Key of A>

The combined script, to be validated, is:

<Signature by Private Key of A> <Public Key of A> OP_CHECKSIG

It is computationally hard to obtain the private key from the public key in an acceptable time period. So the public key is safely used as an address for receiving Bitcoin payments. In P2PK script template, public keys are called P2PK addresses. But nowadays, to ensure better security P2PK addresses are left in place to P2PKH addresses [1, 2].

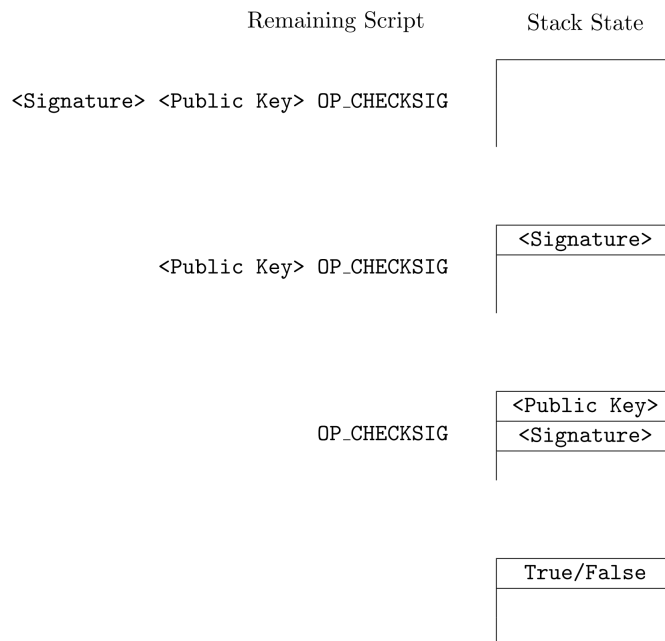


Figure 2.8: The P2PK script flow

2.4.2 Pay To Public Key Hash (P2PKH)

The most common type of transaction script in the Bitcoin network is P2PKH script template. Its locking script contains a public key hash instead of a plain public key as a Bitcoin address. Therefore, the public key must be presented to unlock amount of Bitcoin and also a digital signature created by the corresponding private key.

To illustrate P2PKH script template with an example, let Alice pay to Bob's Cafe. Alice made a payment of 0.015 Bitcoin to the Bitcoin address of the cafe. The locking script of transaction output would be like:

```
OP_DUP OP_HASH160 <Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG
```

Cafe Public Key Hash is the Bitcoin address of the cafe. The unlocking script corresponding to the locking script is:

```
<Cafe Signature> <Cafe Public Key>
```

The validation script is as given below

```
<Cafe Signature> <Cafe Public Key> OP_DUP
```

```
OP_HASH160 <Cafe Public Key Hash> OP_EQUAL OP_CHECKSIG
```

This combined script's execution results TRUE if the unlocking script matches the conditions set by the locking script. In particular, the result will be TRUE if the unlocking script has a valid signature generated by the cafe's private key that corresponds to the public key hash set as a hypothec. Figure 2.9 shows a step-by-step execution of the combined script to validate the transaction [1, 2].



Figure 2.9: The P2PKH script flow

2.4.3 Multisig (M-of-N Multi-Signature)

In multi-signature script template, to release the hypothec, N public keys are recorded in the locking script and at least M of N must match the corresponding signatures. This is also called as an M-of-N scheme, where N is the total number of keys and M is the threshold value of signatures required for validation.

To illustrate multi-signature script template with an example, a 2-of-3 multi-signature requires that three public keys are listed as signers group, to create a valid transaction to spend an amount of bitcoin, at least two of which must be used to create signatures. Standard multisignature scripts may have some limitation, e.g. at most 15 listed public keys, meaning you can use maximum 15-participants group of signers. The maximum number can be adjusted according to improvement in computational power in time.

The general form of a locking script setting an M-of-N multi-signature condition is:

```
M <Public Key 1> <Public Key 2> ...<Public Key N> N OP_CHECKMULTISIG
```

where N is the total number of public keys and M is the value that minimum number of signatures is required to spend the output. For example, in a 2-of-3 multi-signature, the locking script is:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG
```

The corresponding unlocking script that contains two signatures is:

```
OP_0 <Signature B> <Signature C>
```

or the other two-combination of signatures associated with declared three public keys. Hence, the validation script is:

```
OP_0 <Signature B> <Signature C> 2
```

```
<Public Key A> <Public Key B> <Public Key C> 3 OP_CHECKMULTISIG
```

If two signatures in the unlocking script match the two of three public keys in the locking script, combined validation script will result TRUE [1, 2].

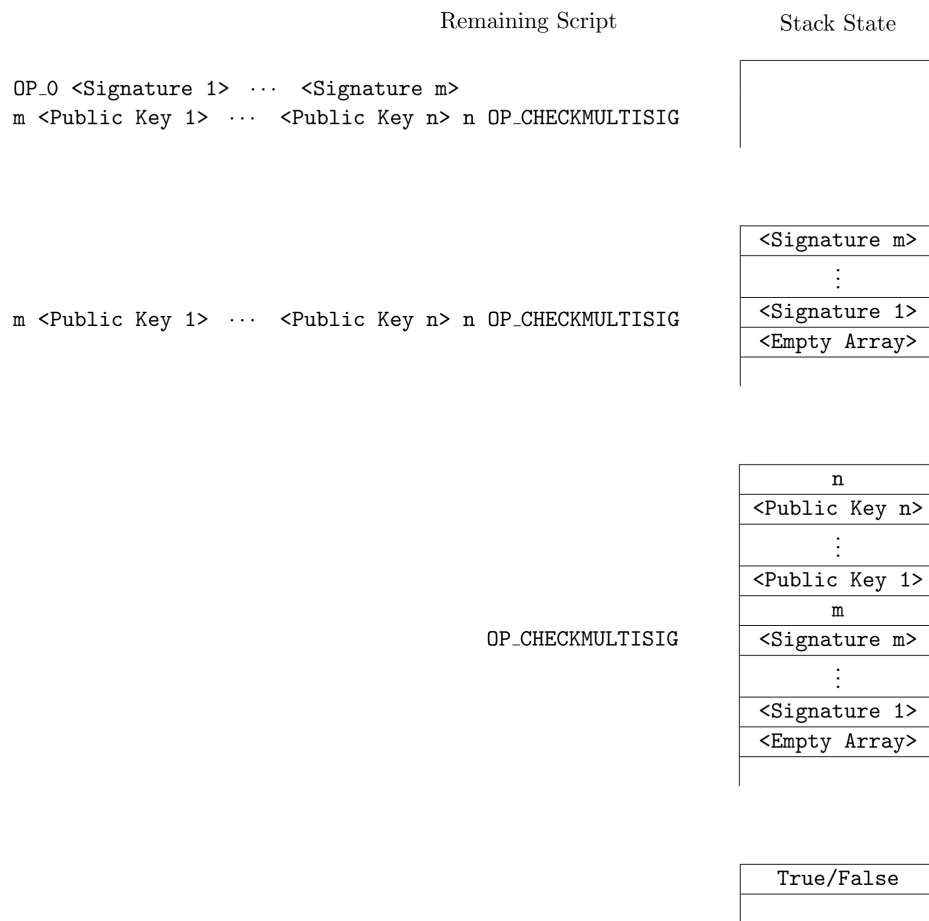


Figure 2.10: The m-of-n multisig script flow

3 ACCOUNTABLE SUBGROUP MULTI-SIGNATURE

Micali, Ohta, and Reyzin [5] defined an accountable-subgroup multisignature (ASM) scheme where any subset S of a group of PK, where PK is the set of the public keys of the signers, can create a valid multisignature that can be verified by the public keys of signers in the subset. To determine whether the subset S is authorized to sign on behalf of PK in a more strictly way, an established access rule can be added into ASM scheme. For example, setting $|S| \geq t$ as a condition, the ASM scheme gets the threshold signature property that authentication can be done by a minimum number of signers.

Initially, a description of the set S of signers in the group PK is necessary for verification of an ASM scheme. Verification procedure is based on a compact aggregate public key and signature [3, 6]. In ASM scheme, the aggregate public key can be generated from the public keys of signers publicly, on the other hand, a membership key which is specific to the group required to sign messages on behalf of the group PK. The group-specific membership key is generated via a onetime group setup.

3.1 Bilinear Groups and Pairing Based Cryptography

Let \mathcal{G} be a bilinear group generator that takes as an input a security parameter 1^κ and outputs the descriptions of multiplicative groups $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$ where $\mathbb{G}_1, \mathbb{G}_2$ are groups of prime order q , and \mathbb{G}_t be another cyclic group of order q written multiplicatively, a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_t$, and g_1 and g_2 are generators of the groups \mathbb{G}_1 and \mathbb{G}_2 , respectively. For later use, it is denoted that $\Lambda^* = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$. If the bilinear map e is a pairing, it has the following properties [10, 14]:

Bilinearity

$$\forall a, b \in \mathbb{F}_q^*, \forall P \in \mathbb{G}_1, Q \in \mathbb{G}_2 : e(P^a, Q^b) = e(P, Q)^{ab}$$

Non-degeneracy

$$e \neq 1$$

Computability

There exists an efficient algorithm to compute e .

As a usage example of bilinear pairings and to construct ASM [3], lets look BLS [17] signature schema. In addition to bilinear paring requirements, $H_0 : \mathcal{M} \rightarrow \mathbb{G}_1$ is a hash function. BLS works as:

- **Key generation:** Select a $sk \xleftarrow{\$} \mathbb{Z}_q$ randomly and return (pk, sk) where $pk \leftarrow g_2^{sk} \in \mathbb{G}_2$.
- **Sign(sk, m):** Return $\sigma \leftarrow H_0(m)^{sk} \in \mathbb{G}_1$.
- **Verify(pk, m, σ):** If $e(\sigma, g_2) \stackrel{?}{=} e(H_0(m), pk)$ return "true", else "false"

In BLS schema, there also can be a simple signature aggregation. Given (pk_i, m_i, σ_i) for $i = 1, \dots, n$, signatures $\sigma_1, \dots, \sigma_n \in \mathbb{G}_1$ can be converted by aggregating them like:

$$\sigma \leftarrow \sigma_1, \dots, \sigma_n \in \mathbb{G}_1$$

To verify the aggregated signature $\sigma \in \mathbb{G}_1$:

$$e(\sigma, g_2) \stackrel{?}{=} e(H_0(m_1), pk_1) \dots e(H_0(m_n), pk_n).$$

All (pk_i, m_i) for $i = 1, \dots, n$ are needed to verify. As a trick, if all messages are chosen the same ($m_1 = \dots = m_n$), verification is downgraded on only two pairings:

$$e(\sigma, g_2) \stackrel{?}{=} e(H_0(m_1), pk_1 \dots pk_n).$$

Hence, the aggregated public key concept is born $apk := pk_1 \dots pk_n \in \mathbb{G}_2$.

The rogue public-key attack.

The signature aggregation in BLS is not so secure, it needs to be improved. See Section 3.4. To illustrate its weakness, lets look at the attack scenario: "an attacker registers a rogue public key $pk_2 := g_2^\alpha \cdot (pk_1)^{-1} \in \mathbb{G}_2$, where $pk_1 \in \mathbb{G}_2$ is a public key of some unsuspecting user Bob, and $\alpha \xleftarrow{\$} \mathbb{Z}_q$ is chosen by the attacker. The attacker can then claim that both he and Bob signed some message $m \in \mathcal{M}$ by presenting the aggregate signature $\sigma := H_0(m)^\alpha$.

This signature verifies as an aggregate of two signatures, one from pk_1 and one from pk_2 , because

$$e(\sigma, g_2) = e(H_0(m)^\alpha, g_2) = e(H_0(m), g_2^\alpha) = e(H_0(m), pk_1 \cdot pk_2).$$

Hence, this σ satisfies. In effect, the attacker committed Bob to the message m , without Bob ever signing m ." [3]

3.2 Construction of ASM Schema

As first defined in [4], an ASM schema which is based on Schnorr signature [18] consists of the algorithms: parameter generation (Pg), key generation (Kg), group setup (GSetup), Sign, key aggregation (KAg), and verification (Vf). The notation $par \xleftarrow{\$} Pg$ is for common parameters generation, $(pk, sk) \xleftarrow{\$} Kg(par)$ is for key generation, $mk \leftarrow GSetup(sk, PK)$ is for generating membership key where $PK = \{pk_1, \dots, pk_n\}$ is a group of signers. Let each signer in PK be assigned a computable index $i \in \{1, \dots, |PK|\}$, for example the index of pk in a sorted list of PK . A subgroup $S \subseteq \{1, \dots, |PK|\}$ signs a message m with the interactive algorithm $\sigma \leftarrow Sign(par, PK, S, sk, mk, m)$. Getting the public keys of PK , key aggregation algorithm generates the aggregated public key apk [7]. The algorithm $Vf(par, apk, S, m, \sigma)$ verifies the signature [3].

With this construction strengthened with BLS schema [17], ASM scheme needs all signers, during group setup, join to multisignatures on the aggregate public key and the index of every signer, such that the i -th signer in PK has a "membership key" which is a multi-signature on (apk, i) . In other words, an accountable-subgroup multi-signature consists of the aggregation of the individual signers' signatures and their membership keys and the aggregate public key of the subgroup S . To verify a signed message by a subgroup S , it can be checked that the signature is a valid aggregate signature where the aggregate public key of the subgroup signed the message and the membership keys corresponding to S [3].

The scheme uses hash functions $H_0 : \{0, 1\}^* \rightarrow \mathbb{G}_1$, $H_1 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$, and $H_2 : \{0, 1\}^* \rightarrow \mathbb{G}_1$ [8].

Parameters Generation. The bilinear group is set up by $Pg(\kappa)$ function and also it returns

parameters

$$par \leftarrow (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2) \leftarrow \mathcal{G}(\kappa).$$

Key Generation. Algorithm $\text{Kg}(par)$ function gets $sk \xleftarrow{\$} \mathbb{Z}_q$ randomly, generates $pk \leftarrow g_2^{sk}$, and returns (pk, sk) .

Key Aggregation. $\text{KAg}(\{pk_1, \dots, pk_n\})$ gets public key values of all signers and returns

$$apk \leftarrow \prod_{i=1}^n pk_i^{H_1(pk_i, \{pk_1, \dots, pk_n\})}.$$

Group Setup. At first, $\text{GSetup}(sk_i, \mathcal{PK} = \{pk_1, \dots, pk_n\})$ checks whether $pk_i \in \mathcal{PK}$ and i is the index of pk_i in \mathcal{PK} . Signer i computes the aggregate public key

$$apk \leftarrow \text{KAg}(\mathcal{PK}) \text{ and } a_i \leftarrow H_1(pk_i, \mathcal{PK}).$$

Then, he sends

$$\mu_{j,i} = H_2(apk, j)^{a_i \cdot sk_i} \text{ to signer } j \ (j \neq i) \text{ publicly.}$$

After receiving $\mu_{i,j}$ from all other signers $j \neq i$, they compute

$$\mu_{i,i} \leftarrow H_2(apk, i)^{a_i \cdot sk_i}$$

and return the membership key

$$mk_i \leftarrow \prod_{j=1}^n \mu_{i,j}.$$

And hence, if all signers behave honestly, this equation must be done

$$e(mk_i, g_2) \stackrel{?}{=} e(H_2(apk, i), apk).$$

In other words, mk_i is a valid multi-signature on the message $H_2(apk, i)$ by all n parties.

Signing. $\text{Sign}(par, \mathcal{PK}, S, sk_i, mk_i, m)$ generates $apk \leftarrow \text{KAg}(\mathcal{PK})$ and

$$s_i \leftarrow H_0(apk, m)^{sk_i} \cdot mk_i,$$

and sends (pk_i, s_i) to a member of S or external one to join them, who generates

$$pk \leftarrow \prod_{j \in S} pk_j, \quad s \leftarrow \prod_{j \in S} s_j,$$

and returns the value $\sigma := (pk, s)$ as the multisignature. Note that we use the set S at signing step, because it isn't fixed. Thus, the set S has not to be same for every signing.

Verification. $\text{Vf}(par, apk, S, m, \sigma)$ evaluates σ as (pk, s) and returns TRUE if and only if

$$e(\text{H}_0(apk, m), pk) \cdot e\left(\prod_{j \in S} \text{H}_2(apk, j), apk\right) \stackrel{?}{=} e(s, g_2)$$

and also returns the information whether S is an authorized set of signers.

Now, we prove the accuracy of the system. Under the condition that all members of S are honest in the group setup and signing stages, and for

$$pk = g_2^{\sum_{i \in S} sk_i}, \quad apk = g_2^{\sum_{i=1, \dots, n} a_i \cdot pk_i},$$

$$s = \text{H}_0(apk, m)^{\sum_{i \in S} sk_i} \cdot \prod_{i \in S} \text{H}_2(apk, i)^{\sum_{j \in 1, \dots, n} a_j \cdot sk_j},$$

equation array below holds:

$$e(s, g_2) = e(\text{H}_0(apk, m)^{\sum_{i \in S} sk_i} \cdot \prod_{i \in S} \text{H}_2(apk, i)^{\sum_{j \in 1, \dots, n} a_j \cdot sk_j}, g_2)$$

$$= e(\text{H}_0(apk, m), pk) \cdot e\left(\prod_{i \in S} \text{H}_2(apk, i), g_2^{\sum_{j \in 1, \dots, n} a_j \cdot sk_j}\right)$$

$$= e(\text{H}_0(apk, m), pk) \cdot e\left(\prod_{i \in S} \text{H}_2(apk, i), apk\right)$$

Example 3.1. To illustrate this schema with an example, the elliptic curve E of the form

$$E : y^2 = x^3 + 18$$

over the finite field \mathbb{F}_p , where

$$p = 2497857711095780713403056606399151275099020724723$$

is used. The cyclic subgroups G_1 and G_2 generated from g_1 and g_2 respectively having prime order

$$q = 2497857711095780713403055025937917618634473494829$$

in the elliptic curve group are generated. As the base for the ASM schema, let's choose Weil pairing as our bilinear map, and H_0 , H_1 and H_2 as simple toy hash functions that are constructed from only the first component of selected points. All other parameters are in Appendix Listing 1. Because of selected groups being additive, we transform the formulation for compliance.

We choose 5 potential signers and 3-user subgroup of them, and their the secure-private key pairs as (sk_i, pk_i) over G_2 manually. The $\mu_{i,j}$ values of all signers 2-combination are generated from their secure-private key pairs, indexes and the aggregated public key apk . Thanks to the membership keys mk_i , which will be used in signing stage, are generated from μ values gathered from other signers, when our subgroup signs a message, the others are also responsible for the signature. Hence, the accountability is provided.

To sign a message m , with calculating s_i values and adding all these values with also adding public keys of our subgroup among themselves, the signature is generated. Finally, to verify the signature, we check this equation:

$$Weilpairing(H_0(apk, m), pk).Weilpairing(\prod_{j \in S} H_2(apk, j), apk) \stackrel{?}{=} Weilpairing(s, g_2).$$

Entire calculation in Magma code is in Appendix Listing 1.

3.3 Proof-of-Possession

Rogue-key attack explained in Section 3.1 can be prevented by letting signers add a so-called proof of possession (PoP) to their public keys, which is simply a signature on their own public key [9].

The proof of possessions gives us some advantages on the ASM scheme. The aggregate public keys are simply calculated values of multiplication or hash of the signers keys. Also, its security proofs avoid forking, efficient for high security needs, and shorter key and signature sizes for physical constraint.

3.4 Accountable-Subgroup Scheme with PoPs

To integrate PoPs, key structure is like $y \leftarrow g_2^{sk}$, $\pi \leftarrow H_3(y)^x$, $H_3 : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ as a new hash, and $pk \leftarrow (y, \pi)$, also key aggregation differs in that the aggregate of a set of keys $\mathcal{PK} = \{(y_1, \pi_1), \dots, (y_n, \pi_n)\}$ in addition to the product $Y \leftarrow \prod_{i=1}^n y_i$, it has the hash of public keys $h \leftarrow H_3(\mathcal{PK})$. The aggregate public key is a pair $apk \leftarrow (Y, h)$. The reason to use hash is, when evaluating $H_2(apk, i)$, to consider whether i is the index of the target signer in the set \mathcal{PK} for which apk is the aggregate public key. Before aggregating, for completeness, first it may be needed to test that:

$$e(H_3(y_i), y_i) \stackrel{?}{=} e(\pi_i, g_2) \text{ for } i = 1, \dots, n.$$

$GSetup(sk_i, \mathcal{PK} = pk_1, \dots, pk_n)$ calculates

$$apk = (Y, h) \leftarrow KAg(par, \mathcal{PK})$$

and sends

$$\mu_{j,i} = H_2(apk, j)^{sk_i} \text{ to signer } j.$$

Receiving $\mu_{j,i}$ from all other signers $j \neq i$, signer i calculates

$$\mu_{j,i} \leftarrow H_2(apk, j)^{sk_i}$$

and returns the membership key

$$mk_i \leftarrow \prod_{j=1}^n \mu_{i,j}.$$

If any signer is not malicious, it must be equal

$$e(mk_i, g_2) = e(H_2(apk, i), Y).$$

Signing and verification stages are same as pure ASM scheme, except the product of public keys $y \leftarrow \prod_{j \in S} y_j$ instead of $pk \leftarrow \prod_{j \in S} pk_j$ [3].

Example 3.2. Now we add the proof of possession mechanism to Example 3.1. Most of the parameters and the scenario being the same as the previous example, H_3 is additionally a new toy hash function generated from the second component of points. To prevent from an attackers presenting a fake signature, we choose a x value randomly from the field \mathbb{F}_p and we generate an additional (y, π) pair over the value x instead of pk which corresponds to sk .

Singing stage is like the previous example. Only the difference, (y_i, π_i) values are used instead of public keys of signers pk_i , and y is for pk with the same calculation. To verify the signature we check now this equation:

$$Weilpairing(H_0(apk, m), Y) \cdot Weilpairing\left(\prod_{j \in S} H_2(apk, j), apk\right) \stackrel{?}{=} Weilpairing(s, g_2).$$

The Magma codes for this example is presented in Appendix Listing 2.

3.5 Application ASM to Bitcoin

To implement ASM schema in Bitcoin transaction scripts, aggregated public key can be used like a Bitcoin address. We can illustrate the possible application with script language. Instead of the locking script in Multisig (M-of-N schema)

```
M <Public Key 1> <Public Key 2> ...<Public Key N> N OP_CHECKMULTISIG
```

and the unlocking script

```
OP_0 <Signature  $n_1$ > ...<Signature  $n_M$ >,
```

the possible locking script in ASM schema should be like

```
<Agg. Public Key> OP_CHECKASM
```

and the unlocking script in ASM

$$OP_0 \langle S \rangle \langle \text{Common Signature of } S \rangle$$

where S is the subgroup of signers and $\langle S \rangle$ is a short string of signers' indexes.

As can be seen from the comparison, the size and computation cost can be saved owing to ASM schema aggregation property.

4 DELEGATABLE CREDENTIALS WITH HIDEABLE ATTRIBUTES

Requirements on data security and privacy are getting more important. In today's technology mathematical tools are widely used for providing this. Privacy-preserving attribute-based credentials (PABCs) [15] allow us to authenticate to a transaction submission service using only necessary and sufficient information.

PABCs is a set of attributes certified to a person. When presenting her credentials, user generates a zero-knowledge proof of possession of a credential named token. When generating a token, user can choose which attribute to be disclosed or not. To verify the token only the issuer's public key is required. Even with having strong privacy features, in some scenarios, PABCs can cause disclosing information about the issuer such as age, gender, location, organization, business unit etc.

"For instance, consider governmental issued certificates such as drivers licenses, which are typically issued by a local authority whose issuing keys are then certified by a regional authority, etc. So there is a hierarchy of at least two levels if not more. Thus, when a user presents her drivers license to prove her age, the local issuer's public key will reveal her place of residence, which, together with other attributes such as the user's age, might help to identify the user. As another example consider a (permissioned) blockchain. Such a system is run by multiple organizations that issue certificates (possibly containing attributes) to parties that are allowed to submit transactions. By the nature of blockchain, transactions are public or at least viewable by many blockchain members. Recorded transactions are often very sensitive, in particular when they pertain to financial or medical data and thus require protection, including the identity of the transaction originator. Again, issuing credential in a permissioned blockchain is a hierarchical process, typically consisting of two levels, a (possibly distributed) root authority, the first level consisting of CAs by the different organizations running the blockchain, and the second level being users who are allowed to submit transactions." [10]

Practical delegatable anonymous credentials [16] can be used for solving this issue. This method gives us to an opportunity to delegate a user credential to another user, also delegated user can delegate his likewise. In this way, unwanted attributes can be hidden since only initial (root) user's public key is needed for verification.

4.1 Practical Delegatable Anonymous Credentials System

The root delegator (issuer) generates a signing and a verification key pair. User A, who gets delegation from issuer, also generates a key pair, public and private. User A sends the public key to the issuer. The issuer signs the public key and attributes of user A and sends the signature back.

User A can also delegate her credential to a user B via signing the public key freshly generated by B and attributes using his private key (A's sk). A sends this signature with his credential acquired from the issuer and attributes to B. At the end, B has two signatures with the corresponding attribute sets, credential public key of A and her own public and private key.

By the same way above, user B can delegate his credential to another user or sign a message by generating the presentation token. The presentation token is a non-interactive zero-knowledge (NIZK) proof of possession of the signatures and the corresponding public keys from the delegation chain which does not disclose their values [11, 10]. Preferably the signing attributes can be disclosed using NIZK. To verificate the token, only the public key of issuer is required, thus, A and B with their unwanted attributes can be hidden.

4.2 Signature Schemes

We define Sig as a digital signature scheme that is a set of PPT (probabilistic polynomial-time) algorithms $\text{Sig} = (\text{Setup}, \text{Gen}, \text{Sign}, \text{Verify})$:

Sig.Setup $(1^{\mathcal{K}}) \xrightarrow{\$} sp$: The setup algorithm gets an input value \mathcal{K} as a security parameter and generates a public system parameters sp which is specific to message space \mathcal{M} .

Sig.Gen $(sp) \xrightarrow{\$} (sk, vk)$: The key generation algorithm gets sp and generates a key pair public and secret key (pk, sk) .

Sig.Sign $(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm gets sk and a message $m \in \mathcal{M}$ and generates a signature σ .

Sig.Verify(vk, m, σ) $\xrightarrow{\S}$ 1/0 : The verification algorithm gets vk , the message m and the signature σ , validate it if it is true returns 1 or if it is false returns 0 according to the inputs.

Construction for delegatable credentials is built from the structure-preserving [12] signature schemes. A signature scheme Sig over a bilinear group Λ generated by $\mathcal{G}(1^k)$, that outputs system parameters $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$, is said to be structure preserving if:

(1) the verification key vk comes from the group parameters and group elements in \mathbb{G}_1 and \mathbb{G}_2 ;

(2) the messages and the signatures comes from group elements in \mathbb{G}_1 and \mathbb{G}_2 ,

(3) the verification algorithm evaluates membership \mathbb{G}_1 and \mathbb{G}_2 and pairing product equations as:

$$\prod_i \prod_j e(g_i, h_j)^{a_{ij}} = 1_{\mathbb{G}_t} ,$$

where $a_{11}, a_{12}, \dots \in \mathbb{Z}_q$ are constants, $g_1, h_1, \dots \in \{\mathbb{G}_1, \mathbb{G}_2\}^*$ are group elements appearing in the group parameters, verification key, messages, and signatures.

4.2.1 Groth Structure-Preserving Signature Schema

We shortly give the structure-preserving signature scheme of Groth [13] in this section. The original scheme supports signing blocks of messages in a form of “matrix”, whereas we provide a simplified versions for “vectors” of messages. Let a message be a vector of group elements of length n : $\vec{m} = (m_1, \dots, m_n)$. The notation Groth1 is for signing messages in \mathbb{G}_1 with a public key in \mathbb{G}_2 , on the other hand Groth2 is for signing in \mathbb{G}_2 with a public key in \mathbb{G}_1 . The description of the Groth2 scheme is below, Groth1 is defined similarly.

Groth2.Setup : Let $\Lambda^* = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e)$ and $y_i \xleftarrow{\S} \mathbb{G}_2$ for $i = 1, \dots, n$. Return parameters $sp = (\Lambda^*, \{y_i\}_{i=1, \dots, n})$.

Groth2.Gen(sp) : Choose random $v \xleftarrow{\S} \mathbb{Z}_q$ and set $V \xleftarrow{\S} g_1^v$. Return key pair $pk = V$ as public and $sk = v$ as secret.

Groth2.Sign($sk; \vec{m}$) : To sign message $\vec{m} \in \mathbb{G}_2^n$ choose random $r \xleftarrow{\$} \mathbb{Z}_q^*$ and set

$$R \leftarrow g_1^r \quad S \leftarrow (y_1 \cdot g_2^v)^{\frac{1}{r}} \quad T_i \leftarrow (y_i^v \cdot m_i)^{\frac{1}{r}} .$$

Return signature $\sigma = (R, S, T_1, \dots, T_n)$.

Groth2.Verify(vk, σ, \vec{m}) : Get message $\vec{m} \in G_2^n$ and signature $\sigma = (R, S, T_1, \dots, T_n) \in G_1 \times G_2^{n+1}$, return 1 iff

$$e(R, S) = e(g_1, y_1)e(V, g_2) \wedge \bigwedge_{i=1}^n e(R, T_i) = e(V, y_i)e(g_1, m_i)$$

for any $j \in \{1, \dots, n\}$

$$\begin{aligned} e(R, S) &= e(g_1^r, (y_1 \cdot g_2^v)^{\frac{1}{r}}) & e(R, T_j) &= e(V, y_j)e(g_1, m_j) \\ &= e(g_1, (y_1 \cdot g_2^v)) & &= e(g_1^r, (y_j^v \cdot m_j)^{\frac{1}{r}}) \\ &= e(g_1, y_1)e(g_1, g_2^v) \wedge & &= e(g_1, (y_j^v \cdot m_j)) \\ &= e(g_1, y_1)e(g_1^v, g_2) & &= e(g_1, y_j^v)e(g_1, m_j) \\ &= e(g_1, y_1)e(V, g_2) & &= e(g_1^v, y_j)e(g_1, m_j) \\ & & &= e(V, y_j)e(g_1, m_j) \end{aligned}$$

Groth2.Rand(σ) : To randomize signature $\sigma = (R, S, T_1, \dots, T_n)$, pick $r' \xleftarrow{\$} \mathbb{Z}_q$ and set

$$R' \leftarrow R^{r'} \quad S' \leftarrow S^{\frac{1}{r'}} \quad T' \leftarrow T^{\frac{1}{r'}} .$$

Return randomized signature $\sigma' = (R', S', T'_1, \dots, T'_n)$.

4.2.2 Sibling Signatures

Sibling signature scheme provides one key pair to use two different signing algorithms, each with a dedicated verification algorithm [10]. In the construction, this will allow a user to hold a single key pair to be able to use for both presentation and delegation of a credential.

A sibling signature scheme consists of six algorithms, Setup, Gen, Sign1, Sign2, Verify1, Verify2.

Sib.Setup $(1^\kappa) \xrightarrow{\$} sp$: The setup algorithm gets a security parameter and returns public system parameters that also specify two message spaces \mathcal{M}_1 and \mathcal{M}_2 .

Sib.Gen $(sp) \xrightarrow{\$} (sk, pk)$: The key generation algorithm gets the system parameters and returns a key pair: secret and public (sk, pk) .

Sib.Sign1 $(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm gets sk and a message $m \in \mathcal{M}_1$ and generates a signature σ .

Sib.Sign2 $(sk, m) \xrightarrow{\$} \sigma$: The signing algorithm gets sk and a message $m \in \mathcal{M}_2$ and generates a signature σ .

Sib.Verify1 $(vk, m, \sigma) \rightarrow 1, 0$: The verification algorithm gets pk , the message m and the signature σ , validate it if it is true returns 1 or if it is false returns 0 according to the inputs.

Sib.Verify2 $(vk, m, \sigma) \rightarrow 1, 0$: The verification algorithm gets pk , the message m and the signature σ , validate it if it is true returns 1 or if it is false returns 0 according to the inputs.

4.2.3 Constructing Sibling Signatures.

Setting the public key pk as (pk_1, pk_2) and the signing key as $sk = (sk_1, sk_2)$, a new sibling signature can be easily constructed from two standard signature schemes and using one signature scheme as Sign1 and Verify1 and the other as Sign2 and Verify2. Also, this construction allows that securely share key material between the two algorithms.

Thus, it can be combined by Groth1 signatures and Schnorr-signatures together to form a sibling signature scheme as notation SibGS1. SibGS1 uses only a single key pair and uses the Setup and Gen algorithms of Groth1. Algorithm Sign1 comes from Groth1.Sign, and Sign2 creates a Schnorr signature. In a same way, we can denote SibGS2 as the defined Groth-Schnorr sibling signature where Groth2 instead of Groth1.

4.3 Construction for Delegatable Credentials

An issuer can delegate his credential to a user and the user can present the credential which is acquired and also this can be verified. If we suppose the issuer is Level-0, the first stage delegator is Level-1. And further, the user with Level-($i-1$) is able to delegate his credential to Level- i . The outcome of presenting a delegated credential like signing a message instead of issuer is attribute token that can be verified by the private key of issuer and hiding any attributes of delegators including identities. We now explain on a high level how a user obtains a Level-1 credential and then that credential is delegated [10]. It is then easy to see how a Level- L credential is delegated on Figure 4.1.

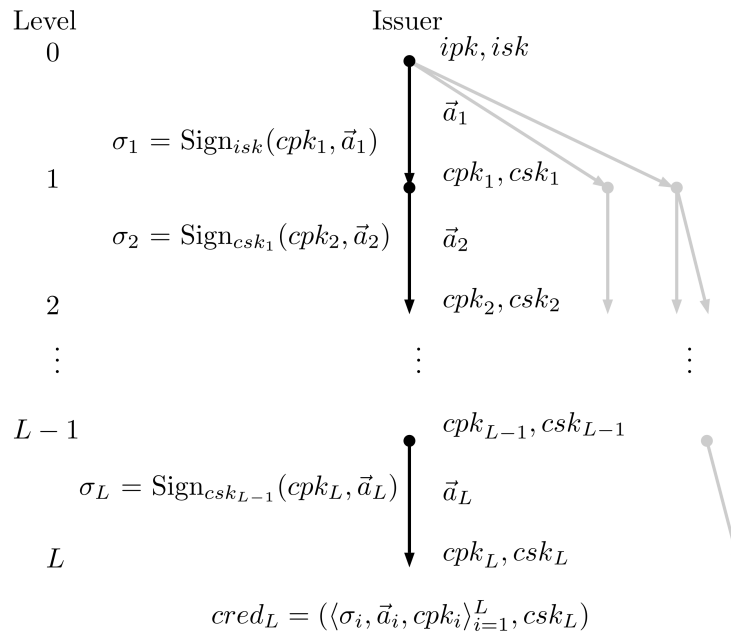


Figure 4.1: The delegation chart

First, the issuer generate a public and private (secret) key pair ipk, isk . The user to whom the issuer wants to delegate Level-1 credential also generate a fresh key pair cpk_1, csk_1 and sends the public one to the issuer. The issuer signs the cpk_1 with the attributes a_1 and sends the signature σ_1 back to user u_1 . In this way, the Level-1 credential $cred_1$ which contains σ_1 , \vec{a}_1 and key pair csk_1, cpk_1 is established.

The user u_1 owned $cred_1$ can also delegate his credential another user u_2 with Level-2. Like first delegation process, u_2 creates fresh key pair csk_2, cpk_2 and send cpk_2 to u_1 . After signing cpk_2 and \vec{a}_2 , u_1 send back to u_2 . This time, Level-2 credential $cred_2$ consists of

signatures σ_1, σ_2 , attributes \vec{a}_1, \vec{a}_2 , keys cpk_1, cpk_2, csk_2 . u_2 can prove that he has the Level-2 credential with his secret key csk_2 .

Level-2 credential can be delegated the same way above and it can continue so on. Because of not sharing the private keys no one present or delegate another level of credential.

With present stage any delegator can prove his possession of all credential and hide the selected attributes from the signature. The proof and also the signature of a message is called as attribute token, which can be verified by the issuer public key [10].

Construction

For delegation process, we use Groth-Schnorr sibling signatures SibGS introduced in previous chapter. Groth signature scheme uses bilinear group $\Lambda = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, e, g_1, g_2)$. Recall that Groth1 signs messages in G_1 with a public key in G_2 , while Groth2 signs messages in G_2 with a public key in G_1 . Therefore, we set Level- $2n$ to SibGS1 and Level- $2n+1$ to SibGS2. This means that $\mathbb{A}_{2n} = \mathbb{G}_1$ and $\mathbb{A}_{2n+1} = \mathbb{G}_2$.

Also, SibGS1 requires $y_{1,1}, \dots, y_{1,n+1} \in \mathbb{G}_1$ as parameters, where n is the maximum number of attributes signed at an odd level ($n = \max_{i=1,3,\dots}(n_i)$), and SibGS2 requires $y_{2,1}, \dots, y_{2,n+1} \in G_2$, for n the maximum number of attributes signed at an even level ($n = \max_{i=2,4,\dots}(n_i)$).

Let issuer be \mathcal{I} and users be u_i . They can delegate, present and verify the credentials via four stage; Setup, Delegate, Present, Verify [10]. The method is described in a protocol way as below;

Setup : After receiving (SETUP, $sid, \langle n_i \rangle_i$):

- Check $sid = \mathcal{I}, sid'$ for some sid'
- Run $(ipk, isk) \leftarrow \text{SibGS2.Gen}(1^\kappa)$ and let $cpk_0 \leftarrow ipk$
- Output (SETUPDONE, sid)

Delegate : Any user u_i with Level- $(L-1)$ can delegate to another user u_j with Level- L . u_j

can select which attributes are disclosed or hidden.

- Input(Delegate, $sid, ssid, a_1, \dots, a_L, u_j$) to user u_i with $\vec{a}_L \in \mathbb{A}_L^{n_L}$.
- If $L = 1$, u_i only check $sid = (\mathcal{I}, sid')$. Else u_i search that $cred = (\langle \sigma_i, \vec{a}_i, cpk_i \rangle_{i=1}^{L-1}, csk_{L-1})$ in $Lcred$.
- Send $(sid, ssid, \vec{a}_1, \dots, \vec{a}_L)$ to u_j .
- After receiving $(sid, ssid, \vec{a}_1, \dots, \vec{a}_L)$ by u_j from u_i , u_j generates fresh key pair $(cpk_L, csk_L) \leftarrow \text{SibGS}(L \bmod 2). \text{Gen}(1^\kappa)$.
- Send cpk_L to u_i .
- After receiving cpk_L , u_i computes $\sigma_L \leftarrow \text{SibGS}(L \bmod 2). \text{Sig1}(csk_{L-1}; cpk_L, \vec{a}_L)$ and sends back $\langle \sigma_i, cpk_i \rangle_{i=1}^L$ to u_j .
- After receiving $\langle \sigma_i, cpk_i \rangle_{i=1}^L$, u_j verifies $\text{SibGS}(i-1 \bmod 2). \text{Verify1}(csk_{i-1}, \sigma_i, cpk_i, \vec{a}_i)$ for $i = 1, \dots, L$ and stores $cred \leftarrow (\langle \sigma_i, \vec{a}_i, cpk_i \rangle_{i=1}^L, csk_L)$ in $Lcred$
- Output (DELEGATE, $sid, ssid, \vec{a}_1, \dots, \vec{a}_L, u_i$)

Present : A user can present his credential via signing a message and proving his possession. All attributes are described by $\vec{a}_1, \dots, \vec{a}_L, u_i$ and $\vec{a}_i = a_{i,1}, \dots, a_{i,n} \in \mathcal{A}$. Let D be the set of disclosed attributes, hence $\vec{a}_i \in A/D$ are hidden attributes.

- After receiving (Present, $sid, m, \vec{a}_1, \dots, \vec{a}_L$) with $\vec{a}_i \in \mathcal{A}_i^{n_i}$ for $i = 1, \dots, L$
- Look up $cred = (\langle \sigma_i, \vec{a}'_i, cpk_i \rangle_{i=1}^L, csk_L)$ in \mathcal{L}_{cred} such that $a_i \preceq a'_i$ for $i = 1, \dots, L$. If there is no such a credential, then abort.
- Create the attribute token by proving possession of the credential:

$at \leftarrow \text{NIZK/Present} \{(\sigma_1, \dots, \sigma_L, cpk_1, \dots, cpk_L, \langle a'_{i,j} \rangle_{i \notin D}, tag) :$

$$\bigwedge_{i=1,3}^L \dots 1 = \text{SibGS1.Verify1}(cpk_{i-1}, \sigma_i, cpk_i, a'_{i,1}, \dots, a'_{i,n_i})$$

$$\bigwedge_{i=2,4}^L \dots 1 = \text{SibGS2.Verify1}(cpk_{i-1}, \sigma_i, cpk_i, a'_{i,1}, \dots, a'_{i,n_i})$$

$\wedge 1 = \text{SibGSb.Verify2}(cpk_L, tag, m)$

Detailed computation is given in Appendix Listing 3.

- Output (TOKEN, sid, at).

Verify : Receiving (Verify, $sid, at, m, \vec{a}_1, \dots, \vec{a}_L$)

- Verify at with respect to m and $\vec{a}_1, \dots, \vec{a}_L$. If it is valid set $f \leftarrow 1$ else $f \leftarrow 0$.
- Output (Verified, sid, f).

Detailed computation is in Appendix Listing 4.

4.4 Application of Delegatable Credentials to Bitcoin

Application delegatable credentials to Bitcoin ecosystem might be for two aims. First, as a facility, someone can delegate another to spend a Bitcoin amount individually. With the property of keeping attribute values, the issuer can set some conditions to limit or control the spendings, like the condition of maximum payments by the position of the delegate which also is able to be stored in his credentials.

Second, as an additional security measure, delegatable credentials can be used in a certificate authority structure suggested in [10]. It may be needed especially for a private block chain. Let there be a membership service in this private block chain. It provides credentials to members. This structure is used for permitting transactions, authentication, controlling access, cancelling credentials, and auditing transactions. But, by the nature of delegatable credentials, there is a security and privacy vulnerabilities, because transactions can be trackable. Fortunately, hideable property of [10] can solve this issue.

For the first usage suggestion, implementation delegatable credentials schema in Bitcoin can also be illustrated with script language. The possible locking script in delegatable credentials schema should be like

`<Attribute Conditions> <Issuer Public Key> OP_CHECKDELEGATABLE`

and the unlocking script

OP_0 <Attribute Token>

where <Attribute Conditions> is a set of rules that the issuer has set to limit the later spendings and <Attribute Token> is a kind of signature which is signed by a delegate.

5 CONCLUSION

In this thesis, the transaction authentication mechanisms in a blockchain, especially in Bitcoin ecosystem, and two alternatives are presented.

Firstly, the topics of existing structure of blockchain concept, Bitcoin, its transaction scripts and most used authentication schemes are introduced. In addition to the necessity of protecting the Bitcoin assets, which is so valuable and equivalent to money, and must be kept safely, it is determined that the transactions need to be carried out effectively.

With discovering ineffective use, Accountable Subgroup Multi-Signature (ASM) is proposed instead of Multisig transaction script which is a kind of multi-signature schema. After first concept of ASM schema [5], it is improved by [3] to adding the capability of key aggregation to the schema using the pairing-based cryptography. Therefore, implementing ASM to the Bitcoin ecosystem can reduce the transaction size and computational effort through aggregating public keys and also make transactions more secure.

And also discovering a gap in Bitcoin usage about transferring the spending authority to someone else, The Delegatable Credential cryptosystem can be a solution to this. The suggested schema in [16] is also based on bilinear pairings. It gives the opportunity that the public key of issuer who is the first delegator is sufficient to verify the signature that is signed by any further delegate. It allows someone to spend another's Bitcoin amount with a delegated credential. At the same time, it is thought that the ability to maintain the user-specific attribute values can also create authorisation control mechanism via keeping user's authorisation level in an attribute and checking it at the signing stage, like how much a delegate be able to spend etc.

Although this approach has good facilities, it is not in Bitcoins transaction scripts currently. On the other hand, it brings along a difficulty to implement. Because the public keys of the all previous delegators is included in the signature, called attribute token in [16]. Like Multisig script, all public keys is not stored in locking scripts, but must be stored in unlocking script which includes the signature.

As a future work, the size issue of Delegatable Credentials schema in [16] can be overcome with developing a combined or aggregated public key structure like ASM schema. In this

way, Delegatable Credentials is likely to add Bitcoin transaction scripts. For ASM schema, in order to add more authorization capabilities, it is considered that attributes can be included in Group Setup and Signing stage.

References

- [1] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*, O'Reilly Media, Inc., **2014**.
- [2] S. Vijayakumaran, *An Introduction to Bitcoin*, **October 2017**.
- [3] D. Boneh, M. Drijvers, and G. Neven, Compact multi-signatures for smaller blockchains, *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, Cham, **2018**.
- [4] M. Bellare, G. Neven, Multi-signatures in the plain public-key model and a general forking lemma, In: Juels, A., Wright, R.N., Vimercati, S. (eds.) *ACM CCS 06: 13th Conference on Computer and Communications Security*. pp. 390–399. ACM Press, Alexandria, Virginia, USA (**Oct 30 – Nov 3, 2006**).
- [5] S. Micali, K. Ohta, L. Reyzin, Accountable-subgroup multisignatures: Extended abstract. In: *ACM CCS 01: 8th Conference on Computer and Communications Security*. pp. 245–254. ACM Press, Philadelphia, PA, USA (**Nov 5–8, 2001**).
- [6] G. Maxwell, A. Poelstra, Y. Seurin, P. Wuille, Simple schnorr multi-signatures with applications to bitcoin. *Cryptology ePrint Archive*, Report 2018/068 (**2018**).
- [7] D. Boneh, B. Lynn, H. Shacham, Short signatures from the Weil pairing. In: Boyd, C. (ed.) *Advances in Cryptology – ASIACRYPT 2001*. Lecture Notes in Computer Science, vol. 2248, pp. 514–532. Springer, Heidelberg, Germany, Gold Coast, Australia (**2001**).
- [8] A. Budroni, F. Pintore, Efficient hash maps to G2 on BLS curves. *Cryptology ePrint Archive*, Report 2017/419 (**2017**).
- [9] T. Ristenpart, S. Yilek, The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In: Naor, M. (ed.) *Advances in Cryptology – EUROCRYPT 2007*. Lecture Notes in Computer Science, vol. 4515, pp. 228–245. Springer, Heidelberg, Germany, Barcelona, Spain (**May 20–24, 2007**).
- [10] J. Camenisch, M. Drijvers, M. Dubovitskaya, Practical UC-secure delegatable credentials with attributes and their application to blockchain, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, **2017**.

- [11] J. Camenisch, A. Kiayias, and M. Yung, On the Portability of Generalized Schnorr Proofs. In EUROCRYPT 2009 (LNCS), Antoine Joux (Ed.), Vol. 5479. Springer, Heidelberg, 425–442 **2009**.
- [12] M. Abe, G. Fuchsbauer, J. Groth, K. Haralambiev, M. Ohkubo, Structure-Preserving Signatures and Commitments to Group Elements. In CRYPTO 2010 (LNCS), Tal Rabin (Ed.), Vol. 6223. Springer, Heidelberg, 209–236, **2010**.
- [13] J. Groth, Efficient Fully Structure-Preserving Signatures for Large Messages, In ASIACRYPT 2015, Part I (LNCS), Tetsu Iwata and Jung Hee Cheon (Eds.), Vol. 9452. Springer, Heidelberg, 239–259, **2015**.
- [14] Wikipedia, Pairing-based cryptography, https://en.wikipedia.org/wiki/Pairing-based_cryptography (Accessed: **June 27, 2019**).
- [15] J. Camenisch, M. Dubovitskaya, R. R. Enderlein, A. Lehmann, G. Neven, C. Paquin, F.-S. Preiss, Concepts and languages for privacy-preserving attribute-based authentication, *J. Inf. Sec. Appl.* 19, 1 , 25–44, (**2014**).
- [16] Camenisch, Jan, Manu Drijvers, and Maria Dubovitskaya. "Practical UC-secure delegatable credentials with attributes and their application to blockchain." Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. ACM, **2017**.
- [17] Boneh, Dan, Ben Lynn, and Hovav Shacham. "Short signatures from the Weil pairing." International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg, **2001**.
- [18] C.-P. Schnorr, Efficient signature generation by smart cards, *Journal of cryptology* 4.3, 161-174, **1991**.
- [19] K. Itakura, K. Nakamura, A public-key cryptosystem suitable for digital multisignatures, Tech. rep., NEC Research and Development, **1983**.
- [20] S. Park, K. Kim, D. Won, Two efficient RSA multisignature schemes, In: Han, Y., Okamoto, T., Qing, S. (eds.) ICICS 97: 1st International Conference on Information and Communication Security. Lecture Notes in Computer Science, vol. 1334, pp. 217-222. Springer, Heidelberg, Germany, Beijing, China (**Nov 11-14, 1997**).

- [21] L. Harn, Group-oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proceedings-Computers and Digital Techniques* 141(5), 307-313 (**1994**).
- [22] K. Ohta, T. Okamoto, Multi-signature schemes secure against active insider attacks. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 82(1), 21-31 (**1999**).
- [23] C. Ma, J. Weng, Y. Li, R. Deng, Efficient discrete logarithm based multisignature scheme in the plain public key model. *Designs, Codes and Cryptography* 54(2), 121-133 (**2010**).
- [24] S. Lu, R. Ostrovsky, A. Sahai, H. Shacham, B. Waters, Sequential aggregate signatures and multisignatures without random oracles. In: Vaudenay, S. (ed.) *Advances in Cryptology - EUROCRYPT 2006*. Lecture Notes in Computer Science, vol. 4004, pp. 465-485. Springer, Heidelberg, Germany, St. Petersburg, Russia (**May 28 - Jun 1, 2006**).
- [25] A. Boldyreva, C. Gentry, A. O'Neill, D.H. Yum, Ordered multisignatures and identity-based sequential aggregate signatures, with applications to secure routing. In: Ning, P., di Vimercati, S.D.C., Syverson, P.F. (eds.) *ACM CCS 07: 14th Conference on Computer and Communications Security*. pp. 276-285. ACM Press, Alexandria, Virginia, USA (**Oct 28-31, 2007**).
- [26] R.E. Bansarkhani, J. Sturm, An efficient lattice-based multisignature scheme with applications to bitcoins. In: Foresti, S., Persiano, G. (eds.) *CANS 16: 15th International Conference on Cryptology and Network Security*. Lecture Notes in Computer Science, vol. 10052, pp. 140-155. Springer, Heidelberg, Germany, Milan, Italy (**Nov 14-16, 2016**).
- [27] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, H. Shacham, Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO 2009 (LNCS)*, Shai Halevi (Ed.), Vol. 5677. Springer, Heidelberg, 108–125, **2009**.
- [28] M. Chase, M. Kohlweiss, A. Lysyanskaya, S. Meiklejohn, Malleable Signatures: Complex Unary Transformations and Delegatable Anonymous Credentials, *Cryptology ePrint Archive*, Report 2013/179, (**2013**).
- [29] M. Chase, A. Lysyanskaya, On Signatures of Knowledge. In *CRYPTO 2006 (LNCS)*, Cynthia Dwork (Ed.), Vol. 4117. Springer, Heidelberg, 78–96, **2006**.

- [30] G. Fuchsbauer, Commuting Signatures and Verifiable Encryption. In EUROCRYPT 2011 (LNCS), Kenneth G. Paterson (Ed.), Vol. 6632. Springer, Heidelberg, 224–245, **2011**.
- [31] M. Trolin and D. Wikström, Hierarchical Group Signatures. In ICALP 2005 (LNCS), Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung (Eds.), Vol. 3580. Springer, Heidelberg, 446–458, **2005**.
- [32] W. Bosma, J. J. Cannon, C. Fieker, A. Steel (eds.), Handbook of Magma functions, Edition 2.16, 5017 pages, (**2010**).

APPENDIX

Listing 1: The Magma Codes of ASM Example

```
//-----Begin Parameter Generation Stage-----//
> Zs<s> := PolynomialRing(Integers());
> ps := 36*s^4 + 36*s^3 + 24*s^2 + 6*s + 1;
> ns := 36*s^4 + 36*s^3 + 18*s^2 + 6*s + 1;
> z := 513235038556; // arbitrary initial value

> repeat
>   z := z+1;
>   p := Evaluate(ps, z);
>   q := Evaluate(ns, z);
> until IsProbablePrime(p) and IsProbablePrime(q);

> p:
2497857711095780713403056606399151275099020724723

> q:
2497857711095780713403055025937917618634473494829

> Fp := FiniteField(p);
> b := Fp!0;

> repeat
>   repeat b := b + 1; until IsSquare(b + 1);
>   y := SquareRoot(b + 1);
>   E := EllipticCurve([Fp!0, b]);
>   G := E![1, y];
> until IsZero(q*G);

> E;
Elliptic Curve defined by  $y^2 = x^3 + 18$  over
GF(2497857711095780713403056606399151275099020724723)

> t := p + 1 - q;
> k := 12; // security multiplier
```



```

> Fpk := GF(p, k);
> N := Resultant(s^k - 1, s^2 - t*s + p); // number of points over big field
> Cofac := N div q^2;

> g1 := E(Fpk)!G;
> g2 := Cofac*Random(E(Fpk)); // g2 has order q now
//-----End of Parameter Generation Stage-----//

> H0:=function(apk,m,g1)
> return ((Integers()!(Eltseq(apk[1])[1]))+m)*g1;
> end function;

> H1:=function(pk,pkS,q)
> return Integers()!(Eltseq(pk[1])[1] * Eltseq(pkS[1])[1]) mod q;
> end function;

> H2:=function(apk,i,g2)
> return ((Integers()!(Eltseq(apk[2])[1]))+i)*g1;
> end function;

//-----Begin Key Generation Stage-----//
> sk1:=11;
> pk1:=sk1*g2;

> sk2:=13;
> pk2:=sk2*g2;

> sk3:=15;
> pk3:=sk3*g2;

> sk4:=17;
> pk4:=sk4*g2;

> sk5:=19;
> pk5:=sk5*g2;
//-----End Key Generation Stage-----//

//-----Begin Group Setup Stage-----//
> a1:=H1(pk1,pk1+pk2+pk3+pk4+pk5,q);

```

```

> a2:=H1(pk2, pk1+pk2+pk3+pk4+pk5, q);
> a3:=H1(pk3, pk1+pk2+pk3+pk4+pk5, q);
> a4:=H1(pk4, pk1+pk2+pk3+pk4+pk5, q);
> a5:=H1(pk5, pk1+pk2+pk3+pk4+pk5, q);

> apk:= a1*pk1+a2*pk2+a3*pk3+a4*pk4+a5*pk5;

> mu11:= (a1*sk1)*H2(apk, 1, g1);
> mu12:= (a2*sk2)*H2(apk, 1, g1);
> mu13:= (a3*sk3)*H2(apk, 1, g1);
> mu14:= (a4*sk4)*H2(apk, 1, g1);
> mu15:= (a5*sk5)*H2(apk, 1, g1);
> mu21:= (a1*sk1)*H2(apk, 2, g1);
> mu22:= (a2*sk2)*H2(apk, 2, g1);
> mu23:= (a3*sk3)*H2(apk, 2, g1);
> mu24:= (a4*sk4)*H2(apk, 2, g1);
> mu25:= (a5*sk5)*H2(apk, 2, g1);
> mu31:= (a1*sk1)*H2(apk, 3, g1);
> mu32:= (a2*sk2)*H2(apk, 3, g1);
> mu33:= (a3*sk3)*H2(apk, 3, g1);
> mu34:= (a4*sk4)*H2(apk, 3, g1);
> mu35:= (a5*sk5)*H2(apk, 3, g1);
> mu41:= (a1*sk1)*H2(apk, 4, g1);
> mu42:= (a2*sk2)*H2(apk, 4, g1);
> mu43:= (a3*sk3)*H2(apk, 4, g1);
> mu44:= (a4*sk4)*H2(apk, 4, g1);
> mu45:= (a5*sk5)*H2(apk, 4, g1);
> mu51:= (a1*sk1)*H2(apk, 5, g1);
> mu52:= (a2*sk2)*H2(apk, 5, g1);
> mu53:= (a3*sk3)*H2(apk, 5, g1);
> mu54:= (a4*sk4)*H2(apk, 5, g1);
> mu55:= (a5*sk5)*H2(apk, 5, g1);

> mk1:= mu11+mu12+mu13+mu14+mu15;
> mk2:= mu21+mu22+mu23+mu24+mu25;
> mk3:= mu31+mu32+mu33+mu34+mu35;
> mk4:= mu41+mu42+mu43+mu44+mu45;
> mk5:= mu51+mu52+mu53+mu54+mu55;

```

```

> WeilPairing (mk1, g2, q) eq WeilPairing (H2 (apk,1,g1), apk, q);
true
> WeilPairing (mk2, g2, q) eq WeilPairing (H2 (apk,2,g1), apk, q);
true
> WeilPairing (mk3, g2, q) eq WeilPairing (H2 (apk,3,g1), apk, q);
true
> WeilPairing (mk4, g2, q) eq WeilPairing (H2 (apk,4,g1), apk, q);
true
> WeilPairing (mk5, g2, q) eq WeilPairing (H2 (apk,5,g1), apk, q);
true
//-----End Group Setup Stage-----//

//-----Begin Signing Stage-----//
> m:=12345;

> s2:= (sk2*H0 (apk,m,g1)) + mk2;
> s3:= (sk3*H0 (apk,m,g1)) + mk3;
> s5:= (sk5*H0 (apk,m,g1)) + mk5;

> pk:=pk2+pk3+pk5;
> s:=s2+s3+s5;
//-----End Signing Stage-----//

//-----Begin Verification Stage-----//
> WeilPairing (H0 (apk,m,g1), pk, q)
  *WeilPairing (H2 (apk,2,g1)+H2 (apk,3,g1)+H2 (apk,5,g1), apk, q)
  eq WeilPairing (s, g2, q)
true
//-----End Verification Stage-----//

```

Listing 2: The Magma Codes of ASM with PoPs Example

```
//-----Begin Parameter Generation Stage-----//
> Zs<s> := PolynomialRing(Integers());
> ps := 36*s^4 + 36*s^3 + 24*s^2 + 6*s + 1;
> ns := 36*s^4 + 36*s^3 + 18*s^2 + 6*s + 1;
> z := 513235038556; // arbitrary initial value

> repeat
>   z := z+1;
>   p := Evaluate(ps, z);
>   q := Evaluate(ns, z);
> until IsProbablePrime(p) and IsProbablePrime(q);

> p:
2497857711095780713403056606399151275099020724723

> q;
2497857711095780713403055025937917618634473494829

> Fp := FiniteField(p);
> b := Fp!0;

> repeat
>   repeat b := b + 1; until IsSquare(b + 1);
>   y := SquareRoot(b + 1);
>   E := EllipticCurve([Fp!0, b]);
>   G := E![1, y];
> until IsZero(q*G);

> E;
Elliptic Curve defined by  $y^2 = x^3 + 18$  over
GF(2497857711095780713403056606399151275099020724723)

> t := p + 1 - q;
> k := 12; // security multiplier
> Fpk := GF(p, k);
> N := Resultant(s^k - 1, s^2 - t*s + p); // number of points over big field
> Cofac := N div q^2;
```

```

> g1 := E(Fpk)!G;
> g2 := Cofac*Random(E(Fpk)); // g2 has order q now
//-----End Parameter Generation Stage-----//

> H0:=function(apk,m,g1)
> return ((Integers()!(Eltseq(apk[1])[1]))+m)*g1;
> end function;

> H1:=function(pk,pkS,q)
> return Integers()!(Eltseq(pk[1])[1] * Eltseq(pkS[1])[1]) mod q;
> end function;

> H2:=function(apk,i,g2)
> return ((Integers()!(Eltseq(apk[2])[1]))+i)*g1;
> end function;

> H3:=function(y,q)
> return ((Integers()!(Eltseq(y[3])[1]))) mod q;
> end function;

//-----Begin Key Generation Stage-----//
> sk1:=11;
> x1:=20;
> y1:=sk1*g2;
> pi1:=x1*H3(y1,q);

> sk2:=13;
> x2:=22;
> y2:=sk2*g2;
> pi2:=x2*H3(y2,q);

> sk3:=15;
> x3:=24;
> y3:=sk3*g2;
> pi3:=x3*H3(y3,q);

> sk4:=17;
> x4:=26;

```

```

> y4:=sk4*g2;
> pi4:=x4*H3(y4,q);

> sk5:=19;
> x5:=28;
> y5:=sk5*g2;
> pi5:=x5*H3(y5,q);
//-----End Key Generation Stage-----//

//-----Begin Group Setup Stage-----//
> Y:=y1+y2+y3+y4+y5;
> h:=H3(pi1*y1+pi2*y2+pi3*y3+pi4*y4+pi5*y5,q);

> apk:= h*Y;

> mu11:= sk1*H2(apk,1,g1);
> mu12:= sk2*H2(apk,1,g1);
> mu13:= sk3*H2(apk,1,g1);
> mu14:= sk4*H2(apk,1,g1);
> mu15:= sk5*H2(apk,1,g1);
> mu21:= sk1*H2(apk,2,g1);
> mu22:= sk2*H2(apk,2,g1);
> mu23:= sk3*H2(apk,2,g1);
> mu24:= sk4*H2(apk,2,g1);
> mu25:= sk5*H2(apk,2,g1);
> mu31:= sk1*H2(apk,3,g1);
> mu32:= sk2*H2(apk,3,g1);
> mu33:= sk3*H2(apk,3,g1);
> mu34:= sk4*H2(apk,3,g1);
> mu35:= sk5*H2(apk,3,g1);
> mu41:= sk1*H2(apk,4,g1);
> mu42:= sk2*H2(apk,4,g1);
> mu43:= sk3*H2(apk,4,g1);
> mu44:= sk4*H2(apk,4,g1);
> mu45:= sk5*H2(apk,4,g1);
> mu51:= sk1*H2(apk,5,g1);
> mu52:= sk2*H2(apk,5,g1);
> mu53:= sk3*H2(apk,5,g1);
> mu54:= sk4*H2(apk,5,g1);

```

```

> mu55:= sk5*H2(apk,5,g1);

> mk1:=mu11+mu12+mu13+mu14+mu15;
> mk2:=mu21+mu22+mu23+mu24+mu25;
> mk3:=mu31+mu32+mu33+mu34+mu35;
> mk4:=mu41+mu42+mu43+mu44+mu45;
> mk5:=mu51+mu52+mu53+mu54+mu55;

> WeilPairing(mk1, g2, q) eq WeilPairing(H2(apk,1,g1), Y, q);
true
> WeilPairing(mk2, g2, q) eq WeilPairing(H2(apk,2,g1), Y, q);
true
> WeilPairing(mk3, g2, q) eq WeilPairing(H2(apk,3,g1), Y, q);
true
> WeilPairing(mk4, g2, q) eq WeilPairing(H2(apk,4,g1), Y, q);
true
> WeilPairing(mk5, g2, q) eq WeilPairing(H2(apk,5,g1), Y, q);
true
//-----End Group Setup Stage-----//

//-----Begin Signing Stage-----//
> m:=12345;

> s2:= (sk2*H0(apk,m,g1)) + mk2;
> s3:= (sk3*H0(apk,m,g1)) + mk3;
> s5:= (sk5*H0(apk,m,g1)) + mk5;

> y:=y2+y3+y5;
> s:=s2+s3+s5;
//-----End Signing Stage-----//

//-----Begin Verification Stage-----//
> WeilPairing(H0(apk,m,g1), y, q)
*WeilPairing(H2(apk,2,g1)+H2(apk,3,g1)+H2(apk,5,g1), apk, q)
eq WeilPairing(s, g2, q)
true
//-----End Verification Stage-----//

```

Listing 3: The pseudocode of presenting

input: $\langle r_i, s_i, \langle t_{i,j} \rangle_{j=1}^{n_i+1} \rangle_{i=1}^L, csk_L, \langle cpk_i \rangle_{i=1}^L, \langle a_{i,j} \rangle_{i=1, \dots, L, j=1, \dots, n_i}, D, sp, m$
for $i = 1, \dots, L$ **do**
 $\rho_{\sigma_i} \xleftarrow{\$} \mathbb{Z}_q, r'_1 \leftarrow r_i^{\rho_{\sigma_i}}, s'_i \leftarrow s_i^{\frac{1}{\rho_{\sigma_i}}}$
 for $j = 1, \dots, n_i + 1$ **do**
 $t'_{i,j} \leftarrow t_{i,j}^{\frac{1}{\rho_{\sigma_i}}}$
 end for
end for
 $\langle \rho_{s_i}, \langle \rho_{t_{i,j}} \rangle_{j=1}^{n_i+1}, \langle \rho_{a_{i,j}} \rangle_{j=1}^{n_i} \rangle_{i=1}^L, \langle \rho_{cpk_i} \rangle_{i=1}^{L-1}, \rho_{csk_L} \xleftarrow{\$} \mathbb{Z}_q$
for $i = 1, 3, \dots, L$ **do**
 $com_{i,1} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i} \cdot \rho_{s_i}} [\cdot e(g_1^{-1}, g_2)^{\rho_{cpk_{i-1}}}]_{i \neq 1}$
 $com_{i,2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i} \cdot \rho_{t_{i,1}}} \cdot e(g_1, g_2^{-1})^{\rho_{cpk_i}} [\cdot e(y_{1,1}, g_2)^{\rho_{cpk_{i-1}}}]_{i \neq 1}$
 for $j = 1, \dots, n_i$ **do**
 if $(i, j) \in D$ **then**
 $com_{i,j+2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i} \cdot \rho_{t_{i,j+1}}} [\cdot e(y_{1,j+1}, g_2)^{\rho_{cpk_{i-1}}}]_{i \neq 1}$
 else
 $com_{i,j+2} \leftarrow e(g_1, r_i)^{\rho_{\sigma_i} \cdot \rho_{t_{i,j+1}}} \cdot e(g_1, g_2^{-1})^{\rho_{a_{i,j}}} [\cdot e(y_{1,j+1}, g_2)^{\rho_{cpk_{i-1}}}]_{i \neq 1}$
 end if
 end for
end for
for $i = 2, 4, \dots, L$ **do**
 $com_{i,1} \leftarrow e(r_i, g_2)^{\rho_{\sigma_i} \cdot \rho_{s_i}} e(g_1, g_2^{-1})^{\rho_{cpk_{i-1}}}$
 $com_{i,2} \leftarrow e(r_i, g_2)^{\rho_{\sigma_i} \cdot \rho_{t_{i,1}}} e(g_1, y_{2,1}^{-1})^{\rho_{cpk_{i-1}}} e(g_1^{-1}, g_2)^{\rho_{cpk_i}}$
 for $j = 1, \dots, n_i$ **do**
 if $(i, j) \in D$ **then**
 $com_{i,j+2} \leftarrow e(g_1, y_{2,j+1}^{-1})^{\rho_{cpk_{i-1}}} \cdot e(r_i, g_2)^{\rho_{\sigma_i} \cdot \rho_{t_{i,j+1}}}$
 else
 $com_{i,j+2} \leftarrow e(g_1, y_{2,j+1}^{-1})^{\rho_{cpk_{i-1}}} \cdot e(r_i, g_2)^{\rho_{\sigma_i} \cdot \rho_{t_{i,j+1}}} \cdot e(g_1^{-1}, g_2)^{\rho_{a_{i,j}}}$
 end if
 end for
end for
 $c \leftarrow H(sp, ipk, \langle r'_i, \langle com_{i,j} \rangle_{j=1}^{n_i+2} \rangle_{i=1}^L, \langle a_{i,j} \rangle_{(i,j) \in D}, m)$
for $i = 1, 3, \dots, L$ **do**
 $res_{s_i} = g_1^{\rho_{s_i}} s_i^c, [res_{cpk_i} = g_1^{\rho_{cpk_i}} cpk_i^c]_{i \neq L}, [res_{csk_i} = \rho_{cpk_i} + c \cdot csk_i]_{i=L}$
 for $j = 1, \dots, n_i + 1$ **do**
 $res_{t_{i,j}} = g_1^{\rho_{t_{i,j}}} t_{i,j}^c$
 end for
 for $j = 1, \dots, n_i$ **with** $(i, j) \notin D$ **do**
 $res_{a_{i,j}} = g_1^{\rho_{a_{i,j}}} a_{i,j}^c$
 end for
end for
for $i = 2, 4, \dots, L$ **do**
 $res_{s_i} = g_2^{\rho_{s_i}} s_i^c, [res_{cpk_i} = g_2^{\rho_{cpk_i}} cpk_i^c]_{i \neq L}, [res_{csk_i} = \rho_{cpk_i} + c \cdot csk_i]_{i=L}$
 for $j = 1, \dots, n_i + 1$ **do**
 $res_{t_{i,j}} = g_2^{\rho_{t_{i,j}}} t_{i,j}^c$
 end for
 for $j = 1, \dots, n_i$ **with** $(i, j) \notin D$ **do**
 $res_{a_{i,j}} = g_2^{\rho_{a_{i,j}}} a_{i,j}^c$
 end for
end for
output: $c, \langle r'_i, res_{s_i}, \langle res_{t_{i,j}} \rangle_{j=1}^{n_i+1} \rangle_{i=1}^L, \langle res_{a_{i,j}} \rangle_{(i,j) \notin D}, \langle res_{cpk_i} \rangle_{i=1}^{L-1}, res_{csk_L}$

Listing 4: The pseudocode of verifying

input: $c, \langle r'_i, \text{res}_{s_i}, \langle \text{res}_{t_{i,j}} \rangle_{j=1}^{n_i+1} \rangle_{i=1}^L, \langle \text{res}_{a_{i,j}} \rangle_{(i,j) \notin D}, \langle \text{res}_{\text{cpk}_i} \rangle_{i=1}^{L-1}, \text{res}_{\text{csk}_L},$
 $\langle a_{i,j} \rangle_{(i,j) \in D}, D, sp, m$

for $i = 1, 3, \dots, L$ **do**

$\text{com}_{i,1} \leftarrow \text{fexp}(\hat{t}(\text{res}_{s_i}, r'_i) [\cdot \hat{t}(g_1^{-1}, \text{res}_{\text{cpk}_{i-1}})]_{i \neq 1}) \cdot (e(y_{1,1}, g_2) [\cdot e(g_1, \text{ipk})]_{i=1})^{-c}$

$\text{com}_{i,2} \leftarrow \text{fexp}(\hat{t}(\text{res}_{t_{i,1}}, r'_i) [\cdot \hat{t}(y_{1,1}, \text{res}_{\text{cpk}_{i-1}})]_{i \neq 1} [\cdot \hat{t}(\text{res}_{\text{cpk}_i}, g_2^{-1})]_{i \neq L} [\cdot e(g_1, g_2^{-1})^{\text{res}_{\text{csk}_i}}]_{i=L} [\cdot e(y_{1,1}, \text{ipk})^{-c}]_{i=1})$

for $j = 1, \dots, n_i$ **do**

if $(i, j) \in D$ **then**

$\text{com}_{i,j+2} \leftarrow \text{fexp}(\hat{t}(\text{res}_{t_{i,j+1}}, r'_i) [\cdot \hat{t}(y_{1,j+1}, \text{res}_{\text{cpk}_{i-1}})]_{i \neq 1}) \cdot (e(a_{i,j}, g_2) [e(y_{1,j+1}, \text{ipk})]_{i=1})^{-c}$

else

$\text{com}_{i,j+2} \leftarrow \text{fexp}(\hat{t}(\text{res}_{t_{i,j+1}}, r'_i) \cdot \hat{t}(\text{res}_{a_{i,j}}, g_2^{-1}) [\cdot \hat{t}(y_{1,j+1}, \text{res}_{\text{cpk}_{i-1}})]_{i \neq 1} [\cdot e(y_{1,j+1}, \text{ipk})^{-c}]_{i=1})$

end if

end for

end for

for $i = 2, 4, \dots, L$ **do**

$\text{com}_{i,1} \leftarrow \text{fexp}(\hat{t}(r'_i, \text{res}_{s_i}) \cdot \hat{t}(\text{res}_{\text{cpk}_{i-1}}, g_2^{-1})) \cdot e(g_1, y_{2,1})^{-c}$

$\text{com}_{i,2} \leftarrow \text{fexp}(\hat{t}(r'_i, \text{res}_{t_{i,1}}) \cdot \hat{t}(\text{res}_{\text{cpk}_{i-1}}, y_{2,1}^{-1}) [\cdot \hat{t}(g_1^{-1}, \text{res}_{\text{cpk}_i})]_{i \neq L} [\cdot e(g_1^{-1}, g_2)^{\text{res}_{\text{csk}_i}}]_{i=L})$

for $j = 1, \dots, n_i$ **do**

if $(i, j) \in D$ **then**

$\text{com}_{i,j+2} \leftarrow \text{fexp}(\hat{t}(\text{res}_{\text{cpk}_{i-1}}, y_{2,j+1}^{-1}) \cdot \hat{t}(r'_i, \text{res}_{t_{i,j+1}})) \cdot e(g_1, a_{i,j})^{-c}$

else

$\text{com}_{i,j+2} \leftarrow \text{fexp}(\hat{t}(\text{res}_{\text{cpk}_{i-1}}, y_{2,j+1}^{-1}) \cdot \hat{t}(r'_i, \text{res}_{t_{i,j+1}}) \cdot \hat{t}(g_1^{-1}, \text{res}_{a_{i,j}}))$

end if

end for

end for

$c' \leftarrow H(sp, \text{ipk}, \langle r'_i, \langle \text{com}_{i,j} \rangle_{j=1}^{n_i+2} \rangle_{i=1}^L, \langle a_{i,j} \rangle_{(i,j) \in D}, m)$

output: $c = c'$



HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
THESIS/DISSERTATION ORIGINALITY REPORT

HACETTEPE UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING
TO THE DEPARTMENT OF MATHEMATICS

Date:22/07/2019

Thesis Title / Topic: **Alternative Digital Signature Schemas in Blockchain**

According to the originality report obtained by ~~myself~~/my thesis advisor by using the *Turnitin* plagiarism detection software and by applying the filtering options stated below on 22/07/2019 for the total of 47 pages including the a) Title Page, b) Introduction, c) Main Chapters, d) Conclusion sections of my thesis entitled as above, the similarity index of my thesis is 8%.

Filtering options applied:

1. Bibliography/Works Cited excluded
2. Quotes included
3. Match size up to 5 words excluded

I declare that I have carefully read Hacettepe University Graduate School of Science and Engineering Guidelines for Obtaining and Using Thesis Originality Reports; that according to the maximum similarity index values specified in the Guidelines, my thesis does not include any form of plagiarism; that in any future detection of possible infringement of the regulations I accept all legal responsibility; and that all the information I have provided is correct to the best of my knowledge.

I respectfully submit this for approval.

Name Surname: Fahrettin YAVUZYİĞİT

Student No: N10227825

Department: Mathematics

Program: Mathematics

Status: Masters Ph.D. Integrated
Ph.D.

Date and Signature

22.07.2019

ADVISOR APPROVAL

APPROVED.

(Title, Name Surname,
Signature)

Assoc. Prof. Dr. Oguz Toyler

CURRICULUM VITAE

Credentials

Name, Surname : Fahrettin YAVUZYIĞİT
Place of Birth : İstanbul, 1982
Marital Status : Married
E-mail : fahrettinyavuziyigit@gmail.com

Education

High School : 1996-1999 Kaya Bayazitođlu High School
BSc. : 2000-2007 Hacettepe University, Faculty of Science, Department of
Mathematics
MSc. : 2007-2019 Hacettepe University, Institute of Graduate Studies in Science,
Department of Mathematics

Foreign Languages

English

Work Experience

2008- Information Systems Senior Controller, T. Halk Bankası A.Ş.