UAB
Universitat Autònoma
de Barcelona

Bachelor Thesis

User Account Access Graphs in Polkadot

Author:
Carlo Sala Gancho

Supervisors:
Joaquim Roé Vellvé
Jordi Herrera Joancomartí

Bachelor's Degree in Mathematics
Departament de Matemàtiques
Facultat de Ciències

June 25, 2024

A chain is only as strong as its weakest link.

Thomas Reid

Acknowledgments

I would like to express my gratitude to my supervisors, Joaquim Roé Vellvé and Jordi Herrera Joancomartí, for their support, guidance, and shared expertise throughout the development of this thesis. Their feedback and constructive criticism were fundamental to the completion of this study.

I would like to thank all my professors and colleagues at the university for sharing their knowledge and passion for Mathematics and every aspect that surrounds it.

I would like to thank as well my partner for her encouragement and support during all these difficult months upon the completion of this thesis.

Contents

1 Introduction **1**

2 Tensors in a nutshell **2**
2.1 Definitions . . . 2
2.2 Tensor product . . . 2

3 Directed hypergraphs **7**
3.1 Definitions . . . 7
3.2 Node reachability . . . 8
3.2.1 Prior work . . . 8
3.2.2 Adjacency tensor . . . 9
3.2.3 Other ideas on the reachability map . . . 12

4 User Account Access Graphs **13**
4.1 Model description . . . 13
4.2 Account reachability . . . 13
4.3 Scoring schemes . . . 14
4.3.1 An attacker based scoring scheme . . . 16
4.4 Recoverability . . . 16
4.4.1 A lockout scoring scheme . . . 17
4.4.2 Backdoor . . . 18

5 Case study: Polkadot **19**
5.1 Wallet accounts . . . 19
5.1.1 Browser extension wallets . . . 20
5.1.2 Hardware wallet: Ledger . . . 23
5.1.3 Hardware wallet: Polkadot Vault . . . 26
5.1.4 General conclusions . . . 27
5.2 Proxy accounts . . . 27
5.3 Multisignature accounts . . . 28
5.3.1 Recovery pallet . . . 28

6 Conclusion **29**

References **30**

1 Introduction

The security of digital assets has become increasingly critical as blockchain technology and cryptocurrencies gain widespread adoption. Polkadot exemplifies the innovative approaches in its field. This thesis goes deep into user account setups within the Polkadot ecosystem.

To provide a comprehensive analysis, we explore tensors and directed hypergraphs as base tools for modeling the access to user accounts. By leveraging these mathematical objects, we aim to propose a robust model that captures the nuances of account security and recoverability.

We incorporate a case study of different Polkadot wallet alternatives; such as Ledger, Polkadot Vault, among others; identifying potential security flaws and proposing improvements to enhance the user experience. Through this study, we seek to contribute to the broader understanding of securing digital assets in decentralized networks, offering insights that could influence future developments in blockchain security protocols.

This work is structured to first lay the mathematical groundwork, followed by the application of these concepts to formally define User Account Access Graphs. We conclude with the Polkadot ecosystem case study.

2 Tensors in a nutshell

2.1 Definitions

Tensors are going to play a fundamental role in the following sections, especially the tensor product. The goal of this section is to set the basis and to define the tensor product.

Tensor calculus [Wik24c] was first introduced and developed by Ricci-Curbastro and Levi-Civita in 1892 and 1900 respectively, and Einstein popularized it with the introduction of his theory of general relativity. After that, tensors have been applied widely to several fields of physics, such as mechanics, electrodynamics, among others. Besides these classical applications, they have been used lately on unsupervised machine learning [RSG17] and multi-relational data analysis.

We will use the notes from K. Conrad [Con] for most of the definitions, propositions and theorems; but we will use fields instead of commutative rings, and vector spaces instead of modules for convenience. From now on, let K be a field.

Definition 2.1 (Bilinear map). A map $B : E \times F \to G$, where E, F, G are K-vector spaces, is called *bilinear* if it is K-linear in each argument when the other one fixed. That is:

$$B(e_1 + e_2, f) = B(e_1, f) + B(e_2, f), \qquad B(re, f) = rB(e, f) \tag{1}$$
$$B(e, f_1 + f_2) = B(e, f_1) + B(e, f_2), \qquad B(e, rf) = rB(e, f) \tag{2}$$

We will give some examples of bilinear maps:

1. The multiplication over K is bilinear: $K \times K \to K$
2. Dot product $\mathbf{v} \cdot \mathbf{w}$ on $\mathbb{R}^n$ is bilinear: $\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$

We are now ready to define the multilinear map:

Definition 2.2 (Multilinear map). A map $B : E_1 \times \cdots \times E_k \to G$, where $E_1, \ldots, E_k, G$ are K-vector spaces, is called *k-multilinear* if $B(e_1, \ldots, e_k)$ is K-linear in each e_i when the other coordinates are fixed. Note that 1-multilinear means linear, and 2-multilinear means bilinear.

Again, some examples of k-multilinear maps:

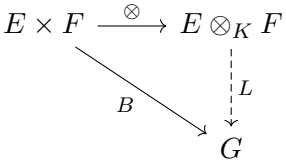
1. The multiplication with k factors over K is k-multilinear: $K \times \cdots \times K \to K$
2. Scalar triple product $\mathbf{u} \cdot (\mathbf{v} \times \mathbf{w})$ on $\mathbb{R}^n$ is trilinear: $\mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$
3. Determinant of a $n \times n$ matrix, seen as a function on the n-tuple of columns (or rows), which is an element of $(K^n)^n$ is n-multilinear.

Having these definitions, we are ready to define the tensor product.

2.2 Tensor product

Definition 2.3 (Tensor product). Let E, F, G be K-vector spaces. The *tensor product* $E \otimes_K F$ is a K-vector space equipped with a bilinear map $E \times F \xrightarrow{\otimes_K} E \otimes_K F$ such that for each bilinear map $E \times F \xrightarrow{B} G$ there is a unique linear map $E \otimes_K F \xrightarrow{L} G$ making

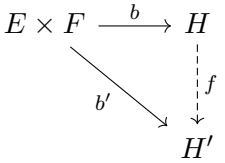
the following diagram commute. This property is called *universality*.



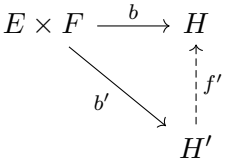
(3)

We will give a proof of its uniqueness and a construction of the tensor product. Complete proofs could be found at [Rom08] or [Con].

Uniqueness. Let H, H' be K-vector spaces, and $b\colon E \times F \to H$ and $b'\colon E \times F \to H'$ maps that satisfy the property of the tensor product. From universality, a unique linear map $f\colon H \to H'$ (same for $f'\colon H' \to H$) makes the following diagrams commute: Combining

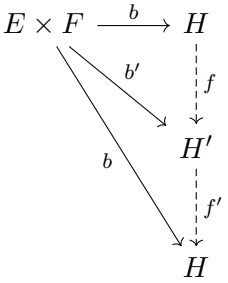


(4)



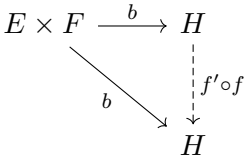
(5)

both Eq. (4) and Eq. (5) in one diagram:



(6)

And, merging that middle H', we have



(7)

From universality of (H, b), a unique linear map fits in Eq. (7). Since the identity map works, we have $f' \circ f = \mathrm{id}_H$. Similarly, $f \circ f' = \mathrm{id}_{H'}$ by stacking Eq. (4) and Eq. (5) in the other order. Thus H and H' are isomorphic vector spaces by f and also $f \circ b = b'$, which means f identifies b with b'. Therefore, two tensor products of E and F can be identified with each other in a unique way compatible with the distinguished bilinear maps to them from E and F. □

Existence. Let $E \times F$ be a K-vector space.

Then, we take D as the subspace of $K^{E \times F}$ spanned by all the elements

$$\delta_{(e+e',f)} - \delta_{(e,f)} - \delta(e', f), \qquad \delta_{(e,f+f')} - \delta_{(e,f)} - \delta_{(e,f')} \tag{8}$$

$$\delta_{(re,f)} - \delta(e, rf), \qquad r\delta_{(e,f)} - \delta(re, f), \qquad r\delta_{(e,f)} - \delta(e, rf) \tag{9}$$

where $\delta_{(e,f)}$ is the standard basis vector of $K^{E\times F}$, corresponding to $(e,f)\in E\times F$. The quotient module will serve as the tensor product:

$$E\otimes_K F \coloneqq K^{E\times F}/D \tag{10}$$

We write the coset $\delta_{(e,f)}+D$ in $E\otimes_K F$ as $e\otimes f$.

Therefore, from the definition of D, we find relations in $(E\times F)/D$ like

$$\delta_{(e+e',f)} \equiv \delta_{(e,f)}+\delta_{(e',f)} \bmod D \tag{11}$$

It can be written as well as

$$(e+e')\otimes f = e\otimes f + e'\otimes f \tag{12}$$

in $E\otimes_K F$. These relations are indeed the reason D is chosen this way, and they indeed show that the map $E\times F \xrightarrow{\otimes_K} E\otimes_K F$ given by $(e,f)\mapsto e\otimes f$ is bilinear.

This construction is compliant with Eq. (3), we recommend reading the proof found at Chapter 14 of [Rom08] or in [Con]. □

The elements of $E\otimes_K F$ are called *tensors*, and will be denoted using the letter t. Tensors with the form $e\otimes f$ are called *elementary tensors*. In fact, each tensor is a *finite sum* of elementary tensors.

It is natural to think that a basis for the tensor product will be the tensors made by members of the base for both E and F. We will show it:

Theorem 2.4. Let E, F be two K-vector spaces, with respective bases $\{e_i\}_{i\in I}$ and $\{f_j\}_{j\in J}$. Then $E\otimes_K F$ is a K-vector space with basis $\{e_i\otimes f_j\}_{(i,j)\in I\times J}$.

Proof. E or F is zero if and only if its basis is empty. In such case, the tensor product is zero and the result is clear. Let them both be non-empty.
First of all we will verify that the tensor product is spanned by the elementary tensors $e_i\otimes f_j$. Indeed, an elementary tensor has the form $e\otimes f$. Take $e=\sum_i a_i x_i$ and $f=\sum_j b_j y_j$ (they are elements of a K-vector space, we are fine), where a_i's and b_j's are 0 for all but a finite number of i and j. Then, from the bilinearity of $\otimes$, we have

$$e\otimes f = \sum_i a_i x_i \otimes \sum_j b_j y_j = \sum_{i,j} a_i b_j x_i\otimes y_j \tag{13}$$

which is a linear combination of the tensors $x_i\otimes y_j$. Every tensor is a sum of elementary tensors. Then, we have that $e_i\otimes f_j$ span $E\otimes F$. We want to see that this spanning set is a base.
To show this spanning set is linearly independent, assume that $\sum_{i,j} c_{i,j} e_i\otimes f_j = 0$, where all but finitely many $c_{i,j}$ are 0. We want to show every $c_{i,j}$ is indeed 0. Now pick two arbitrary vectors out of the basis e_{i_0} and f_{j_0} in E and F respectively. Consider the bilinear function $E\times F\to K$ by $(v,w)\mapsto v_{i_0}w_{i_0}$ where $v=\sum_i v_i e_i$ and $w=\sum_j w_j f_j$ (here v_i and w_j are coordinates in K). By the universal mapping property of tensor products there is a linear map $f_0 : E\otimes_K F\to K$ such that $f_0(v\otimes w)=v_{i_0}w_{j_0}$ on each elementary tensor $v\otimes w$.

$$\begin{array}{ccc} E\times F & \xrightarrow{\otimes} & E\otimes_K F \\ & \searrow^{(v,w)\mapsto v_{i_0}w_{i_0}} & \downarrow f_0 \\ & & K \end{array} \tag{14}$$

In particular, $f_0(e_{i_0} \otimes f_{j_0}) = 1$ and $f_0(e_i \otimes f_j) = 0$ for any $(i,j) \neq (i_0, j_0)$. Applying f_0 to the equation $\sum_{i,j} c_{i,j} e_i \otimes f_j = 0$ in $E \otimes_K F$ tells us $c_{i_0 j_0} = 0$ in K. Since the selection of coordinates was arbitrary, all the coefficients are 0. □

The tensor product can be extended to allow more than two factors. Given n K-vector spaces $E_1, \ldots, E_n$ there is a K-vector space $E_1 \otimes_K \cdots \otimes_K E_n$ that is universal for n-multilinear maps. That is, it admits a n-multilinear map $E_1 \times \cdots \times E_n \xrightarrow{\otimes_K} E_1 \otimes_K \cdots \otimes_K E_n$ and every n-multilinear map out of $E_1 \times \cdots \times E_n$ factors through this by composition with a unique linear map out of $E_1 \otimes_K \cdots \otimes_K E_n$:

$$\begin{array}{ccc} E_1 \times \cdots \times E_n & \xrightarrow{\otimes_K} & E_1 \otimes_K \cdots \otimes_K E_n \\ & \searrow^{\text{multilinear}} & \downarrow \exists \text{ unique linear} \\ & & G \end{array} \tag{15}$$

We write the image of $(e_1, \ldots, e_n)$ in $E_1 \otimes_K \cdots \otimes_K E_n$ is written $e_1 \otimes \cdots \otimes e_n$. This n-fold tensor product can be constructed as a quotient (as we constructed it in Definition 2.3) or using nested tensor products, two at a time:

$$(\cdots((E_1 \otimes_K E_2) \otimes_K E_3) \otimes_K \cdots) \otimes_K E_n \tag{16}$$

Important examples of the n-fold tensor product are *tensor powers* $E^{\otimes n}$ of a single K-vector space E:

$$E^{\otimes 0} = K, \quad E^{\otimes 1} = E, \quad E^{\otimes 2} = E \otimes_K E, \tag{17}$$

and so on. Note that $E^{\otimes 0}$ is a convention.

Theorem 2.5. Let E be a K-vector space with basis $\{e_i\}_{i \in I}$. For $n \geq 1$, the nth tensor power $E^{\otimes n}$ is a K-vector space with basis $\{e_{i_1} \otimes \cdots \otimes e_{i_n}\}_{(i_1, \ldots, i_n) \in I^n}$.

Proof. This is similar to the proof of Theorem 2.4. □

We can see as well some interesting properties of tensor products. We will not prove them, proofs for them could be found in section 5 of [Con].

Proposition 2.6. There is a unique isomorphism $E \otimes_K F \to F \otimes_K E$ where $e \otimes f \mapsto f \otimes e$.

Proposition 2.7. There is a unique isomorphism $(E \otimes_K F) \otimes_K G \to E \otimes_K (F \otimes_K G)$ where $(e \otimes f) \otimes g \mapsto e \otimes (f \otimes g)$.

Proposition 2.8. There is a unique isomorphism

$$E \otimes_K (F \oplus G) \longrightarrow (E \otimes_K F) \oplus (E \otimes_K G) \tag{18}$$

where $e \otimes (f, g) \mapsto (e \otimes f, e \otimes g)$.

We need an important result (we will not prove it, a proof is found in section 5 of [Con]) that will be really useful in the next sections to accomplish the purpose of tensor products in the formalism we will introduce.

We need to define the dual vector space of a vector space before.

Definition 2.9 (Dual vector space). Let E be a K-vector space. The set of linear maps from E to K is called the *dual vector space of* E (or *dual space of* E) and is denoted as E^*.

Theorem 2.10. Let E be a K-vector space with $\dim E < \infty$. There is a natural (basis-free) isomorphism $\mathrm{End}(E)$[1] $\cong E^* \otimes_K E$ sending each elementary tensor $\varphi \otimes e$ to the endomorphism $E \to E$ defined by $(\varphi \otimes e)(x) = \varphi(x)e$.

[1] The vector space of all endomorphisms (linear maps from a space to itself).

3 Directed hypergraphs

3.1 Definitions

Hypergraphs (introduced in 1973 by [Ber73]), understood as a natural generalization of graphs, play an important role in computer science. They are often used to describe database relations, transportation networks, etc. Just as Hypergraphs generalize graphs, a Directed Hypergraph generalizes the concept of a digraph. Some basics need to be defined before all.

Definition 3.1 (Hyperedge). [Gal+93] Let V a set of vertices. A tuple $e := (H(e), T(e))$, where $H(e), T(e) \subset V$, $H(e), T(e) \neq \varnothing$ and $H(e) \cap T(e) = \varnothing$ is called *hyperedge*. $H(e)$ is called the *head* of the hyperedge, and $T(e)$ is called the *tail*. If $|H(e)| = 1$, e is called *B-hyperedge*. Analogously, if $|T(e)| = 1$, e is called *F-hyperedge*.

Definition 3.2 (Directed hypergraph). [Gal+93] Let V be a set of vertices and E a set of hyperedges. The tuple $\mathcal{H} := (V, E)$ is called *directed hypergraph*.
If every edge of E is a B-hyperedge (F-hyperedge), then it is called *B-hypergraph* (*F-hypergraph*).
From now on, we will abuse the notation and consider a hypergraph to be a directed hypergraph.

We will use the concept of *leaf* further on this paper. Even if it is straight-forward term, it is used with different meanings across different references.

Definition 3.3 (Leaf). Let $\mathcal{H} = (V, E)$ a hypergraph. A vertex $v \in V$ is called *leaf* if $\forall e \in E \; v \notin H(e)$ (i.e. there are no edges that head to v).

Definition 3.4 (Incidence matrix). [Gal+93] Let $\mathcal{H} = (V, E)$, $V = \{v_1, \ldots, v_n\}$, $E = \{e_1, \ldots, e_m\}$ be a hypergraph. The matrix $\mathcal{M}_{n \times m}$ with coefficients in $\{-1, 0, 1\}$ defined as

$$\mathcal{M}_{i,j} = \begin{cases} -1 & \text{if } v_i \in T(e_j) \\ 1 & \text{if } v_i \in H(e_j) \\ 0 & \text{otherwise} \end{cases} \tag{19}$$

is called the *incidence matrix* of the hypergraph.

Lemma 3.5. A hypergraph is completely determined by its incidence matrix.

Proof. A hypergraph is defined by its vertices V and its edges E. The rows of the matrix determine the vertices, and every column determines an edge. The only problem could be overlapping (an edge that starts and ends at the same vertex) but, by definition, $\forall e \in E \; H(e) \cap T(e) = \varnothing$, and therefore every coefficient of the matrix is properly defined, i.e. there is a one-to-one correspondence between matrices in $\{-1, 0, 1\}$ and hypergraphs. □

We will introduce an example. We will take a hypergraph with 4 vertices and three edges, with $\mathcal{M}$ as incidence matrix:

$$\mathcal{M} = \begin{pmatrix} -1 & -1 & 1 \\ 1 & -1 & -1 \\ 0 & -1 & -1 \\ 0 & 1 & 0 \end{pmatrix} \tag{20}$$

Its visual representation is shown at Fig. 1.

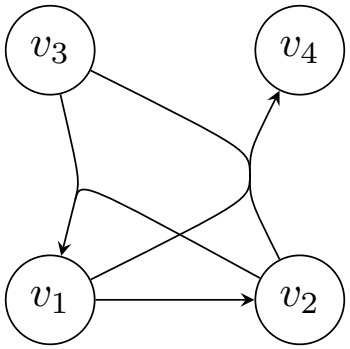


Figure 1: Example of Hypergraph.

While the incidence matrix works well to store the hypergraph information in a fairly compact way there is no much information easily accessible out of it. In the next section we want to find a way of assessing the reachability of a vertex from an initial set of vertices.

3.2 Node reachability

Just like in digraphs [BM08], reachability ([BM08] names it connection instead) is usually defined using paths. For the hypergraph case, the definition needs to be extended because edges might have a number of vertices in their tails bigger than one, and the definition needs to include not only if a vertex is reachable from another vertex, but also if a vertex is reachable from a set of vertices.

Definition 3.6 (Adjacency). Let $\mathcal{H} = (V, E)$ be a hypergraph, $v \in V$ a vertex, and $W \subset V$ a set of vertices. We say that v *is adjacent to* W if $\exists\, e \in E$ that $v \in H(e)$ and $T(e) \subset W$. We also say that the edge e is adjacent to W.
Note: We consider that v is adjacent from W if $v \in W$ as well for convenience.

Definition 3.7 (Hyperpath). Let $\mathcal{H} = (V, E)$ be a hypergraph, $v \in V$ a vertex, and $W \subset V$ a set of vertices. A finite non-null sequence $\mathcal{P} = u_{0_1}, u_{0_2}, \ldots, e_1, u_{1_1}, u_{1_2}, \ldots, e_2, \ldots, v$, whose terms are alternatively a finite number of vertices and an edge such that $H(e_i) = \{u_{i_1}, u_{i_2}, \ldots\}$ the head of the edge e_i, $T(e_i) = \{u_{i-1_1}, u_{i-1_2}, \ldots\}$ the tail of the edge e_i and $\forall i\ T(e_i) \subset W \cup \bigcup_{k>0}^{i-1} H(e_k)$ (i.e. every vertex in the sequence is adjacent to W joined with the already reached vertices) is called *hyperpath from* W *to* v.

And, finally, we have everything we need to define reachability.

Definition 3.8 (Reachability). Let $\mathcal{H} = (V, E)$ be a hypergraph, $v \in V$ a vertex, and $W \subset V$ a set of vertices. We say that v *is reachable from* W exists a hyperpath from W to v. We also say that any edge in the hyperpath is reachable from W.
Note: We consider that v is reachable from W if $v \in W$ as well for convenience.

3.2.1 Prior work

In classic graph theory, reachability is widely studied and several algorithms appeared to assess the reachability of a certain vertex from a given vertex (or vertex set). In the case of a regular digraph (or undirected, does not really matter on this regard), the adjacency matrix Definition 3.9 (and its powers) give all the information required to assess the node reachability.

Definition 3.9 (Adjacency matrix of a digraph). [BM08] The *adjacency matrix* of

a digraph [BM08] D with n vertices is the $n \times n$ matrix $\mathbf{A}_D = (a_{uv})$ where a_{uv} is the number of arcs in D with tail u and head v.

One could notice that it is clear that this definition will not work with directed hypergraphs since the tail and head cardinal might be greater than one, and such matrix cannot be constructed in general for hypergraphs.[2]

The following proposition could help to shape a simple algorithm to verify if a certain vertex is reachable from another one.

Proposition 3.10. Let $\mathbf{A}$ be the adjacency matrix of a digraph D. Then, element (i, j) of the matrix $\mathbf{A}^n$ gives the number of walks of length n from vertex i to vertex j.

Corollary 3.11. Let $\mathbf{A}' = \sum_{k=1}^{n} \mathbf{A}^k$ be the sum of all powers of the adjacency matrix. If and only if the coefficient (i, j) of $\mathbf{A}'$ is different than 0, the vertex j is reachable from i.

In order to replicate this behavior, we introduce the concept of *adjacency tensor*.

3.2.2 Adjacency tensor

The idea behind this adjacency tensor is to use Theorem 2.10 to identify every digraph with a tensor in a particular vector space. Prior work has been made already on this regard, such as [CLN15] and [PZ14]. It will be introduced first for digraphs, and then for hypergraphs.

Definition 3.12 (Adjacency tensor for a digraph). Let $\mathcal{G}$ be a finite digraph with n vertices. Let $E = \mathbb{R}^n$ with basis $(e_1, \ldots, e_n)$, and dual space E^* with basis $(e_1^*, \ldots, e_n^*)$. The tensor $t \in E^* \otimes_{\mathbb{R}} E$ given by

$$t = \sum_{i,j=1}^{n} a_{ij}\, e_i^* \otimes e_j \tag{21}$$

where a_{ij} is the number of edges with tail in vertex i and head in vertex j is called the *adjacency tensor of* $\mathcal{G}$.

Proposition 3.13. The adjacency tensor uniquely identifies a digraph.

Proof. The tensor t is a member of $E^* \otimes_{\mathbb{R}} E$. After Theorem 2.10, we can identify every tensor t with an endomorphism in E. It is trivial to check that the tensor Eq. (21) is mapped to the unique endomorphism whose matrix in the basis $(e_1, \ldots, e_n)$ is $A = (a_{ij})$. Therefore, the adjacency tensor exactly matches the adjacency matrix, which is already known to identify the digraph. □

Extending this idea to directed hypergraphs is possible, even if the tensor space will not be as straight-forward as previous one. We will give the definition just to B-hypergraphs, which are the kind of hypergraphs we will be using further in this article. Even though, it can be easily extended to arbitrary finite hypergraphs.

Definition 3.14 (Adjacency tensor). Let $\mathcal{H}$ be a finite B-hypergraph with n vertices and m edges. Let $E = \mathbb{R}^n$ with basis $(e_1, \ldots, e_n)$, and dual space E^* with basis $(e_1^*, \ldots, e_n^*)$. The tensor $\tau_l \in E^{*\otimes k} \otimes_{\mathbb{R}} E$ given by

$$\tau_l = a\, e_{i_1}^* \otimes \cdots \otimes e_{i_k}^* \otimes e_j \tag{22}$$

[2] It could be constructed in the particular case of directed hypergraphs in which every edge has only one tail.

with a being is the number of hyperedges with tail in vertices $i_1, \dots i_k$ and head in vertex j describes a combination of tails and head. The tensor $t \in (E^* \otimes_{\mathbb{R}} E) \oplus (E^{*\otimes 2} \otimes_{\mathbb{R}} E) \oplus \cdots \oplus (E^{*\otimes n} \otimes_{\mathbb{R}} E)$ given by

$$t = \sum_{l=1}^{m} \tau_l \tag{23}$$

is called the *adjacency tensor of* $\mathcal{H}$.

Similarly to the case of a digraph, the adjacency tensor uniquely identifies the hypergraph.

Since the number of vertices is finite, the number of combinations of tails and head is finite too, and therefore the adjacency tensor is a finite sum of elementary tensors, each of them representing an edge of the hypergraph (or multiple edges with the same head and tail). The elements of the dual basis e_i^* are $e_i^*(x_1, \dots, x_n) \mapsto x_i$.

As an example, the graph in Fig. 1 would be uniquely defined by the following adjacency tensor:

$$t = e_1^* \otimes e_2 + e_2^* \otimes e_3^* \otimes e_1 + e_1^* \otimes e_2^* \otimes e_3^* \otimes e_4 \tag{24}$$

Since there are not multiple edges with the same tail and head, $a = 1$ for every τ in the sum.

Finding an algorithm that fulfills the concept of reachability Definition 3.8 through the iteration of adjacency should be the natural next step. We are looking for a map that, given a vector of initial vertices maps to a vector of vertices that are reachable from the initial vertices. Then, a map $E \to E$ is expected. First of all, we formally define the *adjacency map*:

Definition 3.15 (Adjacency map). Let $\mathcal{H}$ be a finite B-hypergraph with n vertices and k edges and let $t \in (E^* \otimes_{\mathbb{R}} E) \oplus (E^{*\otimes 2} \otimes_{\mathbb{R}} E) \oplus \cdots \oplus (E^{*\otimes n} \otimes_{\mathbb{R}} E)$ be its adjacency tensor, and W a set of initial vertices. By definition, $t = \sum_{i=1}^{k} \tau_i$, where $\tau_i = a\, e_{i_1}^* \otimes \cdots \otimes e_{i_l}^* \otimes e_j$. Every $\tau_i \in (E^{*\otimes l} \otimes_{\mathbb{R}} E)$ defines a l-linear map $E^{\otimes l} \to E$, which we denote by φ_{τ_i}. We now consider the *diagonal* map $\delta_l \colon E \to E^{\otimes l}$ given by $(x_1, \dots, x_n) \mapsto ((x_1, \dots, x_n), \dots, (x_1, \dots, x_n))$.

Therefore, the map $\phi \colon E \to E$, where $x = (x_1, \dots, x_n)$ is defined by

$$\phi(x) = \sum_{i=1}^{k} (\varphi_{\tau_i} \circ \delta)(x) = \sum_{i=1}^{k} \varphi_{\tau_i}(\delta(x))\,^{3} \tag{25}$$

is called the *adjacency map of* $\mathcal{H}$. This map is not linear anymore, but it is clear that it is a component-wise polynomial map.

The adjacency map could be practically understood as: given a set of vertices, *which vertices are adjacent to the given set of vertices.* In the same example Fig. 1, given $x = (x_1, x_2, x_3, x_4)$ as set of vertices, we have

$$\begin{aligned} \phi(x) &= e_1^*(\delta_1(x)) \otimes e_2 + e_2^* \otimes e_3^*(\delta_2(x)) \otimes e_1 + e_1^* \otimes e_2^* \otimes e_3^*(\delta_3(x)) \otimes e_4 && (26)\\ &= x_1 \cdot e_2 + x_2 \cdot x_3 \cdot e_1 + x_1 \cdot x_2 \cdot x_3 \cdot e_4 && (27)\\ &= (x_2\, x_3, x_1, 0, x_1\, x_2\, x_3)\,^{4} && (28) \end{aligned}$$

[3] Note that δ is the appropriate δ_l for each τ_i, that might differ depending on the number of vertices in the edge's tail.

While the adjacency map is useful, extending it to model the reachability Definition 3.8 concept will be rather more useful. We define the following procedure:

Proposition 3.16. Let $\mathcal{H}$ be a finite B-hypergraph with n vertices and k edges and let ϕ be its adjacency map. Let $W^0 \in \mathbb{R}^n$ the set of initial vertices. Let $\Phi(x) = x + \phi(x)$. Then, we say that $W^l = \Phi(W^{l-1})$ are the vertices *reachable in l steps from W^0*. In order to find all reachable vertices from W^0 in any number of steps, the iteration can be performed until the following condition is fulfilled: $\exists z = (z_1, \ldots, z_n)$ that $\forall i \in \{1, \ldots n\}$ $\phi_i(W)(W_i\Phi_i(W)z_i - 1) = 0$, where W_i, $\Phi_i(W)$ and $\phi_i(W)$ stand for the coordinate i of W, $\Phi(W)$ and $\phi(W)$ respectively.[5]

Note that the stop condition is equivalent[6] to $W^l = W^{l-1}$. There are two factors that can make the condition be 0 for a certain i. If $\phi_i(W)$ is 0, the information underlying the statement is that the edges pointing to the vertex i are not adjacent to W (i.e. W has not all the vertices in the edge's tail for every edge adjacent to the vertex i). On the other factor, $W_i\Phi_i(W)z_i - 1$ will only be 0 if both W_i and $\Phi_i(W)$ are different than zero. Otherwise, $W_i\Phi_i(W)z_i - 1 = -1$ for every z_i. Taking $z_i = (W_i\Phi_i(W))^{-1}$ does the trick. The information underlying this statement is that the vertex i is both found in W and in $\Phi(W)$.
Then, joining both statements, the condition is fulfilled when, for every vertex i, i is not reachable by any adjacent edge to W or, in case it is adjacent, it is as well present in W. This is indeed equivalent to $W^l = W^{l-1}$.

We can see the procedure in pseudocode. In this case, OldW will stand for W^{l-1} and W is W_0 at the start, and we get two extra functions: *head* and *tail*, that return the index of the vertices in the tail or head of an edge given respectively. Note that, OldW $\neq$ W only stands for checks if values are different than 0, not the exact value.

Algorithm 1 Node reachability from an initial set of nodes

```
1: function ReachableNodes(W)
2:     OldW ← nil
3:     while OldW ≠ W do
4:         OldW ← W
5:         for edge in edges do
6:             h ← 1
7:             for vertex in tail(edge) do
8:                 h ← h * W[vertex]
9:             W[head(edge)] = h
       return W
```

The solutions of the polynomial equations found in the stop condition of Proposition 3.16 describe the possible sets of *ReachableNodes*. For example, in Fig. 1 if v_1 is reachable then v_2 is automatically reachable as well. Thus, a set of nodes having v_1 and not having v_2 is not a possible value of *ReachableNodes* function. We give the following definition:

[4] In this particular example there are no vertices that receive more than one edge. Therefore, no sums appear in the coefficients of the final vector. This will not be the case in general. I.e. the vector will be, in general, made of polynomials in $x_1, \ldots, x_n$.

[5] This will happen for sure in, at max, $n - 1$ steps. The limit situation will be having $|W^0| = 1$ and we append 1 vertex at every step. We can only add $n - 1$ different vertices until we have them all.

[6] Equivalence is understood here as $W_i^l = 0 \Leftrightarrow W_i^{l-1} = 0$. We are only interested on the reachable side of things, not really the calculated value.

Definition 3.17 (Compatible reachable set). Let $\mathcal{H}$ be a finite B-hypergraph with n vertices and let T be its reachability map. $v \in \mathbb{R}^n$ is a *compatible reachable set* if $T(v) = v$. Otherwise, we say it is *incompatible*.

Following the example Fig. 1, we have the following equations from the conditions, one for each edge:

$$\begin{cases} x_2x_3(x_1x_2x_3z_1 - 1) & = 0 \\ x_1(x_2x_3z_2 - 1) & = 0 \\ 0 & = 0 \\ x_1x_2x_3(x_4x_1x_2x_3z_4 - 1) & = 0 \end{cases} \tag{29}$$

Solving the equations, for z_i arbitrary, we find that only the following 9 reachable sets are compatible: $\{\}, \{v_1, v_2\}, \{v_1, v_2, v_4\}, \{v_2\}, \{v_2, v_4\}, \{v_3\}, \{v_3, v_4\}, \{v_4\}\{v_1, v_2, v_3, v_4\}$. From a total of $4! = 24$ possible sets, only 9 are compatible.

3.2.3 Other ideas on the reachability map

Tropical semirings [MS15] At this point, the reachability function T (understanding it as the iteration described in Proposition 3.16) does not focus on which is the final result in the coefficients of the vector $T(W)$ but rather if they are different than 0. A different approach, while maintaining the structure and calculations, could be proposed. Using $\mathbb{R}$ as the field to define the hypergraph, with the usual operations $+$ and $\cdot$ has the problem that crossing an edge n times will add n times the result of crossing that edge to the head vertex.

The tropical semiring is defined by extending the real numbers with one (or both) infinities with the particularity that the minimum (or maximum) and the addition replace the usual addition and multiplication operations respectively. Using the maximum tropical semiring would avoid the previously mentioned behavior. Then, the compatible reachable sets Definition 3.17 would correspond to the fixed points of the iteration in the reachability map. Even though, we proposed the hypergraphs theory over a field, and not a semiring, and therefore nothing is proved for them. It could be an interesting future case study.

Boolean semiring Following the idea of tropical semirings, the boolean one could be useful to assess just reachability. Allowing only two values 1 and 0 (or `true` and `false`) with the usual operations OR and AND for addition and multiplication could be helpful while implementing the algorithm if we are only interested to verify whether or not a certain vertex is reachable from a particular set of vertices.

4 User Account Access Graphs

4.1 Model description

First of all, we will define the basic concepts needed before presenting the formal model.

Definition 4.1. [Ham+19] A *user* is a person or automated program that interact with services, and have *accounts* in these services. An account can be accessed by providing *credentials*. A set of credentials *provides access* to an account if presenting these credentials is sufficient to access the account. Losing all credentials *locks* a user *out* of an account if at least one of these credentials is necessary to access the account.

This definition provides us with all information required to understand what we are trying to model.

User Account Access Graphs (UAAG) in [Ham+19] are defined as plain directed graphs, with the particularity that edges are colored to differentiate which vertices are needed at the same time to access a third vertex. A different approach is proposed here: using directed hypergraphs Definition 3.2.

We will model credentials and accounts as vertices, and edges will be drawn to show that certain credentials or accounts provide access to another credential or account. Then, we model the relation of providing access to a vertex v (a credential or account) as follows: we draw a hyperedge with head $H = \{v\}$, and as tail every vertex required *simultaneously* to access the account. We draw, then, different edges to represent each *alternative* access method. Having this in mind, we are ready to formally define a User Account Access Graph:

Definition 4.2 (User Account Access Graph). Let $\mathcal{G} = (V, E)$, $|V| < \infty$ be a B-hypergraph Definition 3.2. $\mathcal{H}$ is called a *User Account Access Graph* (or *UAAG*).

Defining UAAGs as directed B-hypergraphs instead of plain digraphs (with colors in the edges though) as in [Ham+19] simplifies its definition. Prior work on reachability [AL17], connectivity [TT09], analysis, or other relevant particularities could be applied on top of them.

An example, got from [Ham+19], will be introduced:

Example *Fig. 3 This is a webshop account* acc_{shop} *that can be accessed with password* pwd_{shop} *or recovered from the e-mail account* acc_{mail}*. This e-mail account requires two-factor authentication with password* pwd_{mail} *and a TOTP code. This code can be generated by an authentication app on a device. This device can be unlocked either using a fingerprint or a PIN.*

4.2 Account reachability

Considering that we are trying to model account access, it is natural to find a way to assess whether or not an account is accessible from a set of already accessible accounts or credentials. We give the definition of access set and minimal access set:

Definition 4.3 (Access set). Let $\mathcal{G} = (V, E)$ be a UAAG, $v \in V$ a vertex, and $W \subset V$ a set of vertices. We say that W *is an access set of* v if v is reachable Definition 3.8 from W.

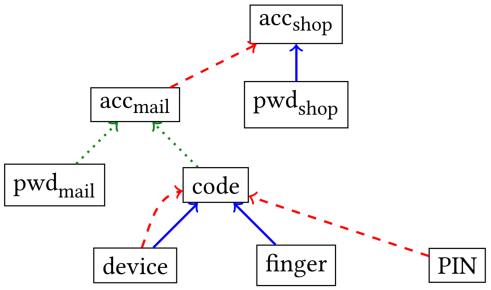


Figure 2: Account Access Graph example. Source: [Ham+19]

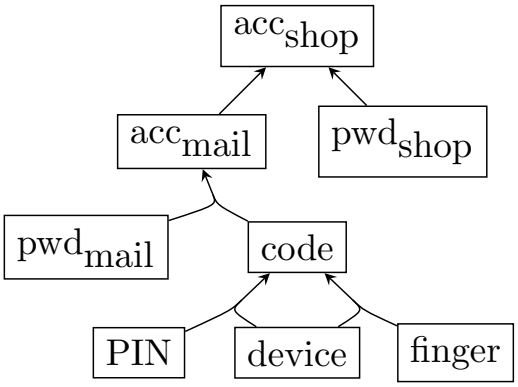


Figure 3: Account Access Graph example expressed in terms of hyperedges instead of colors.

Definition 4.4 (Minimal access set). Let $\mathcal{G} = (V, E)$ be a UAAG, $v \in V$ a vertex, and $W \subset V_0 \subset V$ an access set of v. We say that W *is a minimal access set of* v *from* V_0 if $\forall\, W' \subsetneq W$, W' is not an access set of v. If $V_0 = V$, we say that W *is a minimal access set of* v.

Note that a minimal access set of a vertex is not necessarily unique. For example, in Fig. 3 both $\{device, pin\}$ and $\{device, finger\}$ are minimal access sets for *code* vertex.

In order to find out if a set of vertices W is an access set of a certain vertex v we could use the reachability map of the hypergraph defined in Proposition 3.16, that gives all all the vertices reachable from the initial set W.

The notion of *reachability* could be extended to model not only the way of reaching a vertex, but also the likelihood of success of an attacker trying to break into an account. The following section will follow this path.

4.3 Scoring schemes

Until this point, a UAAG has been defined without paying attention to any possible weights for the nodes. By itself, an UAAG does not carry any operation for the vertices, but adding it could be useful. Therefore, we will propose the concept of a *scoring scheme* on top of an UAAG to provide it this functionality.

Based on the scoring scheme definition by [Ham+19], we give the following definition for a scoring scheme:

Definition 4.5 (Scoring scheme). Let $\mathcal{G} = (V, E)$ be a UAAG, an *scoring scheme over* $\mathcal{G}$ is a set $\mathcal{S}$ (called the *domain* of the scheme) equipped with:

1. $\preceq$, a partial order for elements in D.
2. $V_0 \subset V$, a set of initial vertices.
3. $\mathcal{S}_0 \in D$, the score of V_0.
4. $\epsilon \colon E \to D$, mapping every hyperedge to its score. It is called the *eval* function.
5. $\sigma \colon \mathcal{P}(\mathcal{S}) \to D$, where $\mathcal{P}(\mathcal{S})$ are the parts of the domain, mapping a set of hyperedges' scores to a final score for the head vertex. All hyperedges must have the same head. It is called the *combine* function.

Normally, we want to assign ϵ to a function that has the *and*-semantics for the elements in the tail of the edge. It makes sense, since access for every vertex in the tail is needed to cross an edge. An example could be an multiplication function. Analogously, we want to assign σ to a function that holds the *or*-semantics, since the ability to reach a vertex is given by the possibility of crossing any edge that leads towards it. For instance, it could be an addition function.

An example of scoring scheme could be the reachability function defined in Proposition 3.16. In that particular instance, the tensor product hold the *and*-semantics, and the addition hold the *or*-semantics. In this case, $\mathcal{S} = \mathbb{R}$ is equipped with the usual $\leq$, ϵ is the addition and σ is the tensor product.

An important point to highlight is that scoring schemes do not necessary need to be complete, in the sense that not all vertices are reachable from V_0. An attacker (or, alternatively, the user trying to recover access to an account) might not have access to every authentication or account, and the difficulty to get access could be hard to assess. An incomplete scoring scheme could be proposed then.

We give a definition to differentiate both:

Definition 4.6 (Complete scoring scheme). Let $\mathcal{G} = (V, E)$ be a UAAG and $\mathcal{S}$ be a scoring scheme over it. We say that $\mathcal{S}$ is *complete* if $\forall\, v \in V$, V_0 is an access set of v. A scoring scheme that does not meet this requirement is called *incomplete*.

We could extend ϵ and σ functions to an aggregate function that, given any vertex, returns the final score for that vertex.

Definition 4.7. Let $\mathcal{G} = (V, E)$ be a UAAG and $\mathcal{S}$ be a scoring scheme over it. Let $V' \subset V$ and $E' \subset E$ the set of all vertices and all sets reachable from V_0. We define:

$$\delta \colon V' \longrightarrow D, \quad v \mapsto \sigma\left(\{\epsilon(e) : e \in E', H(e) = \{v\}\}\right) \tag{30}$$

Based on this ideas, we will define the soundness of a scoring scheme:

Definition 4.8 (Soundness). Let $\mathcal{G} = (V, E)$ be a UAAG and $\mathcal{S}$ be a scoring scheme over it. We say that $\mathcal{S}$ is sound if, for any $a, b \in V$ that are reachable from V_0, a, b meet

$$\begin{aligned} \forall\, V_a \text{ minimal access set of } a, V_a \text{ access set of } b \\ \implies \delta(b) \preceq \delta(a) \end{aligned} \tag{31}$$

I.e. if any credentials valid to access a are valid as well to access b, the score of a is, at least, the score of b.

Depending on what we are assessing, the soundness attribute might have different meanings. For example, while assessing security the score describes the security of a vertex. I.e. if the score is higher, then the security is higher (or the risk of attack is lower) according to $\preceq$. On the other hand, we might assess lockout risk (we will introduce it in Definition 4.10), where higher score means higher risk. The soundness attribute has to be true in either case.

In general, we expect that the score found for a vertex will be opposed to reachability, in the sense that higher reachability will have lower score, and vice versa. The limit example would be an account that is not accessible in any way, it should have the maximum score (both for security and lockout risk); and an account that is accessible without any credentials will have the minimum score.

4.3.1 An attacker based scoring scheme

One of the best ways to measure the security of any account is, indeed, figure out which attackers could theoretically compromise the account. [Ham+19] introduced an example of this scoring scheme. Formally, an attacker based scoring scheme will be a scoring scheme based on n attribute sets tuple $(A_1, \ldots, A_n)$, all of them fully ordered.

Definition 4.9 (Attribute based scoring scheme). Let $\mathcal{G} = (V, E)$ be a UAAG, $\{(a_1, \ldots, a_n) : a_i \in A_i\}$ the set of all combinations of n-tuples of with elements in $(A_1, \ldots, A_n)$. A_i sets are fully ordered. $\mathcal{S} = \mathcal{P}(\{(a_1, \ldots, a_n) : a_i \in A_i\})$ the parts of them, equipped with

1. $\leq$, given by the following definition: for any $S_1, S_2 \in \mathcal{S}$, $S_1 \leq S_2 \Leftrightarrow \forall\, u = (u_1, \ldots, u_n) \in S_2\ \exists\, t = (t_1, \ldots t_n) \in S_1 \ : \ t_1 \leq s_1, \ldots, t_n \leq s_n$.[7]
2. ϵ, given by the component-wise maximum for each attribute set given. Note that it the function collapses all sets in just one attribute set.
3. σ, given by the minimal sets compared as a whole. The function could return a set of multiple attribute sets, and will only collapse them with the minimum function if they are fully comparable with $\leq$.

For example, one could define a tuple with just two attributes: location and skill. An attacker might be *local* or *remote* and have a different level of skill. For example, a piece of paper with a password written in it could be attacked by a *local* attacker with no level of skill at all (it is just taking the paper) but it is impossible to be attacked by a *remote* attacker. Therefore, we define $\mathcal{S} = (\{\text{remote}, \text{local}\}, \{\text{none}, \text{some}, \text{high}\})$ with remote $\leq$ local and none $\leq$ some $\leq$ high as an attacker-based scoring scheme.

For instance, in the example Fig. 3 we could take { device, finger, PIN, pwd_{mail}, pwd_{shop} } as initial vertices and (remote, some) to passwords, (local, none) for the device, (local, some) for the PIN and (local, high) for the finger. Then, we find that the code has (local, some) as score, after combining (local, some) and (local, high). Going up until the account, we find that the account is only accessible with (local, some). That account is probably safe enough, since an attacker has to have local access to our phone and be experienced.

4.4 Recoverability

Until now, we only focused on the security of an account, but recoverability needs to be analyzed as well. On a limit case, the maximal theoretical security (thought as possibility of access by an attacker) is reached if the account is not reachable using any credentials. Even though, the user would be locked out as well from the account. Then, a good balance between security and recoverability is desirable.

The definition of minimal access set Definition 4.4 could be thought as well as minimal lockout sets. Instead of thinking that we need all vertices to reach the particular account, we only need to lose access to anyone to get locked out of the account. This idea makes the study of recoverability completely analogous to the one of security, just by defining a different scoring scheme: less secure becomes less lockout risk.

Another important situation that could happen is the inherent lockout risk. For example, it is what would happen if there is an internet outage where the servers of the account we

[7] It is trivial to verify that this scoring scheme is sound Definition 4.8.

try to access are located. This was already described in [Ham+19], but given the kind of study we will try to do further on in this paper, we will not take it into account.

While assessing security, we mainly considered that, if multiple ways were suitable to access an account or credential, we consider that the security of that account or credential was the lower of them all. In the recoverability this is not exactly the same. For example, having two different ways to recover an account increases the recoverability than having just one, regardless of the nature of them. This is, of course, in general, there might be cases (such as one of this ways is not accessible, and therefore it does not increase the recoverability).

4.4.1 A lockout scoring scheme

Taking the previous ideas into consideration, we propose the following lockout scoring scheme fulfilling the definition given in Definition 4.5:

Definition 4.10 (Probablity based lockout scoring scheme). Let $\mathcal{G} = (V, E)$ be a UAAG, $\mathcal{S} \coloneqq [0, 1] \subset \mathbb{R}$, equipped with

1. $\leq$ fully ordering $\mathbb{R}$
2. ϵ, given by the maximum for each tail score of the edge.
3. σ, given by the multiplication of all scores from the different edges.

It is trivial to verify that this scoring scheme is sound according to Definition 4.8.

The score is understood as the probability of getting locked out (i.e. losing access to a credential or account). 0 means that it is impossible to get locked out of the account and 1 means that there is certainty that the user will get locked out.

Despite the fact that this model is very simple, it could be really difficult to assess the lockout probability for a credential or account. We will not focus on this topic in this paper.

This simple lockout scoring scheme has a noticeable flaw that needs to be taken into account when applying it. We propose an illustrative example in Fig. 4 that highlights it.

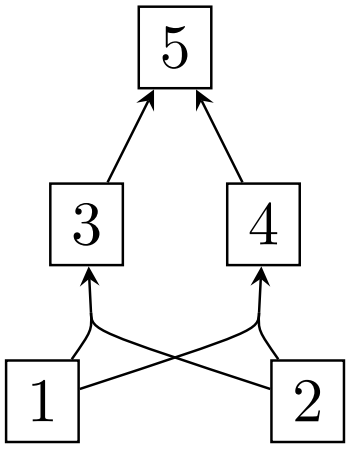


Figure 4: UAAG showing Definition 4.10 flaws.

Even if this setup might seem artificial at first sight, inefficiencies in a user setup like this one might happen (even if they are masked behind several layers). In this case, 1 and 2

simultaneously give access to 3 and 4, and either of this two could be used to access to 5. If we give a score of 0.5 for both 1 and 2 (actually, it reproduces with any scoring different than 1 for both of them), 3 and 4 have a score of 0.5 as well. Then 5 would get a score of 0.25. If we look closely to it, 5 has exactly the same level of lockout risk as 3 and 4, and this is not reflected properly in the lockout score. We need to be thoughtful when using this lockout scoring scheme to avoid falling in this kind of behaviors.

4.4.2 Backdoor

The concept of *backdoor* was introduced in [Ham+19]. An account has a backdoor if the security score accessing it through the recovery method is lower than accessing it through the regular access method. The regular access method to an account would be the one that the user usually uses to access.

For example, in Fig. 3 the regular access method would be the pwd_{shop} and accessing through acc_{mail} would be the recovery method. Using the scoring scheme described in Section 4.3.1 with the example given for Fig. 3 we conclude that acc_{shop} in has no backdoors.

5 Case study: Polkadot

A blockchain [Wik24a] is a distributed ledger with growing lists of records (also known as blocks) that are securely linked together via cryptographic hashes. Typically, this ledger is managed by a peer-to-peer (also known as P2P [DPH14]) computer network (whose members are also known as nodes) that adhere to a consensus algorithm protocol.

In particular, Polkadot [Foub] is the first layer-0 blockchain that provides shared security and secure interoperability to layer-1 blockchains. Those layer-1 blockchain attached to Polkadot are also called *parachains*.

Polkadot uses Nominated Proof-of-Stake (known as NPoS) as its mechanism for selecting the validator set (the chosen nodes that will produce new blocks). It uses what is known as *hybrid consensus*: it splits the block production mechanism (BABE) from the finality gadget (GRANDPA [SK20]). Polkadot stands out among other blockchain projects due to its governance mechanism.

Polkadot uses a sophisticated governance mechanism called *OpenGov* allowing it to evolve gracefully overtime at the ultimate behest of its assembled stakeholders. The goal is to ensure that most of the stake can always command the network. Even though, some times this governance system might behave unexpectedly to other builders coming from other blockchains.
There are different *tracks* that accept proposals that stakeholders could vote on. Every track has their own rules defined in the blockchain code, with different levels of required agreement to execute certain actions. Theoretically, any arbitrary function could be executed on behalf of any account if a vast majority of the stakeholders agrees on it through the `Root` origin [Fouc]. Compared to *Bitcoin*, among the majority of blockchains, this is unexpected since the only way of executing something on behalf of any account is by having access to the account itself cryptographically.

The fact that Polkadot is a layer-0 blockchain coordinating many parachains makes Polkadot ecosystem to be remarkably heterogeneous, especially while discussing about functionality, where every parachain differ substantially. Nevertheless, when looking closely to security, cryptography, and account management, the whole ecosystem uses the same principles and foundations, and this extends as well to wallets and keyring management tools.

Therefore, our study could apply to any chain based on the Polkadot SDK and FRAME framework [Tecb], and we will particularly focus on Polkadot and Kusama runtimes [Fel] developed by the Polkadot Technical Fellowship.

On this study, we will only focus on Polkadot accounts whose users hold the cryptographic information (specifically, the private key in asymmetric cryptography) by themselves. These accounts are known as *non-custodial*, since no one besides the user holds any sensitive information. An interesting study on a login setup for the cryptocurrency exchange *Binance* is found in [Ham+19]. Binance is built on a *custodial* structure, meaning that the exchange holds the tokens on behalf of the user and the user cannot access them permissionlessly.

5.1 Wallet accounts

When we talk about wallet accounts we mean that are accounts on the network that represent the public key of a public/private keypair [Sal96]. Not every account in Polkadot

is a wallet account, since we have multi-signature accounts or pure proxies (among others) that lack of a public key counterpart. Then, we give a formal definition:

Definition 5.1 (Wallet account). A *wallet account* is a valid private key [Sal96] derived using a supported algorithm[8] in the blockchain.

Definition 5.2 (Wallet). A *wallet* is a container of wallet accounts with a feature allowing to create valid digital signatures for every one of this private keys.

In Polkadot every public key derived from a supported algorithm[9] is an account, but not all accounts have a public key counterpart. Therefore, we call *wallet accounts* to those accounts that represent a public key in the blockchain.

In Polkadot there are mainly two kinds of wallets: browser extension based and hardware based. Often, hardware wallets are used with a companion browser extension that handles the communication part, but the device stores the private key and therefore the wallet is the device, not the extension.

As we previously mentioned, we need to consider as well that Polkadot is a blockchain with a strong on-chain governance. In theory, anything could be done on behalf of any account through the `Root` origin [Fouc]. Therefore, there will always be an additional access method for every account that will be on-chain governance.[10]

An important thing to consider as well is that this paper only focuses on the possibility that an attacker could get access to a private key owned by a user. There are other kinds of attacks that might affect the security of users interacting with a blockchain, such as man-in-the-middle (also known as MITM) attacks, or even social engineering attacks, such as address poisoning attacks. The scope of this study, though, is the attacker gains complete and unrestricted access to the blockchain account, normally through its private key.

5.1.1 Browser extension wallets

Currently, browser extension-based wallets are the most commonly used type of wallets in the ecosystem. They offer the possibility to directly sign transactions crafted by a decentralized application in the browser, being really flexible. The private keys are stored in the browser storage (normally encrypted using a password), which makes them less secure than other key storing mechanisms.

We assessed three different wallets:

- Polkadot-JS Extension [GR]
- Talisman Extension [Tal]
- SubWallet Extension [Kon]

While having some differences among them, especially on the UI side and some additional features besides key management, we noticed that the security setup of every wallet assessed is almost identical. Therefore, we will treat them as just one case study.

[9] Polkadot-based chains accept mainly accounts using SR25519 or ED25519 signatures. [Foua]

[10] While accessing an account through the `Root` track is theoretically possible, in practice Polkadot governance generally rejects to get involved in recovering access to individual accounts, or force transfer funds from a compromised account, etc. This could change at any point if enough voting power agrees on it, and it is something that should be considered as a feasible option. Other Polkadot-based chains used to accept similar proposals in the past, but at least in Polkadot it does not happen.

The process of creating an account on any these wallets is:

1. The wallet asks for a password to encrypt the subsequent generated secrets. It is, at least, 8 chars long.[11]
2. The wallet randomly generates a private key using SR25519 algorithm.[12]
3. The wallet shows on screen a mnemonic phrase of 12 or 24 words, depending on the wallet (some of them offer the option to choose).[13] The wallet recommends to back it up in a secure and non-digital way. We will consider that is backed up offline in a piece of paper.
4. The wallet stores in the browser storage an encrypted version (using the password generated in Item 1) of the seed.

Therefore, in order to decode the private key and gain access to the account using the normal login method the user (or attacker) needs to have access to the computer (physical or remote) and know the password to decode the secret. There is a backup method as well using the written mnemonic phrase and importing it back in any wallet. Besides the own methods for this particular kind of wallet, the on-chain governance is an access method as well.

We will need, as well, to make some assumptions. The setup for every user might be really different regarding security standards, OS, etc. We will then consider:

- The particular account is only installed in one browser at the same time. Theoretically, one could import the account in an arbitrary number of devices. We will not consider this situation.
- We consider as well that the computer has an OS password different than the wallet password. This is indeed the most
- We consider that the storage is encrypted using the OS password, and that a physical attack to the storage in order to compromise the information cannot be done.
- The wallet password is not stored in the computer.

With all the information gathered, the resultant UAAG is shown in Fig. 5.

The three minimal access sets for this graph (with the initial set being the leaves of the graph) are

$$\left\{\text{pwd}_{\text{computer}}, \text{computer}, \text{pwd}_{\text{wallet}}\right\}, \ \{\text{on-chain governance}\}, \ \{\text{mnemonic}\} \tag{32}$$

As explained in Footnote 10, getting access to an account (or its funds) through on-chain governance is really difficult and should not be considered a real option. We will consider it to be impossible; therefore, only two minimal sets apply.

Using the attacker-based scoring scheme described in Section 4.3.1, with ((remote, local), (none, some, high)) as attributes for the attacker, we assign the following scores to the leaves of the graph:

[11] Some wallets force the user to use at least a number, a capital letter, and a symbol. We will not consider any difference regarding this restriction.

[12] Polkadot-based chains accept mainly accounts using SR25519 or ED25519 signatures [Foua] (ECDSA signatures are supported as well), but the browser-based wallets generally prefer to use SR25519.

[13] While not being equivalently secure, we will not make a difference between them and we will consider for this case study that is impossible to brute-force attack a randomly generated keypair with, at least, 128 bits of entropy.

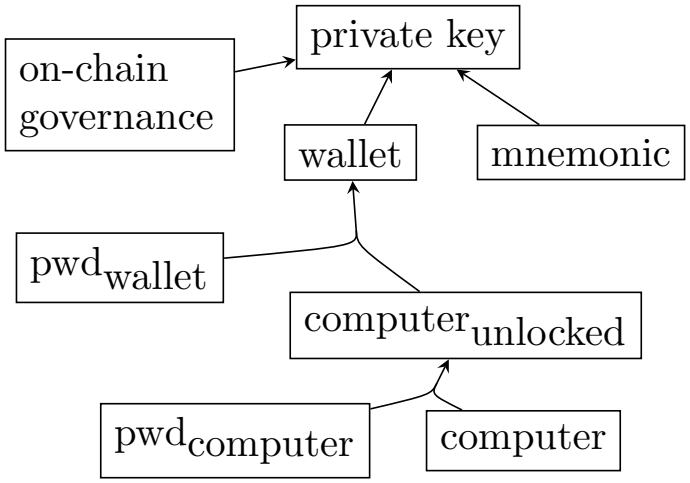


Figure 5: UAAG for a browser extension wallet.

- Both passwords get a (remote, some). In general, they can be tried as many times as necessary by the attacker and could be prepared remotely and, in general, highly resilient passwords are not necessarily enforced.
- Computer get ((local, none), (remote, high)). Getting access to the computer while being locally is trivial, but gathering access remotely is, in general, very difficult with modern OSs.
- The mnemonic gets (local, none). It is trivial to get access locally and it is impossible to get access to a piece of paper otherwise.

Then, by calculating the security of *private key* we get that is accessible by (local, none) through *mnemonic* and (remote, high) through *wallet*. While it was collapsed due to a lower level of security, it is also accessible by (local, some) through *wallet*.

Theoretically, the recovery method of this account is the mnemonic seed written in paper, and the normal way of accessing it is through the wallet. Thus, we conclude that the mnemonic paper constitutes a backdoor to the account since its security for a local attacker is lower. If we consider remote attackers, it does not constitute a backdoor.

While this conclusion is theoretically true given our calculations, normally the mnemonic seed is hidden in a place that only the user knows. Therefore, the likelihood of finding it for an attacker is noticeably lower than finding the computer, that might be way more accessible. This depends on the setup of the particular user.

In order to increase the security, the following recommendations apply:

1. The wallets should enforce a long, random, and difficult to guess password. A minimum of 16 characters with mandatory letters (both capital and not), numbers and symbols would be better than the current situation. Then, on our model the score could improve to (remote, high) for the wallet password.
2. The wallets could recommend the user to hide the mnemonic paper in a really unexpected place or even split the mnemonic in different places. The security would improve, then, to (local, some) because the attacker would need some knowledge to find the mnemonic.

In terms of recoverability, this setup is really weak. We assumed the mnemonic seed phrase

is written in a piece of paper and only copied once. If the user loses access to the place where it is stored, or it becomes inaccessible due to other reasons (e.g. a fire at home where might be stored) the user would lose any possibility of recovering the account. On the other side, losing access to the computer (either physically or due to an attack[14]) or any of the passwords would lock out the user.

Considering the lockout scoring scheme proposed in Definition 4.10, we assign the following scores to the leaves:

- Both passwords get a 0.5 score. It is generally easy to lose access to a password.
- The computer gets a 0.8 score. Having it stolen, broken or lost is reasonably common.
- The mnemonic gets a 0.3 score. Generally the mnemonic is stored in a safe place with low risk of loss or degradation (i.e. at home).

Then, the final score for the *private key* is 0.24. It is a reasonably high number, which means that the lockout risk is not negligible.

Clearly, the main issue of this setup is the low level of recoverability that it holds. It should be improved in order to avoid users being locked out. Having all of this into account, the following recommendations apply:

1. The user could copy several times the mnemonic key and store it in different places. This reduces drastically the lockout risk. For example, for two copies the final score would be ≈ 0.07, for three copies ≈ 0.02, etc. This would increase lightly the attack surface (since there are now more mnemonic copies to protect) but in general it should be worth.
2. The user could backup the data stored in the wallet encrypted with the wallet password[15] (or a different one) in other places than the main computer. This would reduce the lockout risk (since we are adding new ways to access the private key) while increasing the attack surface. Depending on the result, it might be worth.

5.1.2 Hardware wallet: Ledger

Ledger [Leda] and Trezor [Sat] devices are the most widely used multi-blockchain hardware wallets. They allow to store private keys in a device that is air-gapped (i.e. does not interact at all with Internet). Despite the fact that Polkadot ecosystem is really popular, Trezor does not support it and we will only focus on Ledger.

Compared to browser extension wallets, private keys are stored in a secure device that has no connection to Internet at all, and therefore cannot be accessed remotely by an attacker. Due to this fact, its security is expected to be higher than in browser extension wallets. Besides the remote protection, the Ledger Nano S+ [Ledb] (which is the device of Ledger used for this study) stores the secrets necessary to get access to the private key in a secure element certified with EAL6+ [Wik24b], making it secure as well from a physical point of view.

A Ledger Nano S+ with firmware version `v1.1.2` was used for this testing. The setup process to create an account is:

[15] All the wallets studied offer this possibility.

[15] Ransomware [Bea+21] are increasing these days and the risk of being attack by them is not negligible. A ransomware attack would leave the filesystem of the computer completely inaccessible.

1. The device asks the user if it should generate a new seed or load another one that the user might have as a mnemonic. We proceed with generating a new seed.
2. The device asks the user to choose a PIN from 4 to 8 digits (numbers from 0 to 9). We will consider that the user chooses a 4 digit PIN for simplicity.
3. The device randomly generates a seed (that will be then used to generate private keys for every blockchain) and prompts the 24 words mnemonic. It recommends the user to write it in a piece of paper and never type it in any electronic device or show the mnemonic to a camera.
4. The device stores in its secure storage the seed.

In order to get access to the private keys stored in the device the user (or attacker) has to have access to the device (only physically, air-gapped devices cannot be accessed otherwise) and know the PIN. There is, of course, a backup method by loading the written mnemonic in another device. Besides that, the on-chain governance option is present as well like all other accounts.

The resultant UAAG for a Ledger device is shown in Fig. 6.

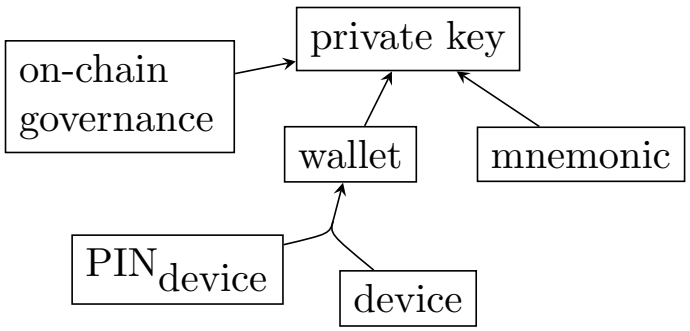


Figure 6: UAAG for a Ledger device.

The three minimal access sets for this graph (with the initial set being the leaves of the graph) are

$$\{\text{PIN}_{\text{ledger}}, \text{ledger}\}\,,\ \{\text{on-chain governance}\}\,,\ \{\text{mnemonic}\} \tag{33}$$

As explained in Footnote 10, getting access to an account (or its funds) through on-chain governance is really difficult and should not be considered a real option. We will consider it to be impossible; therefore, only two minimal sets apply.

In this case, even if the setup is rather simple, it gives a higher level of security without compromising the recoverability compared to browser extension wallets.

Using the same attacker-based scoring scheme, with ((remote,local), (none, some, high)) as attributes for the attacker, we give the following scores to the leaves of the graph (the initial set):

- PIN gets (remote, high). While it can be prepared remotely, it can only be tried three times before the device wipes itself. Then, a brute-force cannot be done. With a simple probability calculation, the chance of guessing a random 4 digit PIN in three attempts is $0,03\%$. That means that an attacker would only guess it one time every more than 3000 attempts. This is a clear advantage against browser extension wallets, whose passwords can be brute-force attacked.

- Device gets (local, none). Getting physical access to the device locally is trivial.
- Mnemonic gets (local, none). Getting physical access to the piece of paper locally is trivial.

Then, the *private key* is accessible by (local, none) through the mnemonic. Through *wallet*, it is accessible by (local, high). This setup with Ledger device improves substantially the extension one since it is not accessible at all for a remote attacker.

Once again, we conclude that the mnemonic is a backdoor to the account since its security against a local attacker is lower than the security of the regular access method. While being theoretically true, it might be difficult to find the mnemonic since users usually hide it.

In order to increase the security of the setup, the following recommendations apply:

1. Choosing a longer PIN might be a great way to further improve the security of the access. For every digit we add, the probability guess goes down by an order of magnitude (for a 5 digit PIN would be 0.003%). Besides that, enforcing a random PIN (proposing it to the user as well, for example) would reduce drastically the likelihood of non-random guesses (e.g. birthday of the user, year of first son, etc). There is no room for improvement on this regard in our model, since we already gave the maximum score for it.
2. Like in the browser extension case, hiding the mnemonic paper in a difficult to find place, or even split it would increase the security and reduce the likelihood of finding it. In our model this improvement would likely mean an increase of security to (local, some).

Compared to the browser extension, it is clear that the security of the setup is way higher. Besides the mnemonic situation (which is exactly the same situation for both cases) a user that is concerned about its security would probably use a Ledger.

An interesting attack that is not modeled in our case is the supply chain attack. A potential attacker might attack the supply chain of the device to replace clean and safe devices by malicious devices that look exactly the same as a legit one. Ledger develops a desktop program as well called *Ledger Live*. Inside it, a genuine challenge can be sent to the device and, if it happens to be malicious, warn the user. Even though, if the device is malicious and is connected by USB to the computer other security concerns may arise. We will not take into account this possibility and consider that the device is genuine.

In terms of recoverability, the setup is almost identical to the browser extension one. Giving the same scores as before:

- PIN gets a 0.5 score.
- The device gets a 0.8 score.
- The mnemonic gets a 0.3 score.

Then, the result is a lockout risk score of 0.24. It is clearly the main flaw of the setup, taking into account that the security was reasonably high. Then, in order to improve the recoverability, a user could implement any of the following recommendations:

1. Like in the browser extension case, the user could copy several times the mnemonic key and store it in different places. The reduction in lockout risk on our model gets

approximately divided by three for every extra copy.

2. The user could back up the PIN in a password manager. Since the device needs to be accessed locally, this would not reduce the kind of attacker that could get access to the account in our model. Despite this fact, it would indeed increase the attack surface. It might be worth in some cases, but not in all of them.

5.1.3 Hardware wallet: Polkadot Vault

Polkadot Vault (formerly Parity Signer) [Teca] is a mobile app (available for both iOS and Android) that turns an unused mobile device into an air-gapped device to store the keys. This app is downloaded and installed through the official stores Google Play and Apple Store respectively. The communication process between the app and the network is through QR codes, scanned using the camera of the device to receive data and shown in the screen for another device to scan them to send data out. It only works on Polkadot-based networks (unlike Ledger, that works in many different protocols).

It does not use a secure element with certification to store the sensitive data[16], but it forces the phone to have an access PIN, fingerprint, or any other method used by the particular device and encrypts the sensitive information with it. We will not consider the possibility of a physical attack to the device without knowing the password since it is out of scope for this study.

We used an Android phone and Polkadot Vault `v6.3.1` installed through Google Play. The process of creating an account on this device is as follows:

1. The app forces the user to air-gap the device. The user needs to:
   - Enable flight mode.
   - Disable WiFi.
   - Disable Bluetooth.
   - Disable ADB.
   - Disconnect any USB cable from the device.
2. The app randomly generates a seed and shows the 24 words mnemonic on screen. The user should write it down in a piece of paper.

Considering that the access method of the device is a numeric PIN, the graph is exactly the same as in the case of Ledger Fig. 6 and we will assign in general the same scores. Even though, the following differences apply and some scores will change as follows:

1. The number of PIN wrong attempts before wiping the device out varies from device to device. In old Android phones, it used to be really high, and there were other recovery methods (such as mail-based recovery codes). In recent versions of Android this matter changed and nowadays the lock screen PIN cannot be recovered, and a maximum number of 10 attempts is allowed. 10 attempts give a chance of 0.1% (or one chance every 1000 tries) to guess it for a 4 digit PIN. Considering that there was

[16] This is not true for a small minority of devices that allow storing information in secure elements found in the processors of the devices. It is not a general situation and, therefore, we will not consider it. Our study was performed in a fairly modern Android phone (from late 2019) and there was no hardware protection.

a possibility of recovering the PIN in several ways, and that the chance of guessing the PIN is 3 times higher, we will assign a (remote, some) score to the PIN.

2. Besides that, mobile devices have wireless connection (such as mobile data, WiFi, etc) capabilities. Even when the app forces the user to enable flight mode, disable WiFi, Bluetooth and ADB in order to use it, there might be a possibility for a remote attacker of getting access to the device. We will not consider this possibility in this case study.

Then, the final score in terms of security is (local, none) through the mnemonic seed phrase. Through the app it is accessible for (local, some). The lockout risk score is the same 0.24. The same recommendations given for Ledger apply as well to Polkadot Vault.

5.1.4 General conclusions

After studying the three main wallets we expose the following findings:

1. In every wallet studied, the mnemonic backup constitutes a backdoor to the account for a local attacker since it is stored in plain text. The main effort of users should be to effectively protect this mnemonic with any technique available. Wallets could warn the user to pay extraordinary attention and protect themselves from this fact.

2. Besides the already mentioned mnemonic, we gave options to increase security for every kind of wallet. This is particularly important for browser extension wallets, which are the most insecure among all of them in our model.

3. Due to the fact that the setup is really simple in terms of number of credentials and accounts, we did not find any redundancy on access methods (an example of redundancy is found in Fig. 4).

4. The recoverability should be the main concern for every setup studied. Without any further improvement, the risk of lockout is really high. Implementing one of the ideas proposed would decrease substantially the lockout risk.

5.2 Proxy accounts

In Polkadot, a proxy account [Foud] is an account that is allowed to act on behalf of a third account called *stash* for certain purposes. Some examples of these purposes would be `Any`, allowing to execute whichever transaction; `Non-transfer`, allowing to initiate whichever transaction besides transferring funds and `Governance`, allowing to make transactions related to the previously mentioned *OpenGov*; among others.

These proxy accounts could add an extra layer of security. For example, the stash account could be a Ledger hardware wallet, holding a higher level of security, and a proxy for only `Governance` purposes could be created in a browser extension wallet account. Despite the fact that it is definitely more insecure to use a browser extension wallet, it is more usable and might be enough for certain users for non-destructive actions.

This study only focuses on an attacker gaining complete and unrestricted to a certain account, and the only proxy that fulfills it is the

Even though, the setup might differ from account to account and it is interesting to do a proper case study for any user concerned about the security. For an `Any` proxy, the setup would directly be that two different wallets give access to the same private key. Thus,

not only the wallet of the user gives access to the account but also the wallet holding the information of the proxy.

5.3 Multisignature accounts

There is a kind of account in almost every Polkadot-based network called *multisignature* (or *multisig*). This kind of account is deterministically generated from a set of other accounts (wallets, or any other account) and a threshold k. The account is accessible if at least k accounts from the set of accounts agrees on it.[17] In general, regular wallet accounts are used to set up multisig accounts.

Even if a wide variety of different setups might apply, we propose an example for a 2 out of 3 (3 accounts, threshold 2) multisig in Fig. 7.

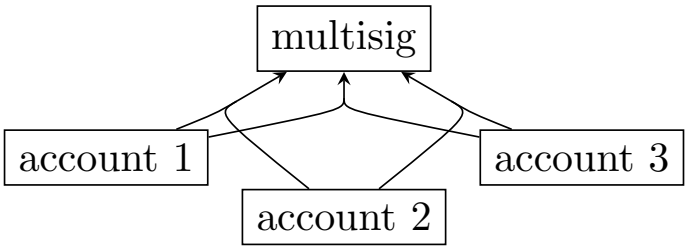


Figure 7: UAAG for a 2 out of 3 multisig.[18]

Any pair of two accounts would be allowed to access the account. Therefore, the study of these three accounts is really important to make sure all of them follow a great level of security principles. In terms of recoverability, it is clear that the recoverability increases due to the different pairs of accounts that, together, can access the multisig account. Even though, the flaw of our simple lockout shown in Fig. 4 scoring scheme is partially reproduced in this particular example.

5.3.1 Recovery pallet

In Polkadot-like networks, features are organized by *pallets* (every feature has its own pallet or group of pallets; e.g. `balances` pallet for transferring and receiving funds). It exist a pallet called *recovery* that allows an account to set any number of trustees that could access the account if a threshold k agree. Among all chains we verified, this feature is only available on *Kusama* network (a canary network of Polkadot) and it is not getting much traction since its inception.

The behavior in terms of security and recoverability would be similar to creating a multisig account with all the trustees, and k as a threshold. Then, creating a `Any` proxy to that account would achieve a similar result, even if they differ some details (such as time to recover, the way of submitting the request of recovery, etc). An example of multisig is introduced in Fig. 7 and could apply as well to a *recovery* setup.

[17] It is really common as well to create multisig accounts using what is called a *pure proxy* [Foue]. Nevertheless, for our case study we will consider that there is no difference between pure proxy based multisigs and native ones.

[18] We are not taking into account the on-chain governance method of accessing the account as explained in Footnote 10.

6 Conclusion

We presented User Account Access Graphs, a formalism constructed upon directed hypergraphs, drawing inspiration from [Ham+19]. This formalism enables detailed analysis of account setup, in terms of both security and recoverability. Its design, based on a well-examined mathematical object like directed hypergraphs, allows for straightforward extensions.

In our case study involving the blockchain network Polkadot, we assessed the prevailing security measures within the ecosystem. We identified subtle flaws in the wallet setups under examination, and gave some enhancements which could be implemented by users and wallet builders to increase the security of the Polkadot ecosystem further. Besides wallets, we also studied other native objects in Polkadot, such as multisignature accounts.

The presented formalism paves the way for extensive examination of account graphs in other blockchain networks, or more broadly, any account setups beyond the blockchain context.

References

[AL17] Giorgio Ausiello and Luigi Laura. "Directed hypergraphs: Introduction and fundamental algorithms—A survey." In: *Theoretical Computer Science* 658 (2017). Horn formulas, directed hypergraphs, lattices and closure systems: related formalism and application, pp. 293–306. ISSN: 0304-3975. DOI: ↗. URL: ↗.

[Bea+21] Craig Beaman et al. "Ransomware: Recent advances, analysis, challenges and future research directions." In: *Computers & Security* 111 (2021). ISSN: 0167-4048. DOI: ↗. URL: ↗.

[Ber73] Claude Berge. *Graphs and Hypergraphs*. North-Holland Publishing Company, 1973. ISBN: 978-044-410-3-9-9-4. URL: ↗.

[BM08] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer London, 2008. ISBN: 978-184-628-9-7-0-5. DOI: ↗. URL: ↗.

[Con] Keith Conrad. *Tensor products*. [Online; accessed 12-May-2024]. URL: ↗.

[CLN15] Lu-Bin Cui, Wen Li, and Michael K. Ng. "Primitive tensors and directed hypergraphs." In: *Linear Algebra and its Applications* 471 (2015), pp. 96–108. ISSN: 0024-3795. DOI: ↗. URL: ↗.

[DPH14] Joan Antoni Donet Donet, Cristina Pérez-Solà, and Jordi Herrera-Joancomartí. "The Bitcoin P2P Network." In: *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, 2014, pp. 87–102. ISBN: 978-36624-4-7-7-4-1.

[Fel] Polkadot Technical Fellowship. *Polkadot Runtimes*. [Online; accessed 8-June-2024]. URL: ↗.

[Foua] Web3 Foundation. *Polkadot Spec*. [Online; accessed 8-June-2024]. URL: ↗.

[Foub] Web3 Foundation. *Polkadot Wiki*. [Online; accessed 23-June-2024]. URL: ↗.

[Fouc] Web3 Foundation. *Polkadot Wiki — Origins*. [Online; accessed 8-June-2024]. URL: ↗.

[Foud] Web3 Foundation. *Polkadot Wiki — Proxy Accounts*. [Online; accessed 25-June-2024]. URL: ↗.

[Foue] Web3 Foundation. *Polkadot Wiki — Pure Proxy Accounts*. [Online; accessed 25-June-2024]. URL: ↗.

[Gal+93] Giorgio Gallo et al. "Directed hypergraphs and applications." In: *Discrete Applied Mathematics* 42.2 (1993), pp. 177–201. ISSN: 0166-218X. DOI: ↗. URL: ↗.

[GR] Jaco GR. *Polkadot JS Extension*. [Online; accessed 8-June-2024]. URL: ↗.

[Ham+19] Sven Hammann et al. "User Account Access Graphs." In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: ACM, 2019, pp. 1405–1422. ISBN: 978-145-036-7-4-7-9. DOI: ↗. URL: ↗.

[Kon] Koniverse. *Subwallet Extension*. [Online; accessed 8-June-2024]. URL: ↗.

[Leda] Ledger. *Ledger*. [Online; accessed 25-June-2024]. URL: ↗.

[Ledb] Ledger. *Ledger — Ledger Nano S+*. [Online; accessed 25-June-2024]. URL: ↗.

[MS15] Diane Maclagan and Bernd Sturmfels. *Introduction to Tropical Geometry*. American Mathematical Society, 215. ISBN: 978-147-046-8-5-6-9. URL: ↗.

[PZ14] Kelly J. Pearson and Tan Zhang. "On Spectral Hypergraph Theory of the Adjacency Tensor." In: *Graphs and Combinatorics* 30.5 (2014), pp. 1233–1248. ISSN: 1435-5914. DOI: ↗. URL: ↗.

[RSG17] Stephan Rabanser, Oleksandr Shchur, and Stephan Günnemann. *Introduction to Tensor Decompositions and their Applications in Machine Learning*. 2017. URL: .

[Rom08] Steven Roman. *Advanced linear algebra*. Springer, 2008. ISBN: 978-038-772-8-3-1-5. DOI: . URL: .

[Sal96] Arto Salomaa. *Public-Key Cryptography*. Springer Berlin, 1996. ISBN: 978-36620-3-2-6-9-5. DOI: . URL: .

[Sat] SatoshiLabs. *Trezor*. [Online; accessed 25-June-2024]. URL: .

[SK20] Alistair Stewart and Eleftherios Kokoris-Kogia. *GRANDPA: a Byzantine Finality Gadget*. [Online; accessed 23-June-2024]. 2020. URL: .

[Tal] Talisman. *Talisman Extension*. [Online; accessed 8-June-2024]. URL: .

[Teca] Novasama Technologies. *Polkadot Vault*. [Online; accessed 25-June-2024]. URL: .

[Tecb] Parity Technologies. *Polkadot SDK*. [Online; accessed 8-June-2024]. URL: .

[TT09] Mayur Thakur and Rahul Tripathi. "Linear connectivity problems in directed hypergraphs." In: *Theoretical Computer Science* 410.27 (2009), pp. 2592–2618. ISSN: 0304-3975. DOI: . URL: .

[Wik24a] Wikipedia contributors. *Blockchain — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-June-2024]. 2024. URL: .

[Wik24b] Wikipedia contributors. *Evaluation Assurance Level — Wikipedia, The Free Encyclopedia*. [Online; accessed 25-June-2024]. 2024. URL: .

[Wik24c] Wikipedia contributors. *Tensor — Wikipedia, The Free Encyclopedia*. [Online; accessed 23-June-2024]. 2024. URL: .