



VITRUEO

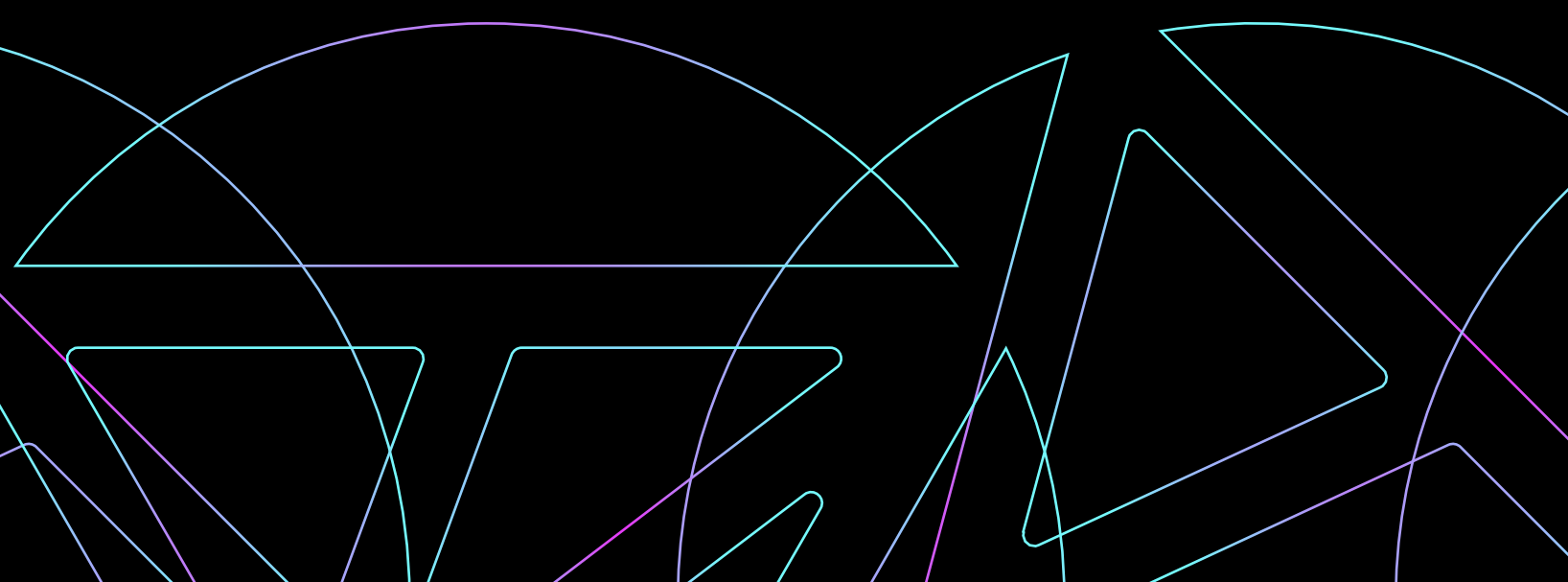
# Create Freely. Earn Fearlessly.

*Blockchain for Creators*

Whitepaper

**V 1.01**

25th December 2023



# Disclaimer

The material provided herein is for informational purposes only. It does not constitute an offer to sell or a solicitation of an offer to buy any interests in any other securities. Certain statements herein may constitute forward-looking statements. When used herein, the words “may,” “will,” “should,” “project,” “anticipate,” “believe,” “estimate,” “intend,” “expect,” “continue,” and similar expressions or the negatives thereof are generally intended to identify forward-looking statements. Such forward-looking statements, including the intended actions and performance objectives of Vitruveo involve known and unknown risks, uncertainties, and other important factors that could cause the actual results, performance, or achievements of Vitruveo in its development of the system, network, its components, and the tokens to differ materially from any future results, performance, or achievements expressed or implied by such forward-looking statements. No representation or warranty is made as to future performance or such forward-looking statements. All forward-looking statements herein speak only as of the date hereof. Vitruveo expressly disclaims any obligation or undertaking to disseminate any updates or revisions to any forward-looking statement contained herein to reflect any change in its expectation with regard thereto or any change in events, conditions, or circumstances on which any such statement is based. You are not to construe this whitepaper as investment, legal, tax, regulatory, financial, accounting or other advice, and this whitepaper is not intended to provide the basis for any evaluation of an investment in an interest. The information provided in this document is for general informational purposes only. It does not constitute, and should not be considered, a formal offer to sell or a solicitation of an offer to buy any security in any jurisdiction, legal advice, investment advice, or tax advice. If you are in need of legal advice, investment advice or tax advice, please consult with a professional adviser. The Vitruveo protocol is under development and is subject to change. As such, the protocol documentation and contents of its website may not reflect the current state of the protocol at any given time. The protocol documentation, document, and website content are not final and are subject to change.

# Introduction



Vitruveo is a Blockchain for Creators. It uses an EVM-based protocol as the foundation for building a decentralized ecosystem to unlock sustainable income for Creators based on trust, technology and community. We're starting with Creators in the visual art domain, then music, gaming, film and other domains.

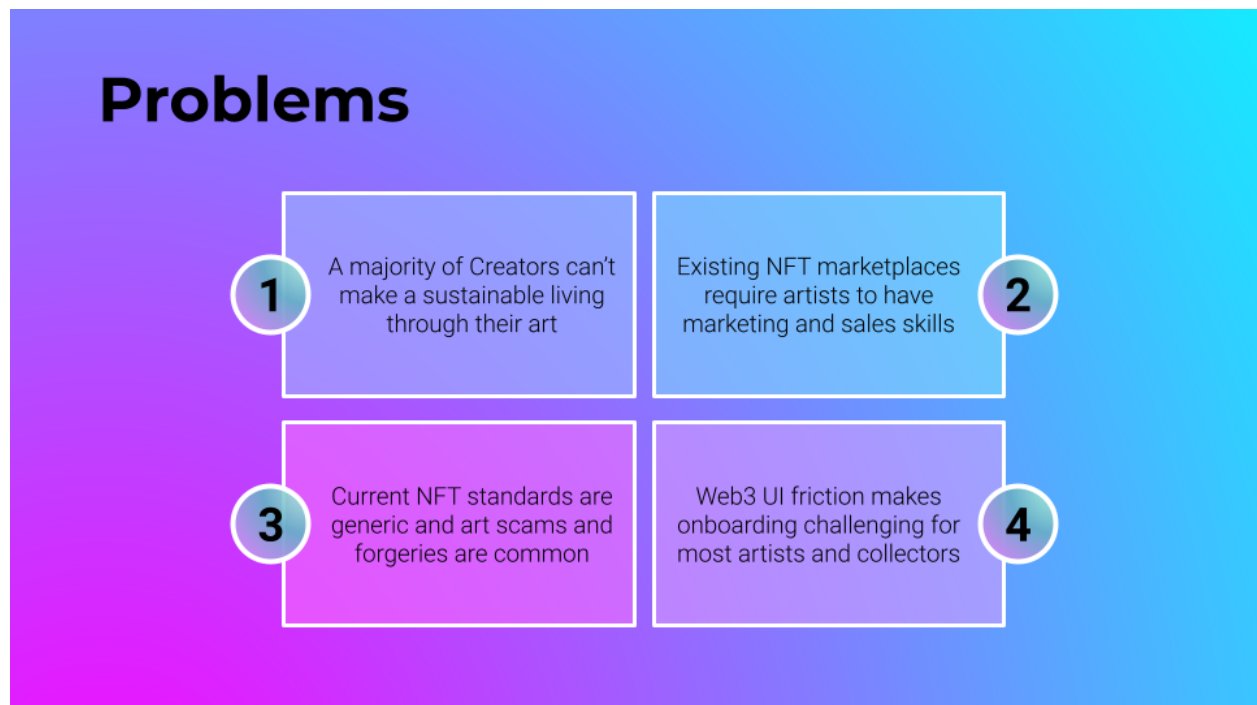
The global art market is approximately \$68 billion, but only around \$11 billion is represented in art-centric NFTs. This is because the current general purpose NFT standards are not very attractive to traditional artists, galleries, art collectors and museums. Also, with the current model Creators have to mint an NFT, then know marketing, sales and social media in order to find buyers.

With Vitruveo, we're building a Vertically Integrated Ecosystem for Creators which focuses on onboarding millions of artists and professional art organizations to Web3 by building trust through Creator KYC, adhering to art metadata standards like LinkedArt, cryptographically signing art media, conforming to established license standards like Creative Commons, and helping Creators beyond just the technology with marketing and social media promotion. We're making Web3 more accessible to the art community

by creating custodial wallets, onboarding workshops, in-person hubs for artists to visit, experience digital art, learn and signup.

Overall, our solution is about making it possible for Creators to have a sustainable income through their art. We are doing this with a blockchain at the foundation and everything in the stack above to focus exclusively on Creators.

## Problems

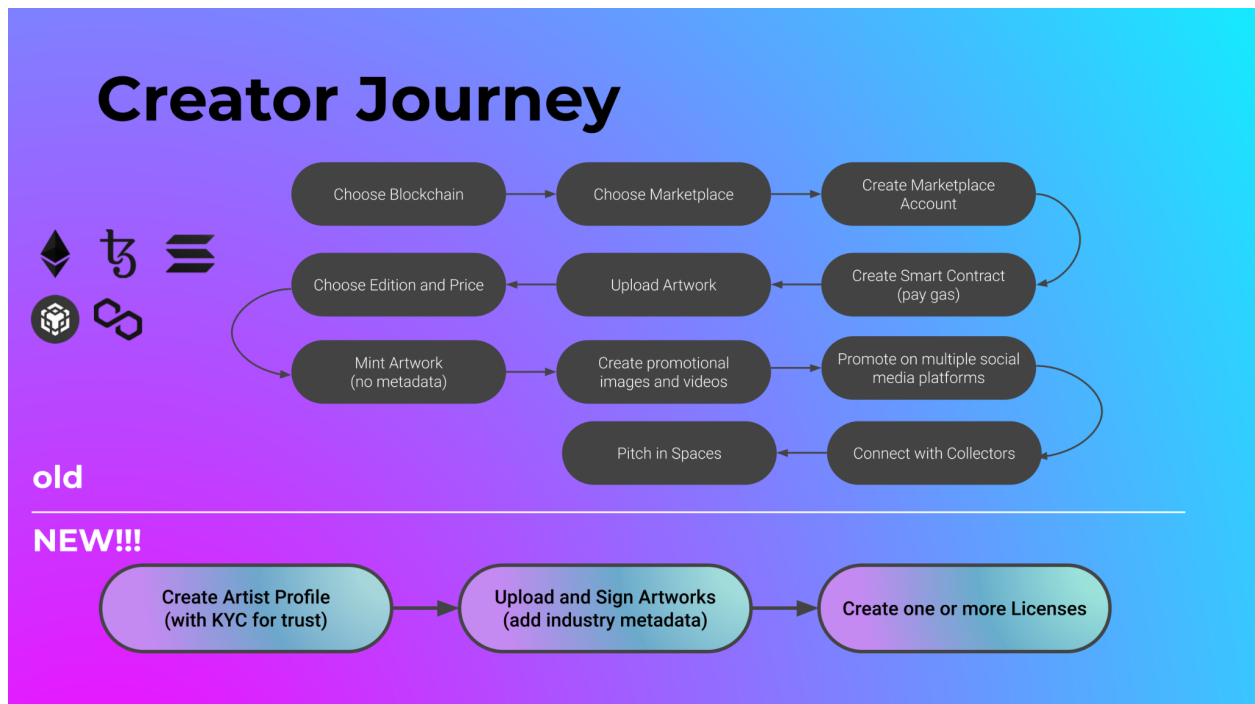


Creators in Web3 today are faced with four primary challenges:

1. **Income:** Due to its evolving nature, Web3 isn't currently well-suited for Creators to earn a full-time income. Sales vary significantly based on the overall crypto market performance and there aren't very many buyers. Furthermore, activity is fragmented across protocols and marketplaces on those platforms making it more challenging. But even in the Web2 world, Creators face similar challenges. Gate-keeping by galleries is a real problem for Creators which is why many of them are starting to look to Web3 in the first place.

2. **Skills:** Existing NFT marketplaces are built on a “Do It Yourself” model. Creators need to learn how to use the technology, list their artworks for sale and then teach themselves how to promote themselves and their work. Artists are forced to learn marketing, sales and social media skills when they would rather be creating art.
3. **Bad Actors:** The complete absence of standard for NFT art has resulted in chaos. NFT marketplaces are like street bazaars with every genre, style, quality and size of artwork thrown together with non-existent metadata for searchability and organization. With no checks for quality or authenticity, fake artworks and scammers have a field day, discouraging many artists and art organizations from entering Web3.
4. **User Experience:** The overall user experience of Web3 continues to be an impediment to large scale adoption. Users have to learn a mind-boggling amount of technology jargon and work with complex software apps and concepts in order to get even basic tasks accomplished. This means only the most determined and patient artists even reach the point where they can list their work on a Web3 marketplace.

## Creator Journey



The Creators who have gained the most traction in Web3 are artists. While Vitruveo is designed to address the needs of all Creators, our initial focus is on artists and the Art industry, which is what we'll focus on for the rest of this document.

Artists who are in Web3 today are primarily finding success on Ethereum, Tezos and Solana, with limited traction on Binance Smart Chain and Polygon. None of these protocols have anything specific for Creators. All the capabilities are provided by third-party marketplaces each with their own brand, features, benefits, policies, Artists and collector audience.

The journey for Artists is daunting and involves making a commitment with imprecise information. Artists need to choose a protocol, then a marketplace on that protocol, abide by all the policies of that marketplace, figure out whether to list their artwork as a single or multiple edition, determine the price point, pay for minting without any guarantee of sale, and that's just the beginning. Then comes the endless cycle of promotion – daily X and Instagram posts, waiting on X Spaces for hours and hours to be able to speak for 2-3 minutes – and all of it may be in vain. Many Artists don't see a single sale for many months, while a small segment of 2-3% see recurring sales.

## Vertically Integrated Ecosystem



Vitruveo is solving all these problems with a **Vertically Integrated Ecosystem**.

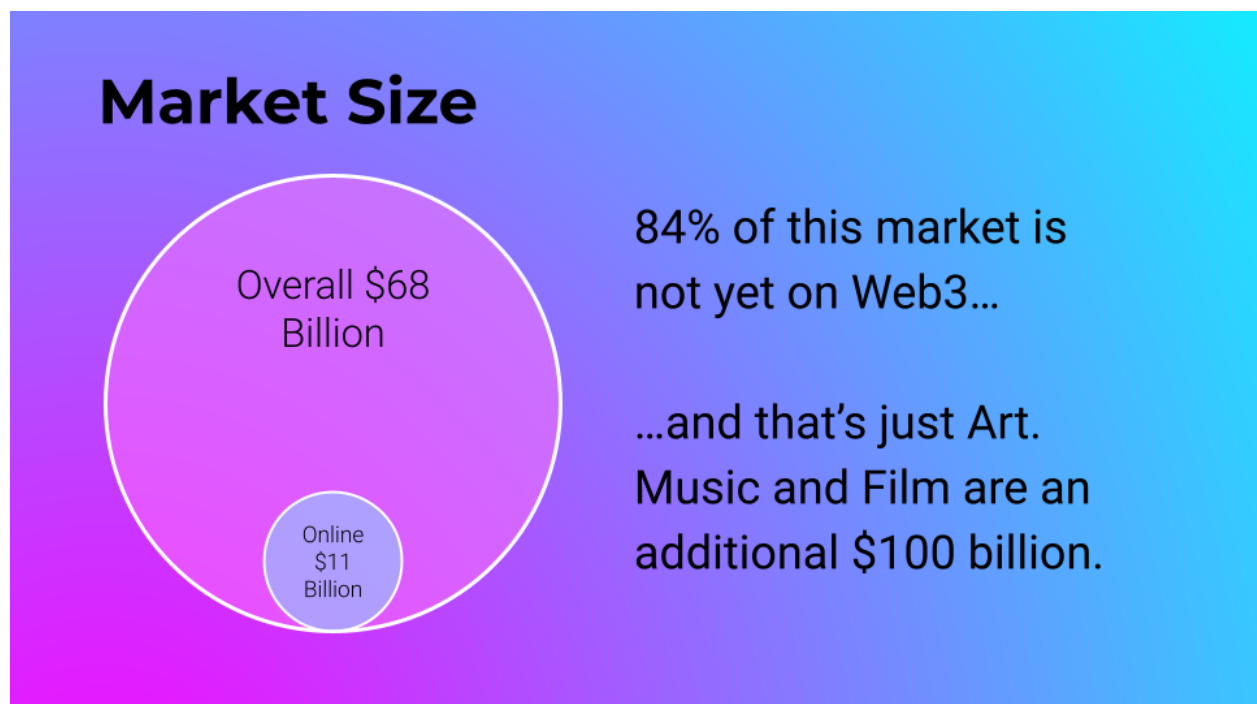
Borrowing a page from Apple's playbook, we are starting with the technology foundation of a blockchain but taking it all the way up the stack with Smart Contracts, Dapps, Marketplace, Tools and Resources, Workshops, Community Engagement and Marketing.

By creating a one-stop shop for Creators, Vitruveo will not only attract all stakeholders in the art ecosystem, but the ready audience will also attract developers and third-party service providers to build and provide services on the platform. Additionally, we can concentrate our marketing and community outreach resources in a very targeted manner.

Today, when you think of eCommerce, the first company that comes to mind is Amazon.

In the future, when you think of Web3 Creator Ecosystem, **Vitruveo** will be the first mention.

## Market Analysis

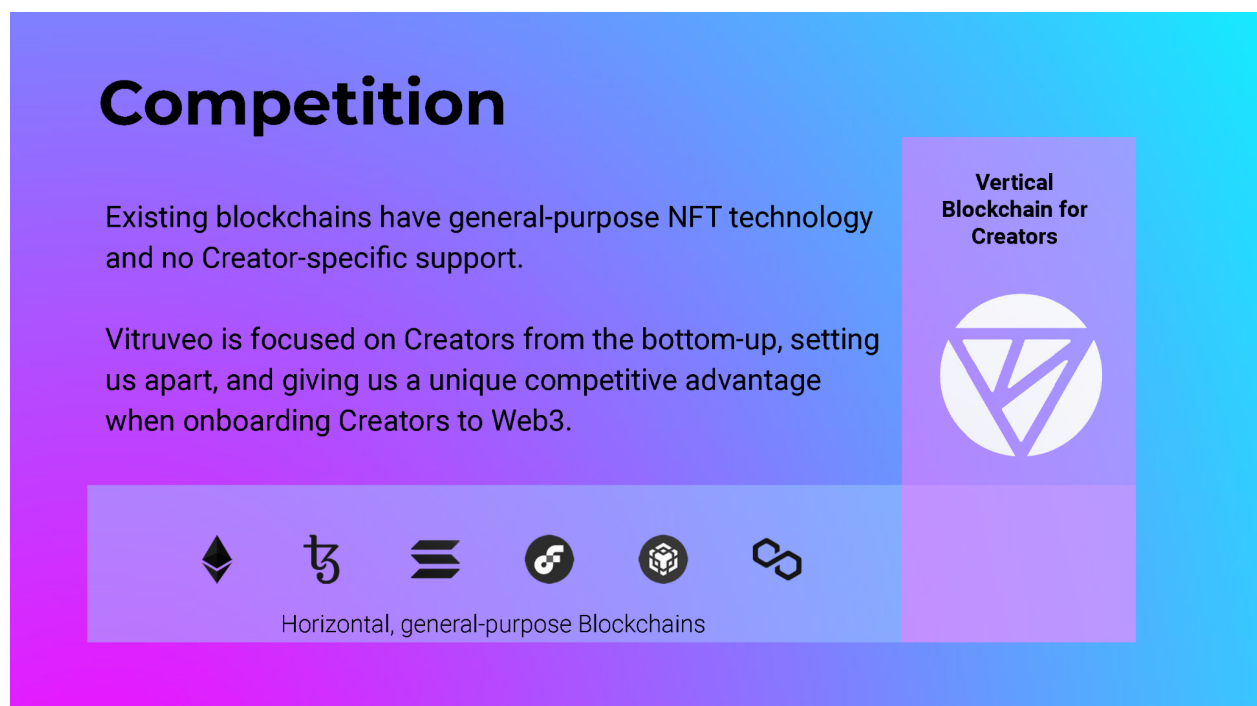


The Art market is large — over \$68 billion. Of this market, 84% remains outside Web3. This is Vitruveo's market opportunity. By creating a blockchain-based economy for

Creators, focusing on a great user experience, adhering to industry standards and focusing on trustworthiness, Vitruveo can onboard a large portion of this 84% onto Web3. We can learn from and optimize our marketing, outreach, onboarding workflows to then rapidly onboard music, gaming, film and other domains which also have very large footprints.

## Competition

One of the biggest advantages that Vitruveo will have over other protocols is our focus on Creators. Other blockchains have individual marketplaces that each provide a different feature set, different user experience, and have different policies for Creators. On Vitruveo, since the asset registry is at the protocol level, all apps and marketplaces will have a uniform set of standards and can be more focused on marketing, discovery and attracting buyers.

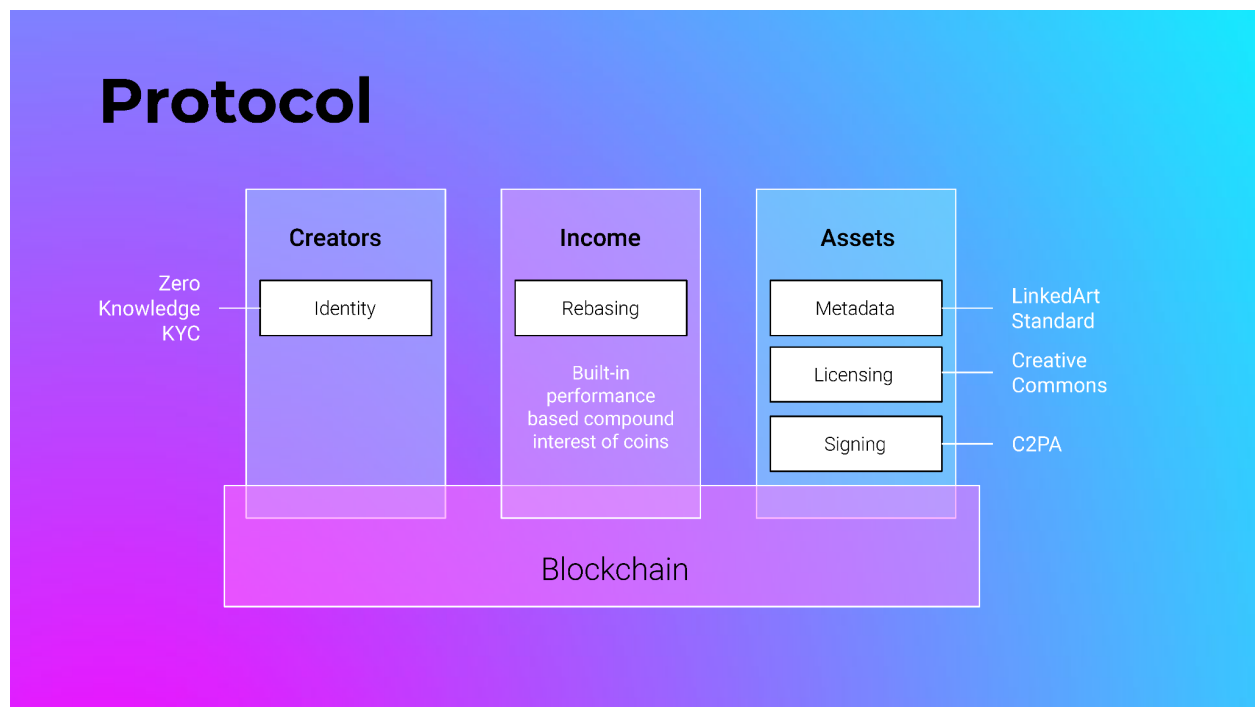


# Technical Overview

Vitruveo is an L1 blockchain that is 100% Ethereum Virtual Machine (EVM) compatible. It uses a Proof of Authority consensus mechanism with a 500 Validator network. The protocol uses GoEthereum, the same software used on the Ethereum blockchain with the following modifications:

## Rebasing

Vitruveo is the world's first rebasing protocol. Every other protocol with rebasing to-date implemented it using a Smart Contract. With Vitruveo, the core protocol has rebasing or built-in.



In simpler terms rebasing is "compound interest." However, without any damping mechanism, rebasing would result in an inflationary cryptocurrency. Vitruveo solves this by coupling rebases to epoch-based transaction volume.

On Vitruveo, the block time is 5 seconds. Each epoch is 17,280 blocks which is 24 hours. At Genesis, the epoch transaction goal is set to 10,000. If this goal is achieved within the epoch with a +/-25% variability, then all \$VTRU (the native coin), regardless of whether it is in a Smart Contract or an Externally Owned Account, increases by

1.00087671 (with +/-25% variability). Each epoch the transaction goal increases by 500, ensuring that the ecosystem has to continually perform well in order to **earn** the rebase. The rebasing mechanism stops when the circulating supply reaches 250 million.

The rebasing feature is built into the protocol and is highly performant. Users with wallets such as MetaMask will see their balance automatically increase in real-time when a rebase occurs. Since rebasing is expected to work only for the first 5-7 years, we call it “Early Adopter Rewards” as it provides passive income to all \$VTRU holders.

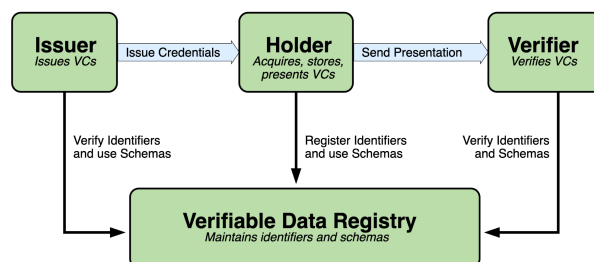
## Creator Base

Vitruveo has protocol-level support for **TruID**, Vitruveo’s solution for validating the true identity of Creators. This is our W3C [Verifiable Credentials](#) compatible ID for Creators. TruID has **zk-KYC** (“Zero Knowledge Know Your Creator”). Third-party providers perform KYC and return the KYC level of a creator, ensuring their privacy.

TruID verification is paid for by Vitruveo and has multiple levels. L1 is required, other levels are optional:

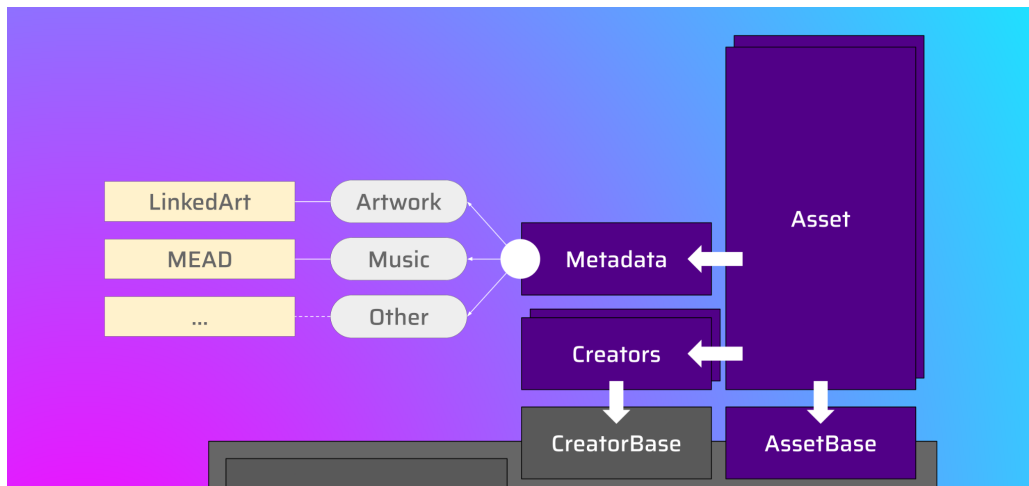
- L1: Email (1 Creator Credit) – required
- L2: Phone (+2 Creator Credits) – optional
- L3: Proof of Address (+3 Creator Credits) – optional
- L4: Liveness (+4 Creator Credits) – optional
- L5: ID Document (+5 Creator Credits) – optional

Feature availability on Vitruveo and third-party apps is linked to TruID levels – higher TruID levels mean more freedom and benefits. Additionally, higher TruID levels also mean more “Creator Credits” which are required for registering assets into AssetBase (described in next section).



## Asset Base

Vitruveo has protocol-level support for Creators to consign their assets. This ensures that consigned assets are available to every dapp on the platform and Creators do not need to individually upload to different dapps or marketplaces.



One of the most unique and innovative features of Asset Base is standards-compliance. Assets registered with Asset Base conform to three standards:

- 1) MetaData Standard:** Vitruveo requires assets to conform to industry standard metadata requirements. In the case of visual art, the standard is [LinkedArt](#) which is used by top museums around the world. For music, the standard is [Media Enrichment and Description](#) (MEAD).
- 2) License Standard:** Vitruveo tokenizes licenses and requires them to conform to [Creative Commons](#), the same standard used by YouTube and other major organizations. There are different License Types (NFT, Streaming, Print on Demand) giving Creators significant flexibility in licensing their artworks for different use cases.
- 3) Signing Standard:** Vitruveo requires Creators to digitally sign their artworks using the [Coalition for Content Provenance and Authenticity](#) (C2PA) standard. This makes it harder for copycats and also provides a mechanism for Creators to affirm their authenticity of their work.

# Creator Support

## Creator Benefits

Everything about Vitruveo is designed to benefit Creators.



## Creator Income

Unlike existing Web3 scenarios where Creators only have one form of income – NFT Sales, Vitruveo changes the dynamic significantly by unlocking new and innovative forms of Creator Income. This is possible through our unique Licensing Architecture that enables Creators to License the same asset for different use cases without requiring an on-chain token to be minted for every scenario.

The basic concept behind the software industry's model for wealth creation is to build something once and then sell it in unlimited quantities with little or no additional cost.

You can't do this with tangible products because there is the cost of manufacturing, shipping etc. You can't do this with human-delivered services because there is the cost

of employing people and the time/labor involved in delivering the service. How can we apply this learning to Web3 art?

We want art to maintain its scarcity, hence the concept of NFT editions. But it isn't possible to scale this up infinitely because it is like the latter — a human-delivered service.

So how does the art industry create a similar model to the software industry for income generation and dare we hope — wealth creation?

For the solution, you have to look beyond Asset Tokenization, which is the basis of NFT 1.0. Instead, you have to bring into the picture License Tokenization, which I call NFT 2.0.

You can continue to have Asset Tokenization, but with License Tokenization, you have the ability to scale art to infinite quantities exactly like software.

License Tokenization is fundamental to Vitruveo because we want to enable wealth creation for artists. It is also standards compliant with Creative Commons because standards mean easier comprehension, adoption and scalability.

Let's look at the types of Licenses you can tokenize with Vitruveo:

**NFT:** This is your standard edition license. Except you don't have to think about whether to make it a 1/1 or 1/n. You can always make it a 1/n where "n" can be 1 or more. You check a box "Elastic Editions" and the Buyer decides for you.

For example, you create a license for 1/10 at an edition price of \$10. If "Elastic Editions" is checked, a Buyer can pay \$100 and make it 1/1.

Another example, with the same pricing and edition size. Two people buy so it's at 2/10. A Buyer can come along, pay \$80 and make it a 3/3.

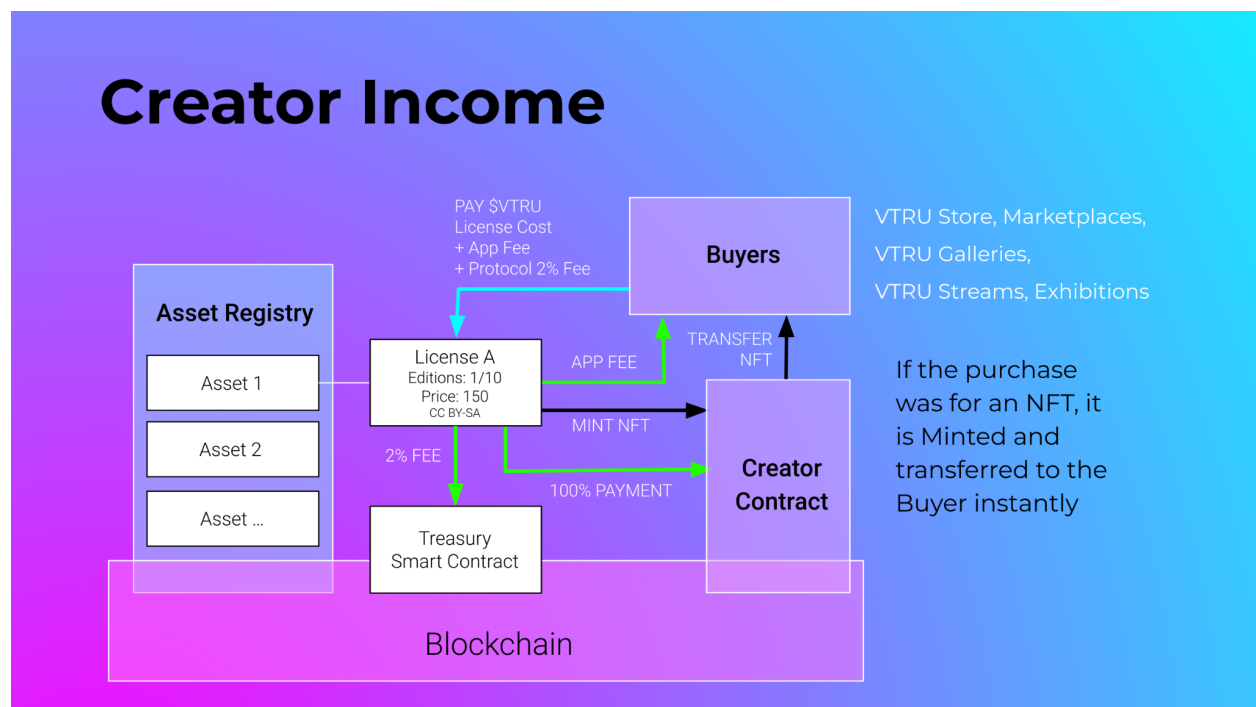
The Edition size maximum is set by the Artist, but with "Elastic Editions," the Buyer can choose the Edition size minimum.

**Print:** This is a license designed for consumers or consumer product manufacturers. No NFT is created. You license your art to be printed anywhere for a fee and a volume discount.

For example, you can set the unit price at \$2 and volume discount of 1% per 1000 units with 50% discount maximum. If someone manufacturing scarves wants to use your artwork for 10000 units, they would pay  $(\$2 \times 10000) - 10\% = \$20000 - \$2000 = \$18000$

But no manufacturer is going to magically find your art and use it for production at scale. This is where VTRU Gallery (see Appendix) comes in. We will enable Curators to become Distributors by making it easy for them to create microsites – Galleries – with curated artworks that have Print Licenses and offer them to consumer goods manufacturers. This is the secret sauce – we enable Curators to make money while promoting your art by creating Galleries

**Stream:** More and more digital frames purpose-built to display artworks continue to enter the market. OTT (Over the Top) devices like AppleTV have the ability to display art as screensavers. With VTRU Stream, Vitruveo Creators can have their work licensed and streamed to homes, businesses, hotel lobbies, airports, train stations and hundreds of other locations. This “Spotify for Art” service will enable passive revenue generation for many Creators.



# Go-to-Market

Vitruveo's go-to-market strategy is to pursue Creators domains in the following order: Visual Arts, Music, Gaming, Film, then all other.

## Creators

Our approach to onboarding Creators in Visual Arts is to first connect with and engage Creators already in Web3 directly and through arts communities through token incentives and giveaways. Next, we'll have outreach to Art Organizations such as galleries and museums to batch onboard their Creators with token incentives. Finally, we will partner with social impact organizations to onboard indigenous artist communities and underprivileged artists around the world.

## Buyers

In order to get sales of artworks, we must onboard buyers and collectors to Vitruveo. Here's our plan.

- **Art Influencers:** Outreach via publications, blogs, YouTubers, Instagrammers etc.
- **Galleries:** When we onboard Galleries, we'll onboard their collectors as well.
- **Exhibitions:** We'll host online and offline exhibitions to connect with collectors.
- **Retail Partners:** Retail partners with personalized art goods will bring new collectors.
- **VTRU Hubs:** VTRU Hubs in major cities will be community hubs and attract both Creators and Collectors. We'll model these like Apple Stores.

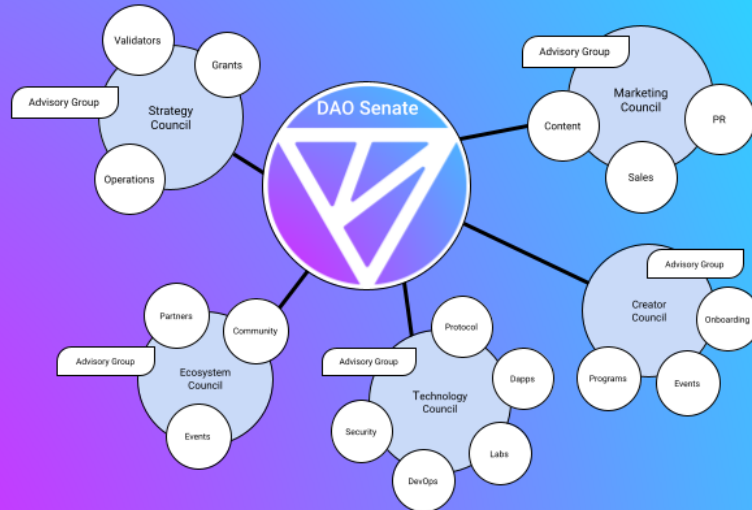
# Ecosystem and Community

Vitruveo has a vibrant community of 10K+ users combined on social media. We are organized as a DAO with five councils: Strategy, Technology, Creator, Ecosystem and Marketing.

Our [Governance Charter](#) describes the DAO in detail.

# Vitruveo n/DAO

Vitruveo n/DAO (nascent DAO) has a governance charter and is organized into five Councils with a nine-member Senate. Voting is based on Vote Credits issued to members at Genesis.



## Tokenomics

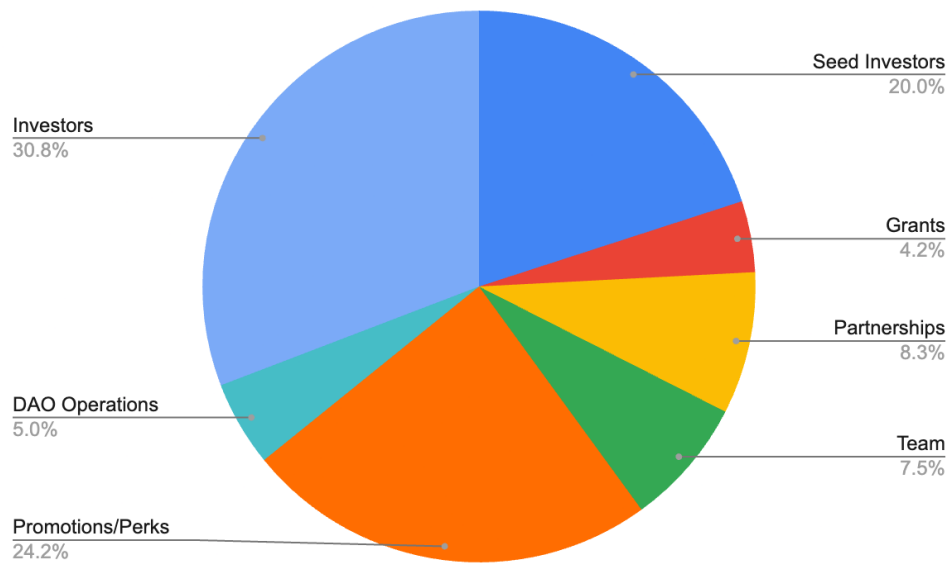
Vitruveo is the first protocol to have Performance-based Rebasing built-in (no Smart Contract required).

For the first 5 years, all **\$VTRU** holders will benefit from having their coins compound daily!



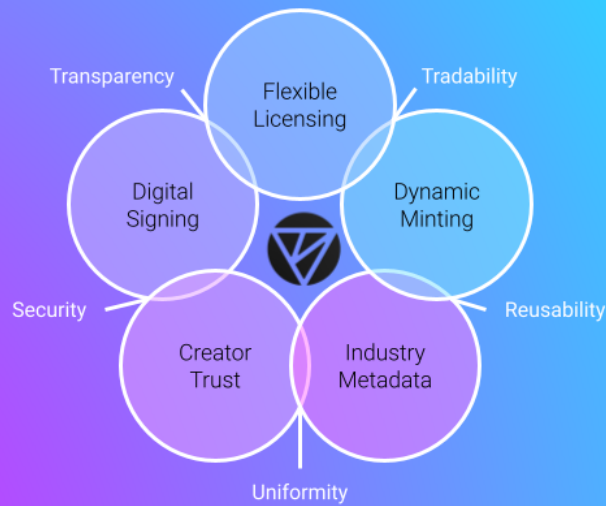
The initial circulating supply of \$VTRU is 60 million. Of these, 40 million are locked or vesting for the first 6 months. With rebasing, the coin is expected to grow 4X over a period of 5-7 years with a maximum supply of 250 million.

## \$VTRU Distribution



# Trust

Our protocol works seamlessly to deliver the missing element of Web3 Art —  
**TRUST**



*We are architecting Vitruveo from the ground up to be the very best Blockchain for Creators. Over the past two years I have spoken to thousands of artists, collected thousands of artworks, hosted many exhibitions both virtual and irl, and studied every aspect of the Web3 art ecosystem: marketplaces, contracts, royalties, wallets, licenses, UIs ... all of it.*

*There are many things that work, but there are many more that are broken. The biggest problem we have to fix if we want to get more artists and art appreciators in the space is TRUST.*

*Web3 has all the technology elements for trustlessness, but ironically, these very elements have thwarted the growth of Web3 art by letting scammers, copycats and fraudsters run amok. The fix for this is to dial the decentralization back a little and add controlled centralized elements by conforming to current industry standards that are trusted by many, especially professionals in the art industry. My vision for Vitruveo blockchain is to build the most trusted blockchain for creators in existence.*

*I think of all the key features of the blockchain as circles that work harmoniously to create a single Circle of TRUST that becomes the foundation of everything we do in our vertically integrated ecosystem.*

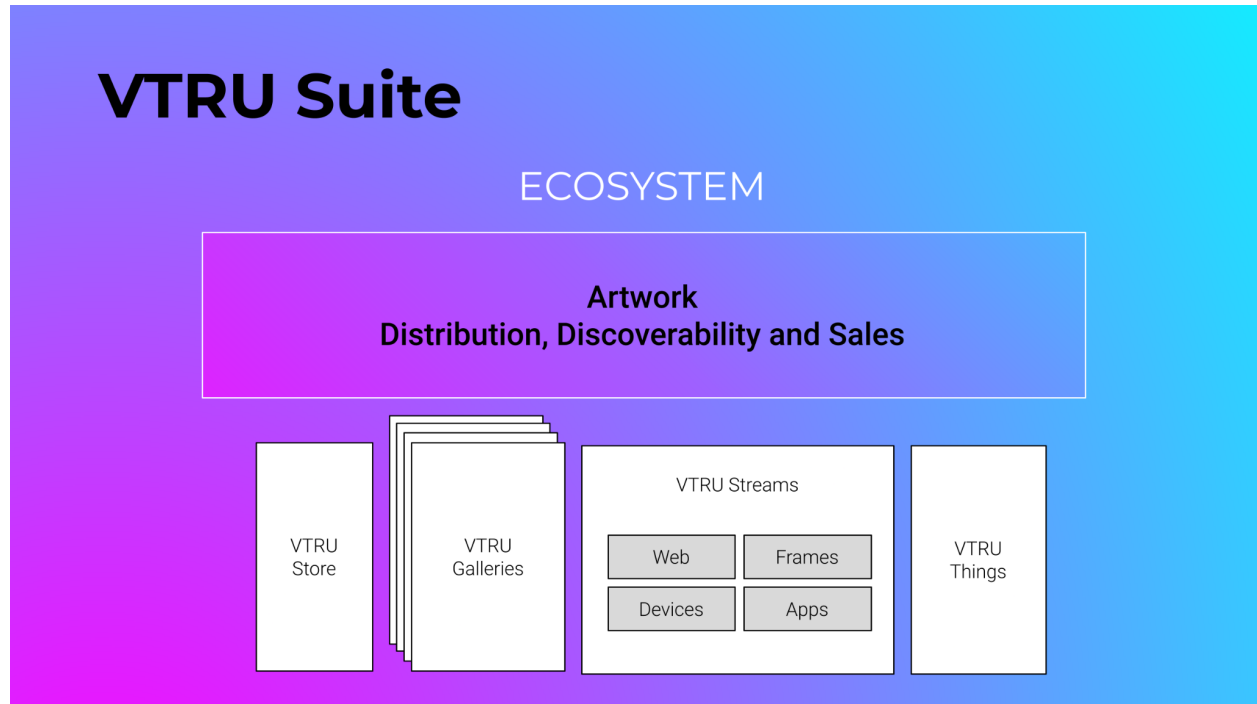
**Vitruveo — it's all about TRUST!**

# Appendix – Platform Dapps

In keeping with our goal of being a Vertically Integrated Ecosystem for Creators, Vitruveo will build and maintain best of class decentralized applications needed for Creators to be successful and to make a sustainable income.

The following pages describe the main Dapps currently envisioned. Through grants, partnerships and strategic acquisitions we will also encourage other Creator-centric dapps to build and launch on Vitruveo.

# VTRU Suite



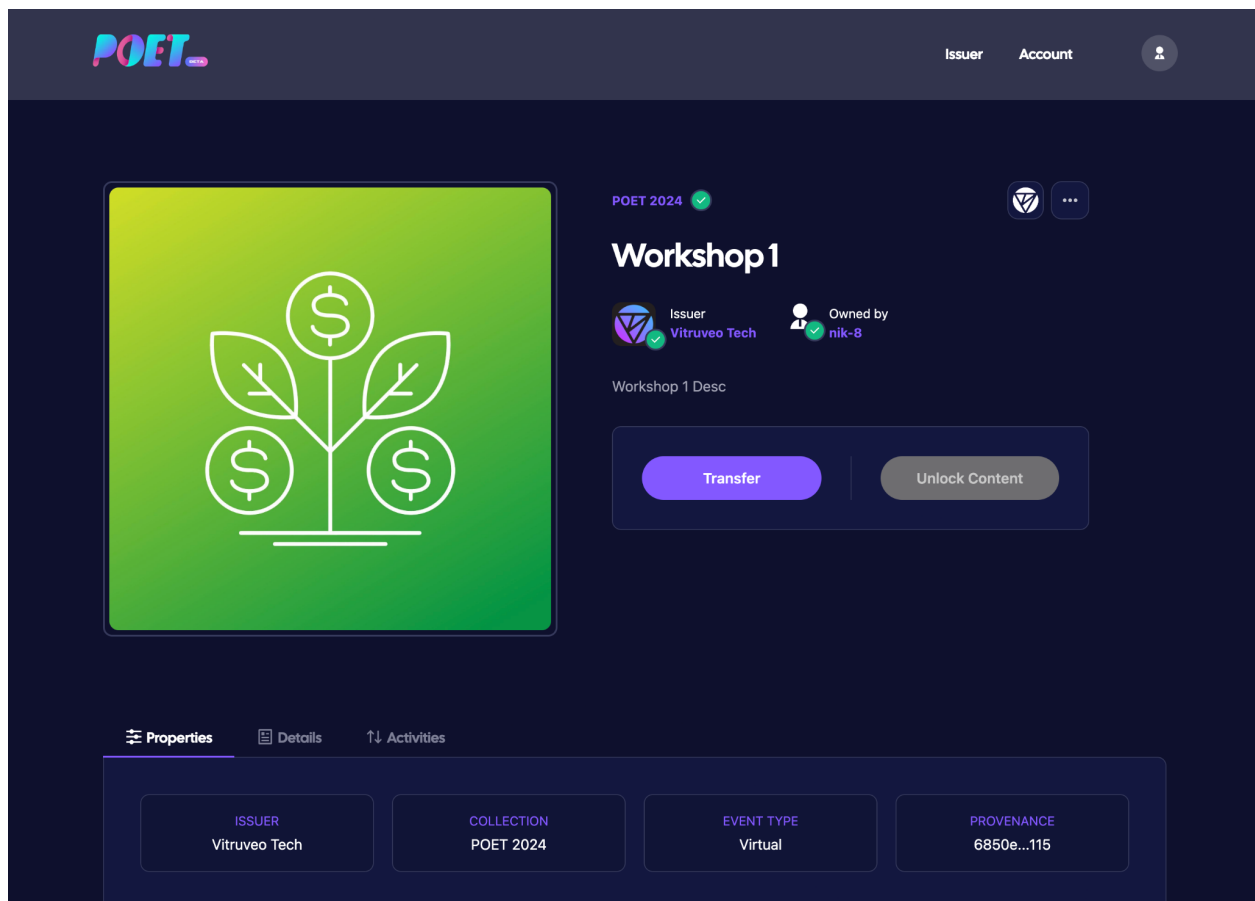
VTRU Suite is a set of dapps designed to increase distribution, discoverability and sales of artworks.

- VTRU Studio: Anyone can consign an artwork (visual art, music, film, poem etc.) with standards-based metadata and licensing terms
- VTRU Store: Our reference marketplace. We will encourage other marketplaces on Vitruveo. Marketplaces will need to use Asset Base (versus having Creators mint on their platform).
- VTRU Gallery: Anyone can create a store to sell artworks consigned to Vitruveo through VTRU Studio “Shopify for Creators”
- VTRU Stream: Anyone can build apps to stream licensed visual artworks to any streaming device like AppleTV etc. “Spotify for Creators”
- VTRU Things: Partner with Print on Demand services to feed artworks for personalized goods.

# POET (Proof of Engagement Token)

POET is a Vitruveo app designed to enable any form of engagement to be tokenized. For example, attendees at an event, students who took a course, people who voted etc.

POET is free to use and only requires an email address thus making it very easy for Web2 users to discover and connect with Web3 platforms and apps.



# PhotoKey

PhotoKey is a secure, visual and user-friendly authentication solution for modern desktop and mobile web browsers. It combines a photo with a sequence of 4-9 emojis called an “EmojiKey” to authenticate a user to any Web 2.0 or Web 3.0 application using the [OpenId Connect](#) standard.

If you happen to mention to a person with knowledge of digital photo formats that you are using photos for security, chances are good that they’ll make a reference to “steganography,” the technique of hiding secret data within an ordinary, non-secret, file or message in order to avoid detection. In modern digital steganography, data is first encrypted or obfuscated in some other way and then inserted, using a special algorithm, into data that is part of a particular file format such as a [JPEG](#) image. The secret data is then extracted at its destination and transformed back to its unencrypted state, revealing the original data.

**PhotoKey does not use steganography.** This is an important assertion to make at the onset because steganography is fairly easy to break with today’s computers and is a poor choice for use in an authentication system.

With that out of the way, let’s take a look at some background information and then review the technical underpinnings of PhotoKey.

## Why is PhotoKey needed?

“Necessity is the mother of invention,” the oft-used proverb states, and never has this been truer than in the case of PhotoKey. It was born out of frustration from enduring the rough user experiences foisted on people by many blockchain “wallets.” There are four significant problems with these solutions:

1. **The term “wallet.”** Since blockchain wallets are typically just a storage list of addresses with obscured private keys and don’t contain any funds as a real-life wallet does, the term is very confusing to new users. With no guidance, they continue to believe that their wallets are a store of value which results in much confusion as their blockchain usage increases.

2. **Conflation of concerns.** Most blockchain wallets have a mechanism through which the user can prove they are the rightful owners of the private keys contained within (“authentication”). In addition, they provide functionality for sending cryptocurrency to other accounts (“payments”), signing transactions and reporting the cryptocurrency balance and transaction history. The problem is that this is a rather one-sided approach to the future of the decentralized web — it presumes that all blockchain apps will require cryptocurrency. A Web2.0 analogy would be a “Sign in with Google” button that insisted on foisting “Google Wallet” on you every time you tried to sign in.
3. **One user, many blockchains.** Blockchains are nascent and as such applications that incorporate this technology make liberal mention of it in the user interface. For each separate blockchain app a user interacts with, they need to download and install a different wallet. In the near future, when the vision of the decentralized web has been realized and blockchains are a commoditized back-end technology, will users continue to tolerate the requirement to have a different wallet for each blockchain?
4. **Private keys. Passwords. Mnemonic Passphrases.** It was bad enough that users had to remember so many different usernames and passwords on Web2.0 websites. Then blockchain and cryptocurrencies came along and users now have to save and keep track of many private keys, passwords and mnemonic passphrases. The situation is absurd! No wonder mainstream users are not signing up for blockchain apps in droves.

A solution to these problems would need to have the following characteristics:

- Not a wallet.
- “Authenticate” a user by securely granting them access to their private key without going through an alternate password-based store or gatekeeper.
- Work across all websites and blockchains, current and future.
- Not require the user to memorize (or even encounter) words such as “private key,” “mnemonic,” “keystore” and other jargon.
- Extremely secure.

And a few more nice-to-have characteristics:

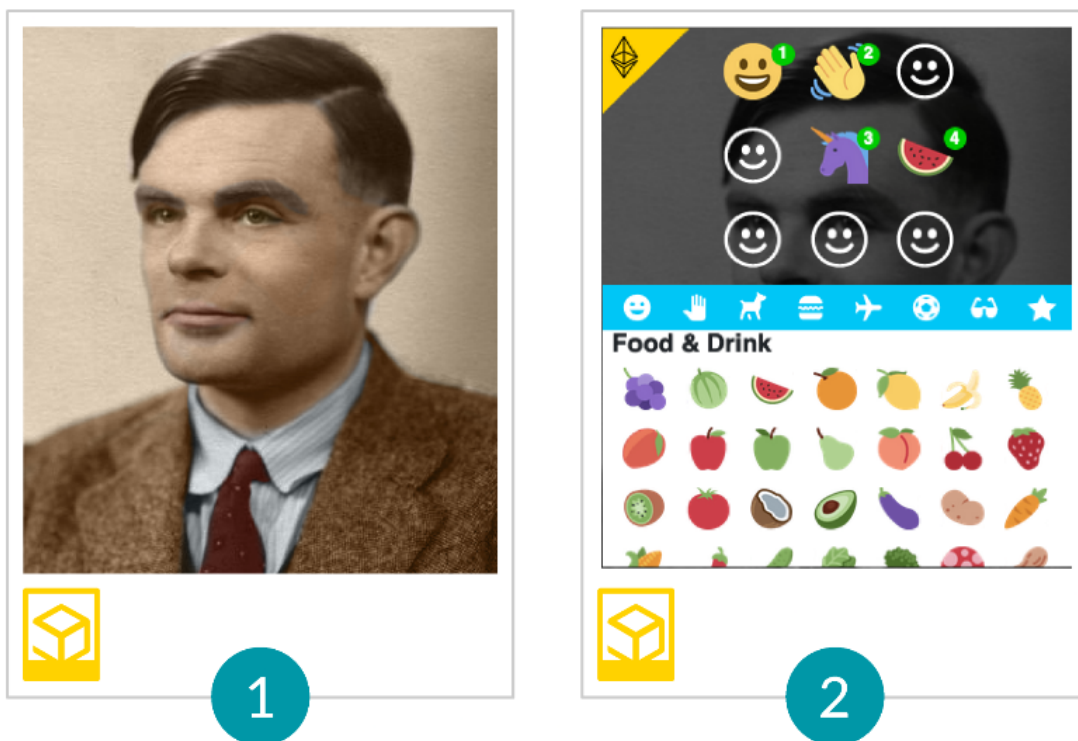
- Fully decentralized
- Fully client-side

- No download or installation required
- VERY SIMPLE USER EXPERIENCE

Implementing these characteristics in a software product resulted in PhotoKey.

## How does PhotoKey work?

From the user's perspective, the PhotoKey experience is very simple — they upload any photo, specify a sequence of emojis and download the newly created PhotoKey. From then on, they can drag/upload the PhotoKey any time they see a PhotoKey login dialog. Once used on a device, the PhotoKey is cached so the user doesn't need to upload it again.



User uploads their photo

User creates EmojiKey by choosing emojis at each of nine positions in sequence.

When the user first uploads a photo, PhotoKey checks to see if it is a JPEG image. If it is, it then examines the photo's embedded [XMP](#) packet to determine if the image is a

PhotoKey. If the image has been previously processed it will contain an XMP packet using the <http://ns.photokey.org/xmp/1.0/> namespace.

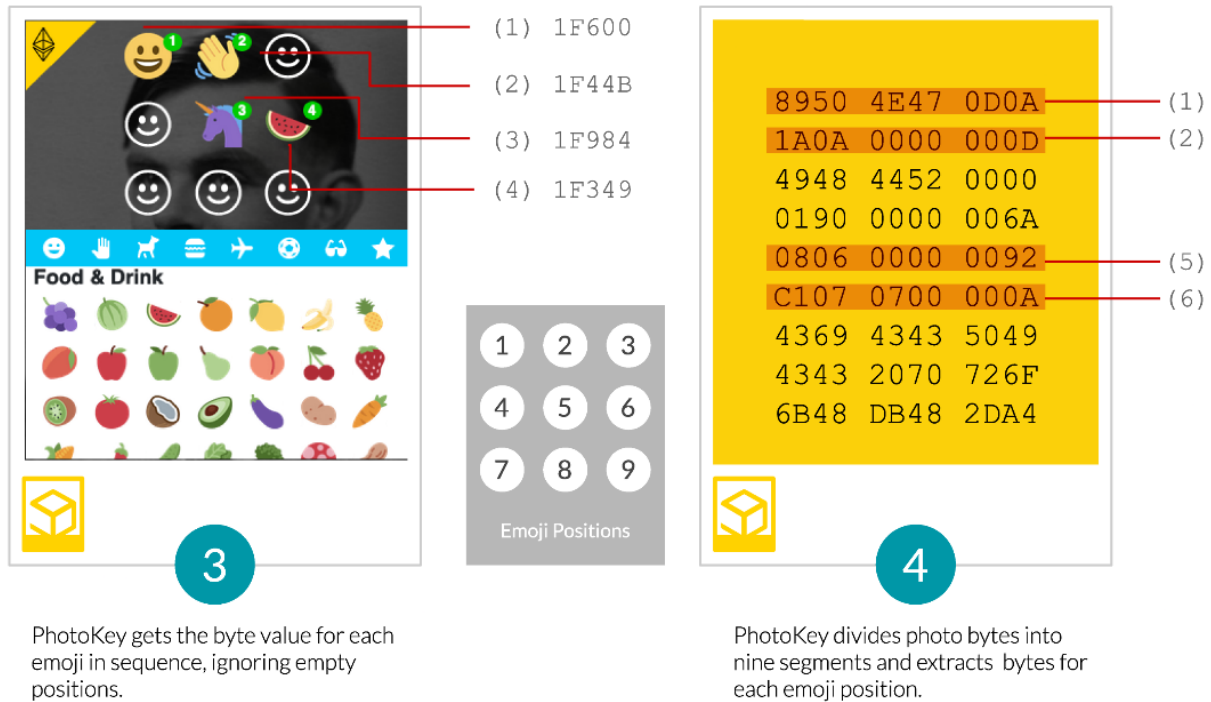
3352	F015	6272	D10A	1624	34E1	25F1	w	!1	AQ	aq	"2.	B....	#3R.	br.	\$4.%.
7A82	8384	8586	8788	898A	9293	9495	&'()	*56789:	CDEFGHIJSTUVWXYZ	cdefghijstuvwxyz.....					
E7E8	E9EA	F2F3	F4F5	F6F7	F8F9	FAFF	.....0.....								
6E3D	2222	2069	643D	2257	354D	304D	.	http://ns.adobe.com/xap/1.0/	<?xpacket begin="	" id="W5M0M					
653A	6E73	3A6D	6574	612F	223E	3C72	pCehiHzeSzNTczkc9d">	<x:xmpmeta xmlns:x="adobe:ns:meta/">	<r						
2D72	6466	2D73	796E	7461	782D	6E73	df:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns								
7470	3A2F	2F6E	732E	6164	6F62	652E	#">	<rdf:Description rdf:about="	" xmlns:xmp="http://ns.adobe.						
6F62	6C6F	636B	2E6F	7267	2F78	6D70	com/xap/1.0/"	xmlns:photoblock="http://ns.photoblock.org/xmp							
6F74	6F62	6C6F	636B	2E6F	7267	2F78	/1.0/">	<photoblock:Web xmlns:web="http://ns.photoblock.org/x							
6636	6338	6461	3866	3032	3963	6439	mp/1.0/Web#">	<rdf:Seq>	<rdf:li web:username="75f6c8da8f029cd9						
3236	2220	7765	623A	6E61	6D65	3D22	0543900f1da7eb384a4f7beae21c95d83ab2b148f1ab5026"	web:name="							
6634	6265	3533	3432	3266	3266	3134	aeeb295b18324326efe70dd5f506a11146143d362e8262f4be53422f2f14								
3335	6331	3661	6135	3834	3564	6539	f13a"	web:publicKey="7fe06342f24d8823bf4016e7735c16aa5845de9							
3037	3939	3038	3039	3465	3131	3132	ed3aada04e5779402ff8297c9"	web:import="69217a3079908094e1112							
2F72	6466	3A6C	693E	3C2F	7264	663A	1d042354a7c1f55b6482ca1a51e1b250dfd1ed0eef9">	</rdf:li>	</rdf:						
4446	3E3C	2F78	3A78	6D70	6D65	7461	Seq>	</photoblock:Web>	</rdf:Description>	</rdf:RDF>	</x:xmpmeta				
800A	28A2	800A	28A2	800A	28A2	800A	>	<?xpacket end="w"?>	..	?	..	..	..	..	..
800A	28A2	800A	28A2	800A	28A2	800A	..	..	..	..	..	..	..	..	..

The data contained in the XMP packet is hashed versions of the following:

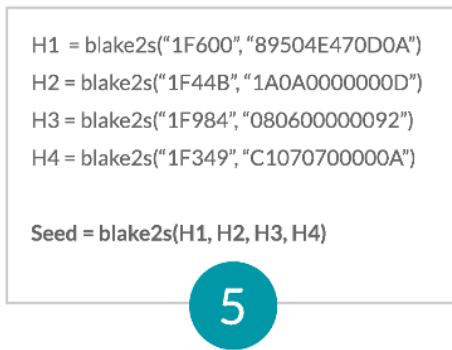
- 1) **UserId:** A deterministically generated user ID in the format {adjective}-{phonetic}. This UserId serves as the user's login for websites. In the future, when we enable inbound email, the UserId will also serve as the username portion of the user's email address. Example: **talented-osotied**. The user may customize this user ID.
- 2) **DisplayName:** This is simply the UserId displayed in first name and last name format. Example: Talented Osotied.
- 3) **PublicKey:** This is the user's web public key for use in scenarios where the user needs to share the key with a host for future session validation scenarios.

When a user wishes to authenticate, they start by uploading their PhotoKey and entering their EmojiKey which is a sequence of Emojis. The Emojis used by PhotoKey are based on [Twitter's Open Source Emoji](#) set. We currently use version 11, but will soon upgrade to version 12.

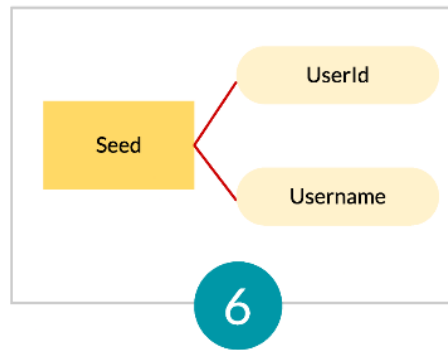
By default, a minimum of four emojis are required, but up to nine may be entered. When entering the Emojis, the position and sequence of Emojis is very important.



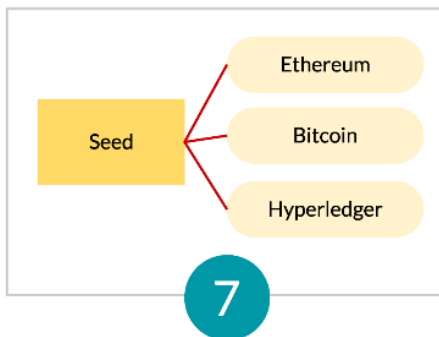
Once the Emojis are entered, PhotoKey divides the photo bytes into nine segments. For each segment corresponding to an Emoji, it calculates the blake2s hash of the Emoji bytes, the blake2s hash of the photo segment bytes and finally a combined hash. This hash will serve as the seed for calculating the private key for each blockchain supported by PhotoKey.



PhotoKey uses the Blake2s algorithm to hash emoji bytes with photo bytes for each position, and finally, produces an aggregate hash of the positional hashes. This is the high-entropy seed for keygen.

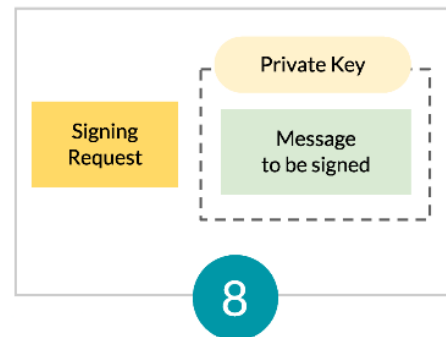


The seed is used to deterministically generate a UserId and Username in "adjective - phonetic word" format. Their hash is compared to the hash stored in the XMP (eXtensible Metadata Platform) photo section.



If the hash matches, the same seed is used to deterministically generate a public key and account address for the blockchain where PhotoKey is being used.

No private key or any other security information is ever stored in PhotoKey!



The public key and account address are reported to the calling application. The private key is only generated for signing requests and not available to the application.

At this point, you are probably curious about the level of entropy of the seed. Since we didn't have the requisite math background to calculate this accurately, very early during the development cycle, we posted a question on the Math Stackexchange about the number of selections possible using Combinatorics with ordering significance.

Home

Questions

Tags

Users

Unanswered

## Combinatorics with ordering significance

Asked 10 months ago   Active 10 months ago   Viewed 67 times

I am trying to write Javascript code to calculate the number of possible ways in which a user can make selections given containers  $A, B, C, D, E$ , each with 10 items numbered 1 through 10 (each container has the same items). I need help with the formula for making this calculation.

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>	<b>E</b>
--	----------	----------	----------	----------	----------

A user must select from at least 2 containers ( $s = 2$ ), but can select from all of them. The order of selection is important. For example, choices a user makes could be:

1.  $B_2, C_5$  ( $s = 2$ )
2.  $C_5, B_2$  ( $s = 2$ )
3.  $A_9, B_1, C_3$  ( $s = 3$ )
4.  $C_3, B_1, A_9$  ( $s = 3$ )

My understanding is that the standard way to calculate the number of variants is  $I^C$  (number of items ( $I$ ) raised to number of containers ( $C$ )), which in this case would be  $10^5$ . I am pretty sure this is incorrect because it doesn't account for the order of selection.

How can I calculate the number of possible user selections for different values of "s," "I" and "c"?


Update: To make this clearer, assume that once a user picks an item from a container it is closed and cannot be opened. A user can select one item and only one item from each container. They MUST select from 2 containers, but can select from more as they wish. The challenge here may not be obvious and is what I am struggling with – because the ORDER of selection matters, there are more possibilities than most well-known formulas will indicate.

combinatorics   permutations

share   cite   edit   delete   flag

edited Oct 19 '18 at 16:28

asked Oct 19 '18 at 1:32

 **Nik Kalyani**  
108 ▲ 5

Ref:

<https://math.stackexchange.com/questions/2961461/combinatorics-with-ordering-significance>

Based on the responses received, we were able to calculate the Entropy Bits for four Emojis at 56.86, which is 131 quadrillion permutations. We believe using four Emojis is a reasonable convenience to strength ratio for consumers using their PhotoKey for simple dapps, but will encourage them to use more Emojis to increase the Entropy Bits.

		Squares	Permutations	Entropy
Emojis	2841	9		
Choices	1	25569		
	2	22728	581,132,232	29.11429123
	3	19887	11,556,976,697,784	43.39382927
	4	11364	131,333,483,193,617,000	56.86601239
	5	8523	1,119,355,277,259,200,000,000	69.92315801
	6	5682	6,360,176,685,386,780,000,000,000	82.39534112
	Entropy Bits		Strength	
	Min	Max		
	0	27	Very Weak	
	28	35	Weak	
	36	59	Reasonable	
	60	127	Strong	
	128		Very Strong	

# UltraContract

As Creators achieve more success, their need for custom licensing contracts increases. In the music industry, it is very common for Creators to have custom licenses that give license holders specific rights that are further constrained by geography. Most licensing systems in Web2 are PDF-driven and digital signatures, while used, are rarely verifiable or used to digitally enforce licensing agreements.

UltraContract is Vitruveo's solution for bridging the two worlds. It enables the creation of custom legal licensing agreements that are not only signed and human-readable, but also deployed on-chain as Smart Contracts so the individual terms are consumable on-chain and therefore enforceable through automation.

The UltraContract system for automated legal contract generation is described using the following topics:

1. **Roles:** Parties involved in the transaction
2. **Workflow Stages:** Process steps for contract generation
3. **Language Manifest:** Data file format for contract generation
4. **Composition:** User interface for referenced document file selection and contract generation
5. **Document Generation:** Digital document generation
6. **Preview:** User interface and digital document preview of contract
7. **Contract Signing:** Cryptographic signing of the contract by parties
8. **Decentralized Publishing:** Publishing of referenced document files and digital document to decentralized file storage
9. **Smart Contract Generation:** Generation of source code of Smart Contract
10. **Smart Contract Deployment:** Compilation and deployment of Smart Contract to the public blockchain

## Roles

The primary roles in the system are “Licensor” and “Licensee.” There may be one or more Licensors and one or more Licensees. All aspects of the system that apply to a single entity in a role, also apply to a plurality of entities in that role. All subsequent references to Licensor and Licensee may be interpreted as equally applicable to singular or plural entities.

## Workflow Stages

### Workflow



There are four stages for contract creation:

1. Composition – Licensor uses the UltraContract user interface and the tools it provides to compose a legal contract and add any document files referenced in the contract.
2. Review – Licensee reviews the contract language and document files referenced in the contract. Licensee may comment on desired changes to any aspect of the contract. Licensor reviews these comments, makes any necessary changes and resubmits the contract for review to the Licensee.
3. Signing – Licensee cryptographically signs the contract.
4. Publishing – Licensor cryptographically signs the contract and publishes it. Publishing involves storing the referenced document files and a digital, human-readable preview of the contract to a decentralized file system, and

publishing a Smart Contract incorporating the contract terms to a decentralized ledger (blockchain).

## Language Manifest

The Language Manifest is a data file in which every possible section for every possible logical permutation of the text of a legal contract is stored as a separate entity, each of which may contain some or all of the following data items:

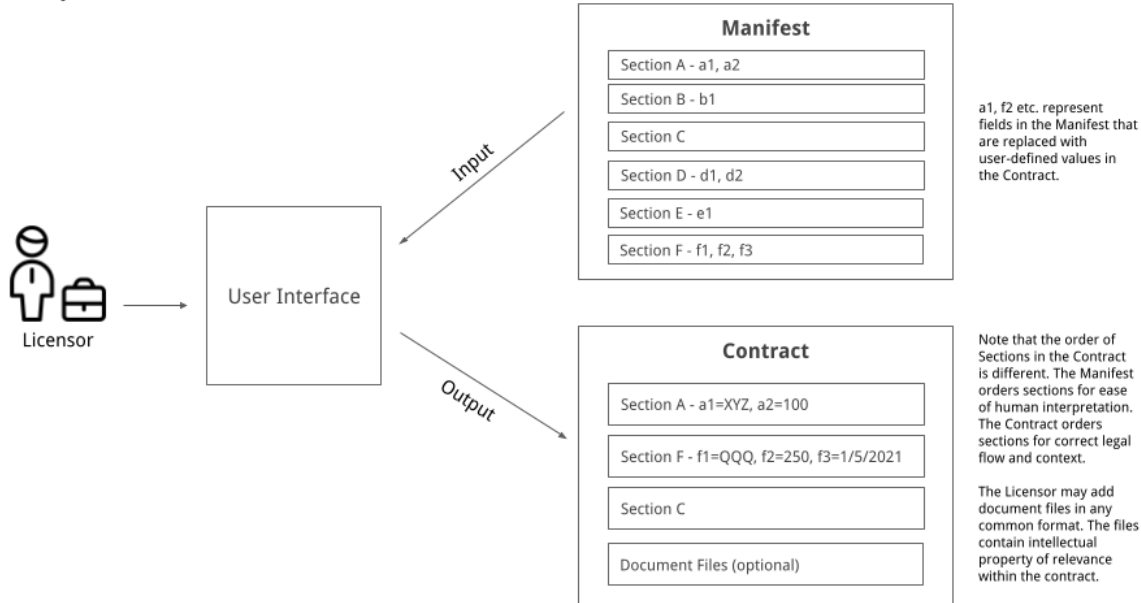
- a. **Name** – Internal, short, unique identifier for the section.
- b. **Title** – Human-friendly description of the section's legal text.
- c. **Category** – Human-friendly category under which the section is grouped.
- d. **Description** – Human-friendly descriptive text indicating the purpose and effect of including the section in the legal contract.
- e. **Relative Sequence** – Number indicating the order in which the section should appear relative to other sections. This is important because without it an automated generation of legal text that references prior or subsequent text would not be possible to automate. In addition, the logical flow of the document could not be automated (example: a signature block cannot appear in the middle of a document; terms of a financial transaction cannot appear before the definition of the financial transaction; etc.)
- f. **Optionality** – True or false value that determines if the section is required or optional.
- g. **Dependencies** – List of other sections on which this section depends. This is important because in legal language, including a clause may require additional required or optional clauses to be included.
- h. **Fields** – List of fields with name, type, title, description, validation criteria and encryption requirements. Each field in the list represents one item of data that will either be supplied by the user or programmatically generated.

- i. **Markup** – HTML formatted legal text for this section with hints for typeface styling, bullets, numbered lists, paragraphs and indents. A placeholder for each defined field is embedded within the formatted text for later replacement with user-provided or auto-generated values.
- j. **Preview** – Properly formatted markup text may appear in a bulleted or numbered list, or it may be indented. When the markup text for the section is displayed to the user in isolation, HTML formatting code included in the markup will be orphaned and impossible to display as it would in its final form. Preview markup defines “open” and “close” markup to simulate the appearance of the markup text for the section while the user is in the process of composing the legal contract.
- k. **Smart Contract Code** – Section-specific blockchain Smart Contract source code snippets with placeholders for field values. The snippets are composed into a functional Smart Contract during the publishing process.

## Composition

The process of creating a legal contract in UltraContract begins with Composition. During this workflow step, the Licensor makes choices from sections defined in the Manifest and composes them into a Contract. The contract sections are automatically sorted based on the Relative Sequence values defined in the Manifest. In addition, any dependent sections specified in the Manifest are automatically added to the Contract.

## Composition



For each section, the Manifest may have field definitions. For each field, the Licensor must input values of the required data type that match the validation criteria for the field.

The Licensor may optionally add one or more document files to the contract. These document files may be in any common file format. The files are intended to contain the Licensor's intellectual property that is being licensed through the contract, although this is not a requirement. They may also just contain content referenced in the contract.

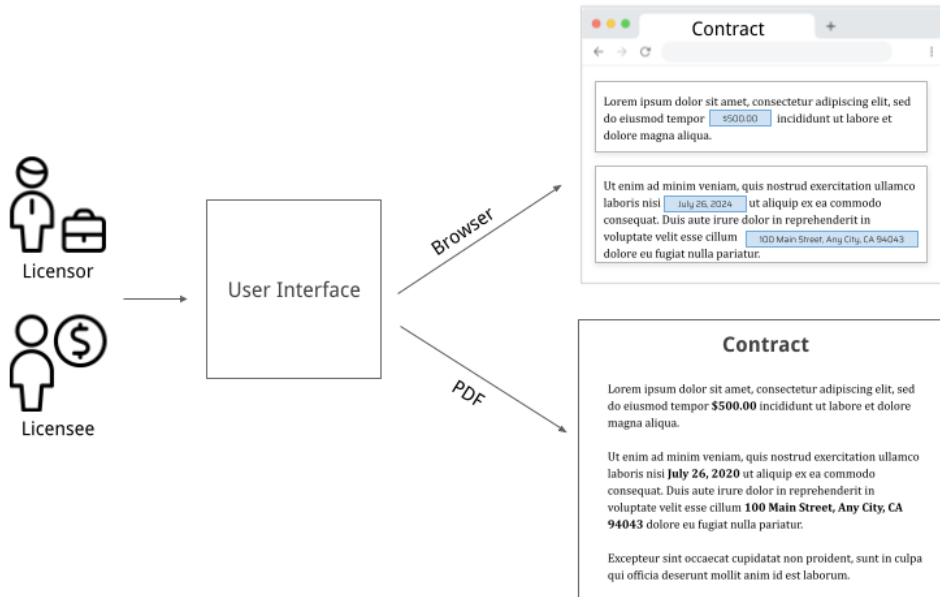
Once the Licensor has completed the Composition step, the contract is allowed to progress to the Review step of the workflow.

## Document Generation

In conformance with generally accepted norms for digital contracts, during each step of the workflow, a properly formatted PDF document containing the exact contract language is dynamically generated. The document includes references to any reference documents stored on decentralized storage and once signed, it includes hashes and signing account information (described in "Contract Signing").

# Preview

## Preview

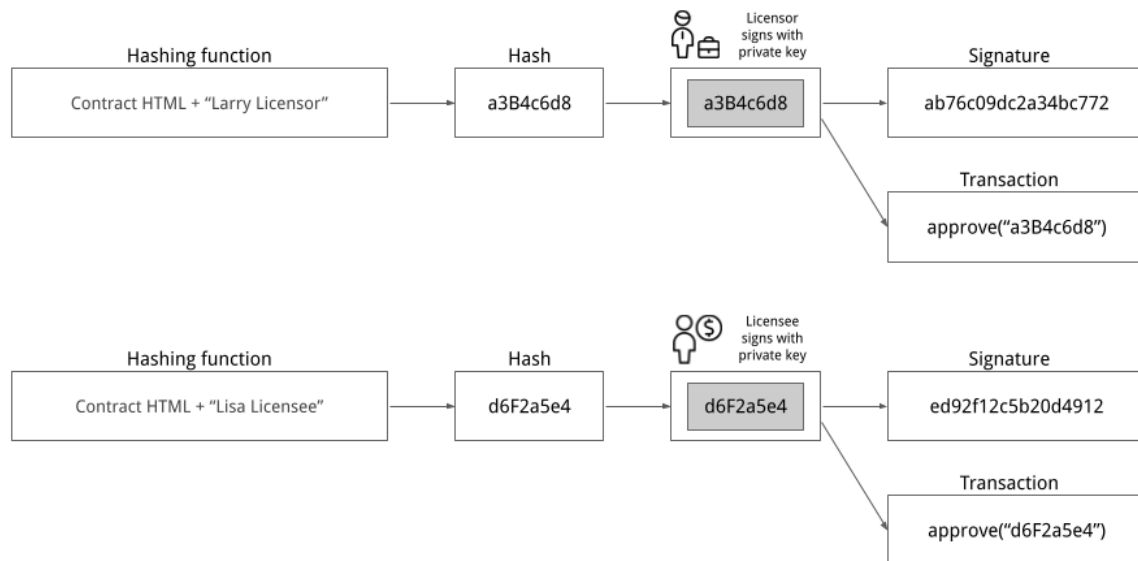


During the Composition workflow step, the Licensor can view a preview of the contract in the browser with full formatting such as bullets, indents, paragraphs etc. One important characteristic of the preview is that it shows the contract sections in correct legal order in contrast to the selection user interface which may show them in a more human-friendly order. In addition, during Composition, the Licensor can also view a fully formatted PDF preview of the contract.

During Review, the Licensee also has access to the browser and PDF previews.

# Contract Signing

## Signing



Licensors and Licensees cryptographically sign the contract using the following process:

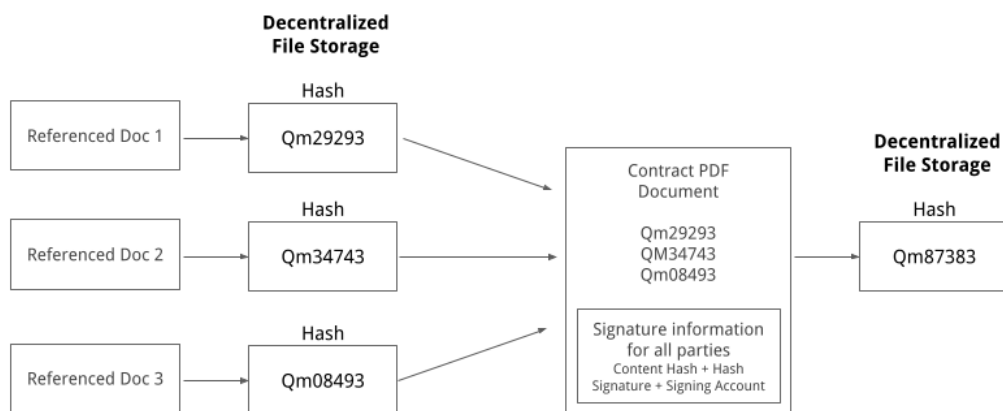
- The entire content of the contract in HTML format along with the signing party's name is cryptographically hashed using a [secure hashing algorithm](#) such as SHA3-256. The algorithm may change as cryptographic techniques improve over time. The resulting hash is a fixed-length text hash string H such that changing even a single byte value of the contract text would yield a different hash.
- Each signing party is in possession of a [private key \(PrK\) and public key \(PuK\)](#) associated with their account A on a decentralized ledger (blockchain) (example: the public Ethereum blockchain).
- Each signing party uses their private key (PrK) to cryptographically sign the content hash H resulting in signature S.
- Each signing party uses their private key (Prk) to cryptographically sign a Smart Contract transaction referencing the content hash H resulting in transaction

message M. This will be used during the “Publishing” step.

- e. For each signing party, H, A and S are visibly added to the signing block of the contract PDF. This information is cryptographic evidence of:
  - i. The content that was signed (H)
  - ii. The account (A) that signed the transaction
  - iii. Signature (S) of the hash that can be verified to be signed by (A)

## Decentralized Publishing

### Decentralized Publishing



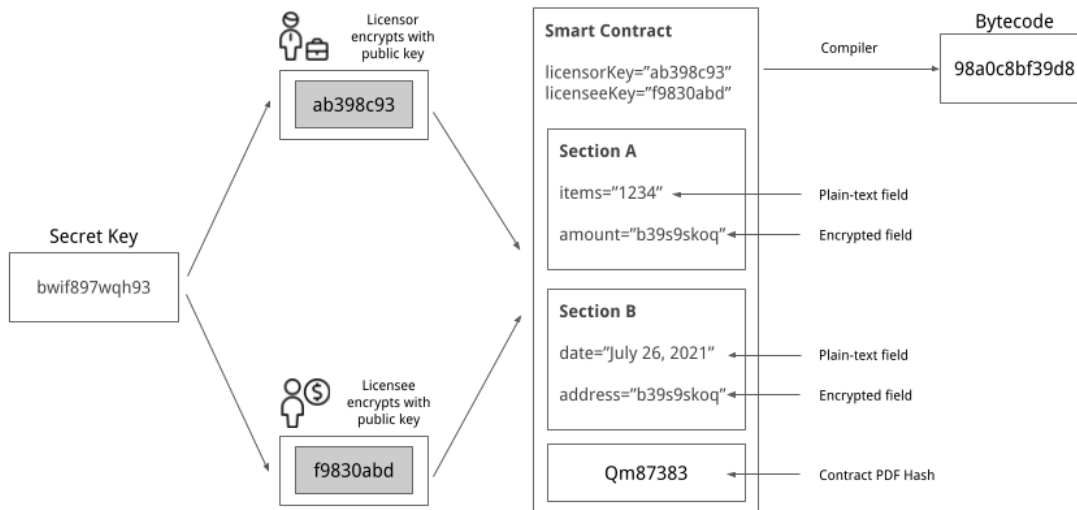
Publishing of files related to the contract to a decentralized file storage system such as IPFS (InterPlanetary File System) occurs in two steps.

Step 1: Any document files referenced in the contract are published to a decentralized file storage system as soon as they are uploaded to the user interface. For each file a unique file hash HF1, HF2 etc. are generated. A reference to each file hash is included in the contract PDF document.

Step 2: After signing, the contract PDF document is encrypted using a secret key Sk (see “Smart Contract Generation” for details of this key) and published to a decentralized file storage system to create a unique file hash HP.

# Smart Contract Generation

## Smart Contract Generation



As described in “Language Manifest,” each contract section may have associated Smart Contract code snippets in a language suitable for the blockchain in use for the system. In the case of Ethereum, the language is Solidity. After the contract is signed by all parties, the Smart Contract code snippets for individual sections are injected into a Smart Contract base template to compose a complete blockchain Smart Contract. This occurs in four steps.

**Step 1:** In this phase, a random secret key Sk is generated. This secret key is then encrypted with the public key PuK of each signing party resulting in an encrypted secret key EsK for each signer. The encrypted secret keys EsK are stored in the Smart Contract. Only the signer using their private key PrK can decrypt the encrypted secret key EsK to get Sk.

**Step 2:** As each section is added to the Smart Contract, the Manifest parameters for that section are enumerated. The section code is designed with placeholders for each parameter. These placeholders are replaced with the data value provided by the Licensor during Composition. If the Manifest specifies the data value should be encrypted, it is encrypted with Sk. The placeholder is then replaced with the encrypted

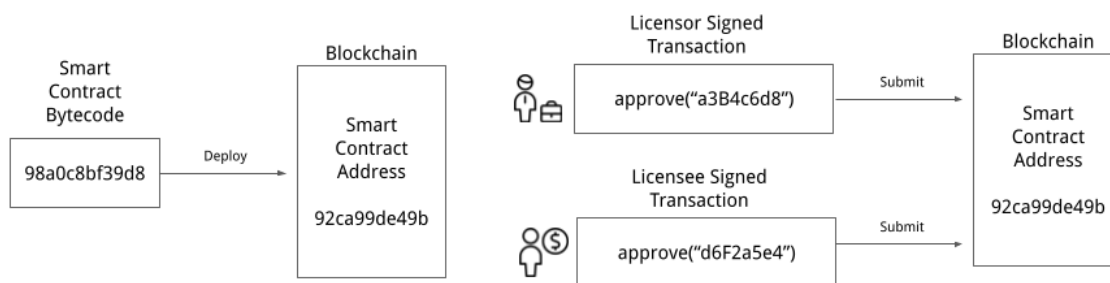
value. If no encryption is required, the placeholder is replaced with the plain-text value.

**Step 3:** A reference to the contract PDF document file hash HP is added to the Smart Contract.

**Step 4:** The Smart Contract code is compiled to produce bytecode ready for deployment to the blockchain.

## Smart Contract Publishing

### Smart Contract Publishing



The last step in the process is for the Smart Contract to be deployed to the blockchain. This is done in two steps:

**Step 1:** The Smart Contract bytecode is published to the blockchain resulting in a unique address.

**Step 2:** Once the Smart Contract is deployed, each of the signed hash transactions M from the signing step are transmitted to the blockchain. These transactions are in effect calling a function of the deployed Smart Contract as proof that the signing parties in fact did sign the content hash (H).

## Summary

A fully deployed UltraContract has the following proofs:

- 1) A PDF document immutably stored on decentralized storage containing signed hashes of the content by each signing party and reference decentralized storage hashes to all documents referenced in the contract.

**Proof Provided:** Each signing party used their secret private key to sign the content hash proving that they were aware of the content represented by the hash.

- 2) A Smart Contract containing encrypted and unencrypted field values for all sections of the contract.

**Proof Provided:** Public, immutable proof of the contract values.

- 3) A Smart Contract containing transactions with the content hash executed by the signing parties.

**Proof Provided:** Each signing party used their secret private key to sign a transaction and submit it to the blockchain proving their knowledge of the existence of the Smart Contract, its stored data values and their knowledge of the content hash.