zkme ZK KYC方案和技术总结

目录

[ZK KYC的背景和需求来源 1](#_Toc151470616)

[DID 2](#_Toc151470617)

[狭义的DID 2](#_Toc151470618)

[广义的DID 3](#_Toc151470619)

[verifiable credential/verifiable presentation 4](#_Toc151470620)

[几个关键角色 4](#_Toc151470621)

[数据格式 5](#_Toc151470622)

[VC 5](#_Toc151470623)

[VP 7](#_Toc151470624)

[ZK KYC 9](#_Toc151470625)

[ZK KYC技术栈 9](#_Toc151470626)

[ZK KYC业务流程 10](#_Toc151470627)

[具体步骤如下： 11](#_Toc151470628)

[ZK KYC过程涉及到的角色 18](#_Toc151470629)

[ZKP,SBT和用户私钥的解耦 18](#_Toc151470630)

[用户使用的页面汇总 19](#_Toc151470631)

[身份验证过程中dapp H5页面和zkMe-widget H5页面的交互过程 20](#_Toc151470632)

[用户身份验证结果的存储和复用 20](#_Toc151470633)

[用户身份明文的存储和解密 21](#_Toc151470634)

[身份验证的具体验证内容 21](#_Toc151470635)

[初始在哪条链铸造SBT&多链支持 23](#_Toc151470636)

[人脸faceprint的隐私 23](#_Toc151470637)

[身份证件源头造假的问题 23](#_Toc151470638)

[总体架构 23](#_Toc151470639)

[dapp和zkme的集成 25](#_Toc151470640)

[零知识证明技术栈 26](#_Toc151470641)

[零知识证明技术发展的历史脉络 27](#_Toc151470642)

[零知识证明的技术流派 27](#_Toc151470643)

[SNARK 28](#_Toc151470644)

[groth16 28](#_Toc151470645)

[零知识证明在ZK KYC中的使用 32](#_Toc151470646)

# ZK KYC的背景和需求来源

近一两年来，各国都开始对加密领域实施监管，制定相关的监管政策、框架和法规，比如欧盟颁布了一项新法律《the Markets in Crypto-Assets (MiCA)》，于2023.5.31正式生效，美国的SEC和CFTC也在对加密领域进行监管，美国的监管形势目前比较复杂且正在发展当中。另外，各国的监管都在遵循国际合作组织Financial Action Task Force (FATF)的要求。从各国的监管措施来看，加密领域的监管有趋紧的趋势，未来对于web3项目来说合规将变得越来越重要。

各国的监管框架大同小异，监管要求主要集中在如下几方面：

1. **注册和牌照**：要求加密企业在相关政府部门注册，并且获得牌照。
2. **反洗钱，反恐怖主义融资**：要求加密企业对用户进行KYC, KYB, KYT(know your transaction)，用户交易数据保留至少5年，在遇到可疑行为时上报给相关监管部门并配合监管部门执法等等。
3. **crypto travel rule**：旅行规则要求加密公司从加密资产的发送方和接收方获取信息，并与交易对手加密资产服务提供商共享。个人理解是只要用户在一家交易所做了KYC，交易所需要把KYC信息共享给其他交易所，那么全部交易所就都知道了客户的KYC信息。
4. **稳定币监管法规**：一些国家已经制定或正在制定的过程中，具体情况未深入了解，和本文的主题ZK KYC关系不大。

关于各国监管的更详细的情况，可参考如下链接：

<https://docs.zk.me/zkme-dochub/zkkyc-compliance-suite/zkkyc-introduction/eu-mica-tfr-regulations>

<https://docs.zk.me/zkme-dochub/zkkyc-compliance-suite/zkkyc-introduction/uk-crypto-regulations>

<https://docs.zk.me/zkme-dochub/zkkyc-compliance-suite/zkkyc-introduction/us-crypto-regulations>

以上说明了**监管的趋紧使得KYC会越来越重要**。

然而，在传统的KYC中，用户的个人信息是明文传输和验证的，用户的个人隐私数据实际上泄露给了企业，企业或者企业做KYC的员工又可能把用户的数据进一步泄露或贩卖给其他实体，造成用户的隐私数据大范围泄露。然而，在加密领域中，去中心化和匿名性是web3的核心原则和普遍认同的价值观，至少很多用户是不愿意在访问web3项目（比如DEX）时暴露自己的真实身份的，即不希望自己的隐私数据被获取和泄露。所以，如果还是按照传统的方式进行KYC，在加密领域可能是不受欢迎的。

以上说明了**传统的KYC和web3底层价值观之间的冲突**。

那么有没有一种技术，可以在对用户做KYC满足合规的同时，又不泄露用户的隐私呢？答案是肯定的，我们称之为**ZK KYC**，ZK代表**零知识证明技术**，其底层技术就是DID, verifiable credential/verifiable presentation, zero knowledge(SNARK)技术的组合。

简单一句话，ZK KYC可以实现在不泄露用户隐私的前提下，验证用户的身份是否合法。比如，在验证用户是否已经年满18岁的同时不泄露用户的生日，在验证用户是否来自美国的同时不泄露用户的具体地址信息。简而言之，就是判断一个statement，得到true/false的答案。

同时，值得注意的是，美国财政部在2023年4月的报告《[Illicit Finance Risk Assessment of Decentralized Finance](https://home.treasury.gov/system/files/136/DeFi-Risk-Full-Review.pdf)》中提到：

*Zero-knowledge proofs can also enable a DeFi service user to confirm that their identity has been verified without revealing personal information.*

*the U.S. government supports privacy enhancing technologies that simultaneously allow for or even promote compliance with AML/CFT obligations.*

从以上的表述中，可以看出美国政府也支持用零知识证明等隐私增强技术对用户身份进行验证。

以上说明了**不泄露隐私的KYC方式是存在的，且被美国政府支持**。

上面介绍了为什么要做ZK KYC，接下来继续说明ZK KYC的具体技术实现，尽量用大白话的风格对这些听起来高深的技术进行去魅式的讲述。

在介绍zkMe ZK KYC的具体架构和流程之前，首先介绍一下DID和verifiable credential/verifiable presentation的概念。

# DID

DID，是一个 Web3 领域的热门概念。在 Twitter 上，几乎每周都有讨论 DID 的 Twitter Space；在线下的各种 Web3 分享会中，DID 也是经久不衰的热门主题之一；在项目的融资 deck 上，无论是社交、GameFi、DeFi、NFT 等应用类项目，还是钱包、域名甚至公链等 infra/ 中间件类项目，都可能会把 DID 加入其叙事之中。如此高的热度，不免的让 DID 这个词被泛用、甚至被滥用。

DID在不同的语境和上下文中，代表不同的意义。简单来说，DID有**狭义**和**广义**的定义。

## 狭义的DID

狭义的DID指的是**Decentralized Indentifiers**，是W3C出于对 Web2 中心化身份体系的担忧，而牵头制定的一套标准。这个 DID 的概念，一开始和区块链/Web3 其实没有直接的相关性，其可以被应用在公链中，也可以被应用在联盟链中（代表项目是微众FISCO BCOS区块链的WeIdentity项目，其对DID的认知很深入）。

DID的W3C标准如下：

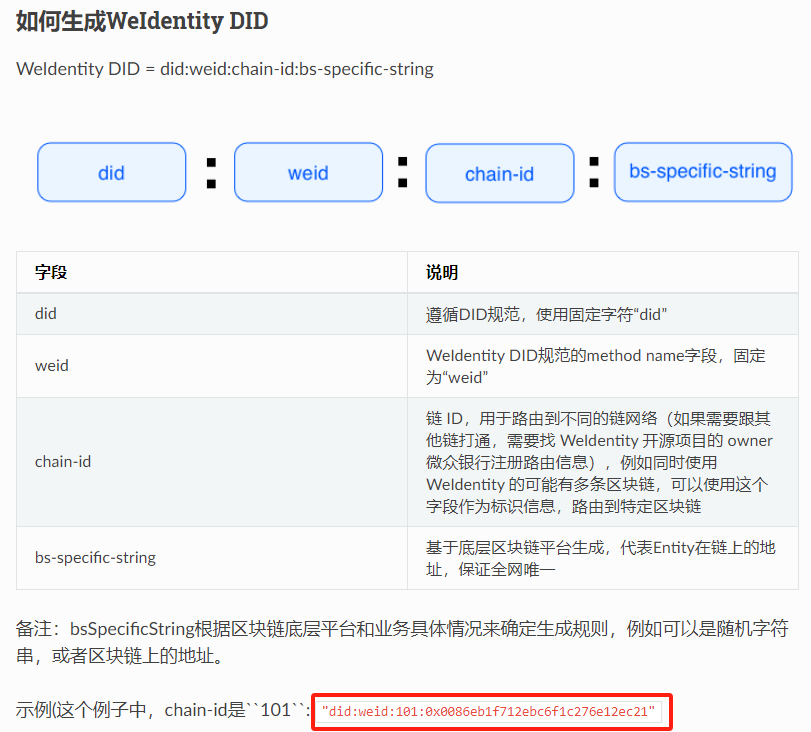
<https://www.w3.org/TR/did-core/>

对于绝大部分非技术专业人士来说，并没有必要阅读DID W3C标准，只需要对DID有个最简单的理解即可，可以把DID理解为分配给一个人或物的一个唯一的ID，这个“唯一”也取决于具体的上下文，比如在一个联盟链中，这个“唯一”就仅限于这条联盟链，在以太坊中，这个“唯一”就仅限于以太坊这条链。

W3C中定义的DID格式如下：



看一个WeIdentity中实际的例子（虽然是联盟链，但很有启发性）：



以上例子中，did和weid是写死的，101代表链id，0x开头的代表用户地址。

对于公链可以如法炮制，比如我们可以定义：

did:evm:1: 0x0086eb1f712ebc6f1c276e12ec21 代表chainid为1的evm链（即以太坊主网）的一个用户。

did:evm:42161: 0x0086eb1f712ebc6f1c276e12ec21 代表chainid为42161的evm链（即arbitrum）的一个用户。

所以，**就简单把狭义的DID理解为分配给一个用户的唯一ID而已，没有更复杂的含义**。

值得注意的是，**在zkMe项目中，DID就是狭义DID的概念**。

## 广义的DID

在现在的 Web3 交流中，DID 更多时候被看作是“**Decentralized Identity**”的简称，也就是泛指“去中心化（数字）身份”。然而，去中心化身份本身也是一个**缺乏明确定义**的词汇，虽然初看每个人都能理解它大概的意思，但在不同场景下具体指的事情可能很不一样；而且在 Web3 的世界里，似乎做什么事情都能和它扯上关系。这也是为什么目前有关 DID 的讨论中概念较为混乱的原因。

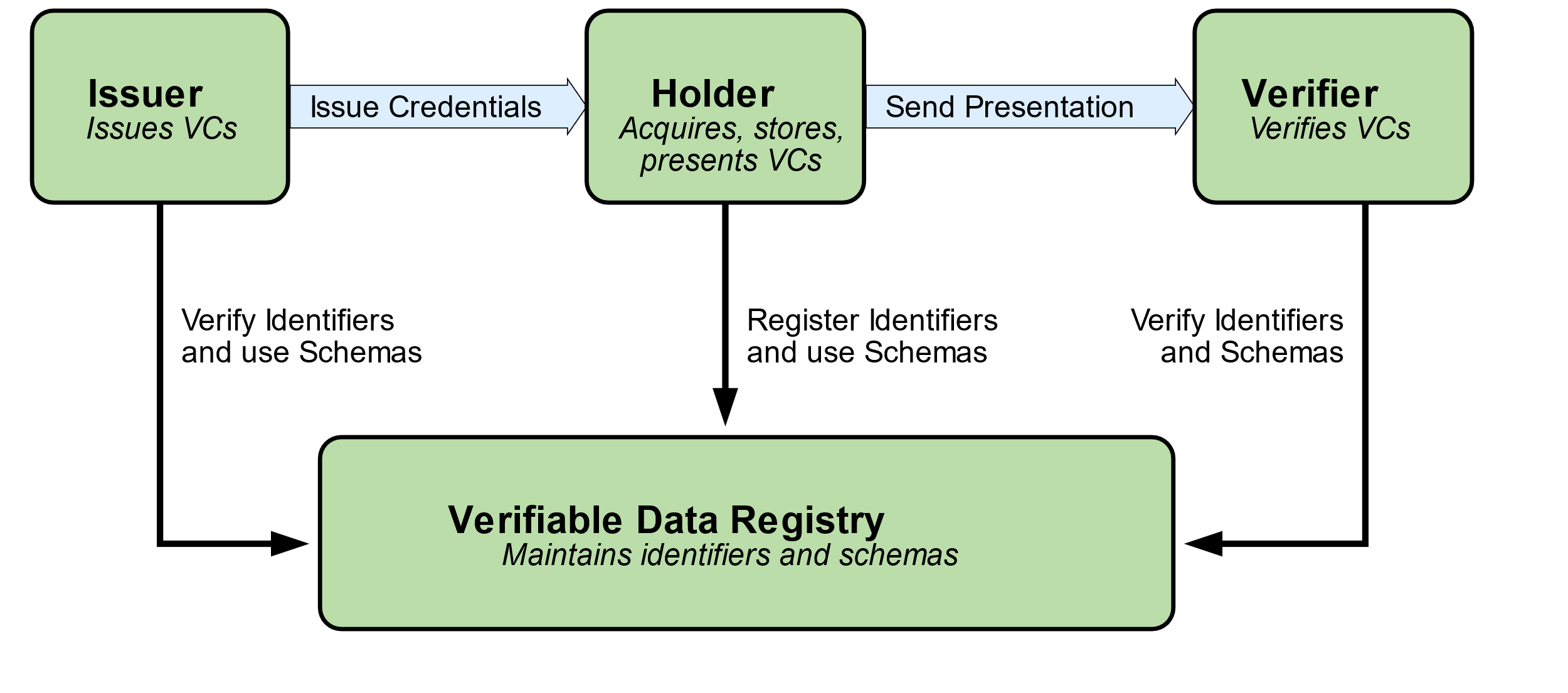
广义的DID在此文就不做进一步阐述了，因为偏离主题了，且过于宽泛。具体的可以参考以下文章：

<https://foresightnews.pro/article/detail/15908>

# verifiable credential/verifiable presentation

简称VC和VP，下文就以VC和VP代称。VC和VP其实**比较复杂**，比狭义的DID复杂，细节很多，此文就简单介绍其核心概念。

首先看一张角色图：



## 几个关键角色

**Issuer**: VC（可验证凭证）的颁发者，比如，公安局可以是一个issuer，向一个公民颁发一张身份证（VC），这个公民就是这个VC的holder。

从技术上讲，任何人或机构都可以颁发VC，但值得注意的是，issuer和holder都有公私钥，issuer需要对颁发的VC用私钥进行签名，以证明VC是自己颁发的，所以在上述的例子中，如果一个造假证的人颁发了一张身份证VC，其实是不被verifier认可的，因为verifier会验证VC的签名，也就是验证颁发者的公钥是否是公安局的公钥，如果不是，就认为这个VC是非法的、不被接受的，所以，只有公安局颁发的VC才能通过验证。

从业务的角度讲，issuer又可以被划分为权威issuer和非权威issuer，上面例子中的公安局就是权威issuer。

zkMe中的issuer其实并不是现实意义中的权威issuer，比如政府部门，而是zkme APP中的程序，对用户的KYC通过之后自动颁发VC，细节此处不讲，放在zkMe的章节中介绍。

上述表述中，为了让读者更容易理解，采用了“verifier验证VC”的表述，实际上是不准确的，应该是“verifier验证VP”，verifier验证的是VP，而不是VC。VP是由VC生成的，后面再详细介绍。

**holder**:即VC和VP的拥有者，比如一个公民，拥有一张身份证或者护照，这个公民就是holder，这很容易理解。当然，VC的主体（subject）和holder不一定是同一个人或物，比如，检疫局给一只狗颁发了一张防疫证明，那么这张防疫证明的对象是狗，即主体是狗，但狗的主人替狗持有了这张防疫证明，所以holder是狗主人，subject是狗。这涉及到VC的字段，此处不细讲，读者先暂时了解到这个程度即可。

**verifier**:VC的验证者，准确来讲，应该是VP的验证者。verifier可以是任何人或机构，比如一个网吧管理员，想验证一个客户是否年满18岁，其本质就是对VP的内容进行验证。VP的内容可以是明文的，比如VP内容可以包含holder的身份证信息明文，这种情况网吧管理员（verifier）在判断客户年龄的同时，也能获取客户的隐私数据；VP的内容可以是非明文的，不包含holder的身份证信息明文，而是包含零知识证明，比如包含一条statement，断言holder是否成年，即”is older than 18”，答案是true，这种情况网吧管理员（verifier）也能可靠地判断holder已经成年，但是无法获取holder的确切生日，所以隐私没有泄露。至于零知识证明的技术原理，后面再介绍。

**verifiable data registry**: 是一个公共数据库，用于存储各个角色（issuer, holder ,verifier）的DID信息，以及VC的schema信息。在web3的上下文中，verifiable data registry通常就是区块链，比如以太坊，也就是把DID和schema这些公开信息存储在区块链上。打个比方，把权威issuer的DID存储在链上，这样verifier在验证holder的VP的时候，就能通过比对公钥，验证VP是否是由权威issuer签名的。

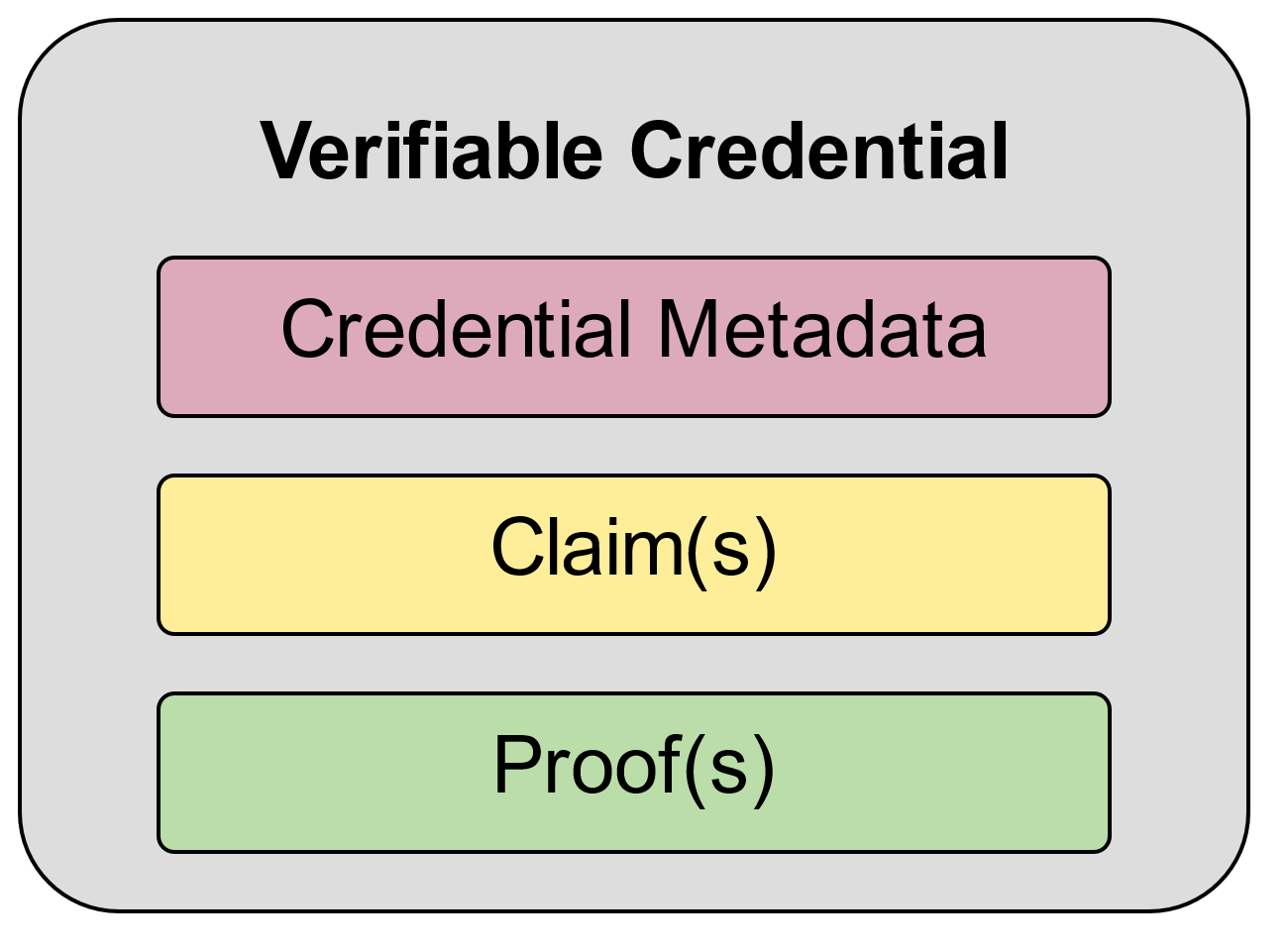
## 数据格式

接下来介绍一下VC和VP的数据格式，直接看例子：

### VC

VC由issuer颁发给holder之后，可以由holder自己持有，存储在本地，也可以存储在某个值得信任的第三方，主要取决于holder意愿和系统设计。如果由holder自己持有，那么只有一份，没有copy。

VC的粗略结构如下，包含三部分内容：



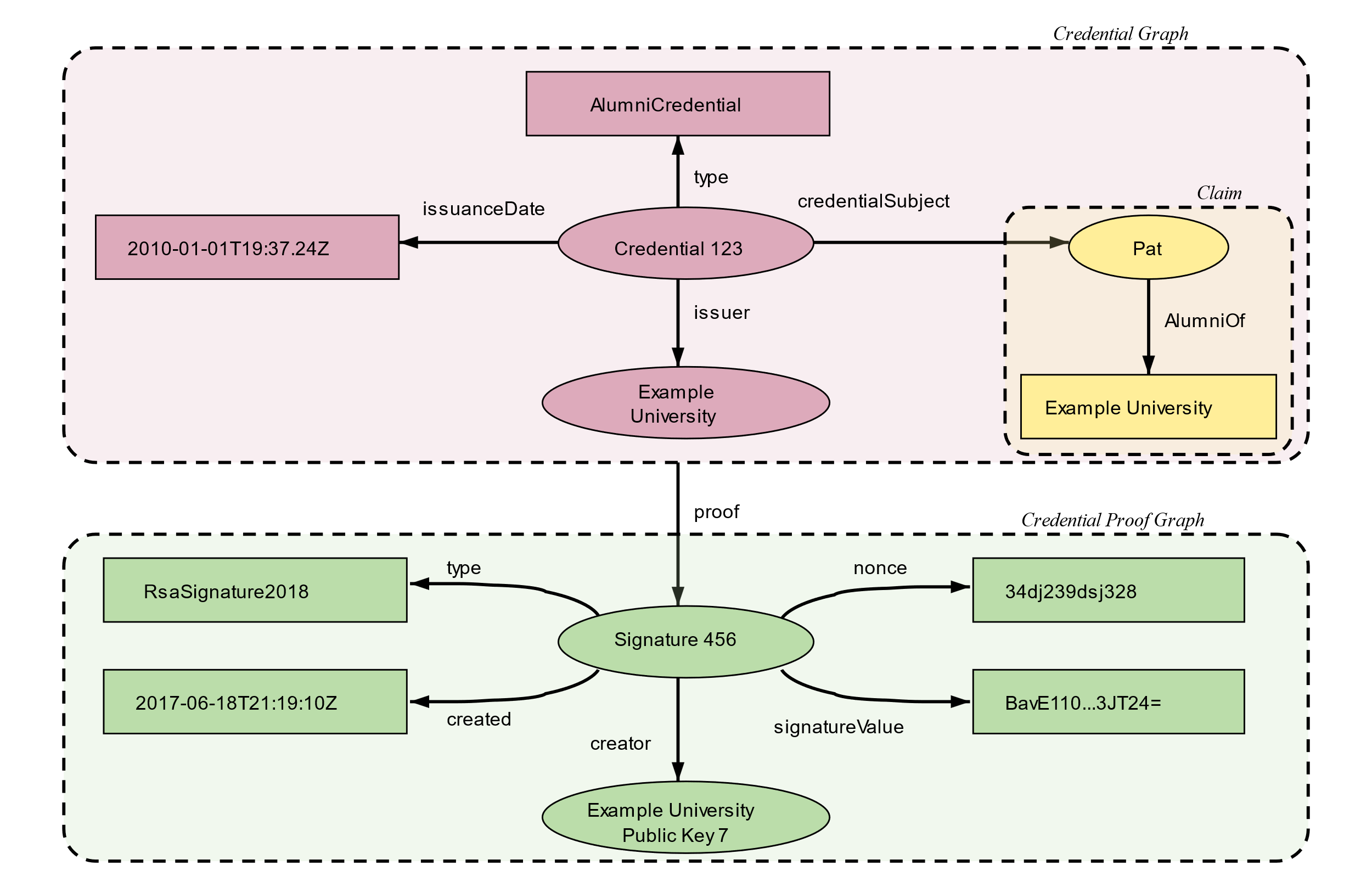
**credential metadata**: 主要是一些元数据，比如VC的id, type, issuer, issuanceDate，这些内容是每一个VC都需要具备的，是通用的。

**claims**: VC的数据主体，就是一些明文的声明，技术上就是“属性：属性值”，比如”birthdate=1990.1.1”, ”name=zhang san”。注意，VC中的claims都是明文的原始信息。

另外，claims得满足一定的schema，比如身份证claims和毕业证claims的属性定义是不同的，即schema不同。

**proofs**: issuer对VC的签名，用于确保VC的完整性、真实性、不可抵赖性、鉴权，verifier可以对签名进行验证，判断是否是合法的issuer颁发的VC。

一个VC示例的结构如下：



上图是一个校友VC，红色部分是credential metadata, 黄色部分是claims，只有一条声明，Pat是某某大学的校友，绿色部分是proof，即大学对VC的签名信息，包括签名算法、签名值、签名公钥等等。

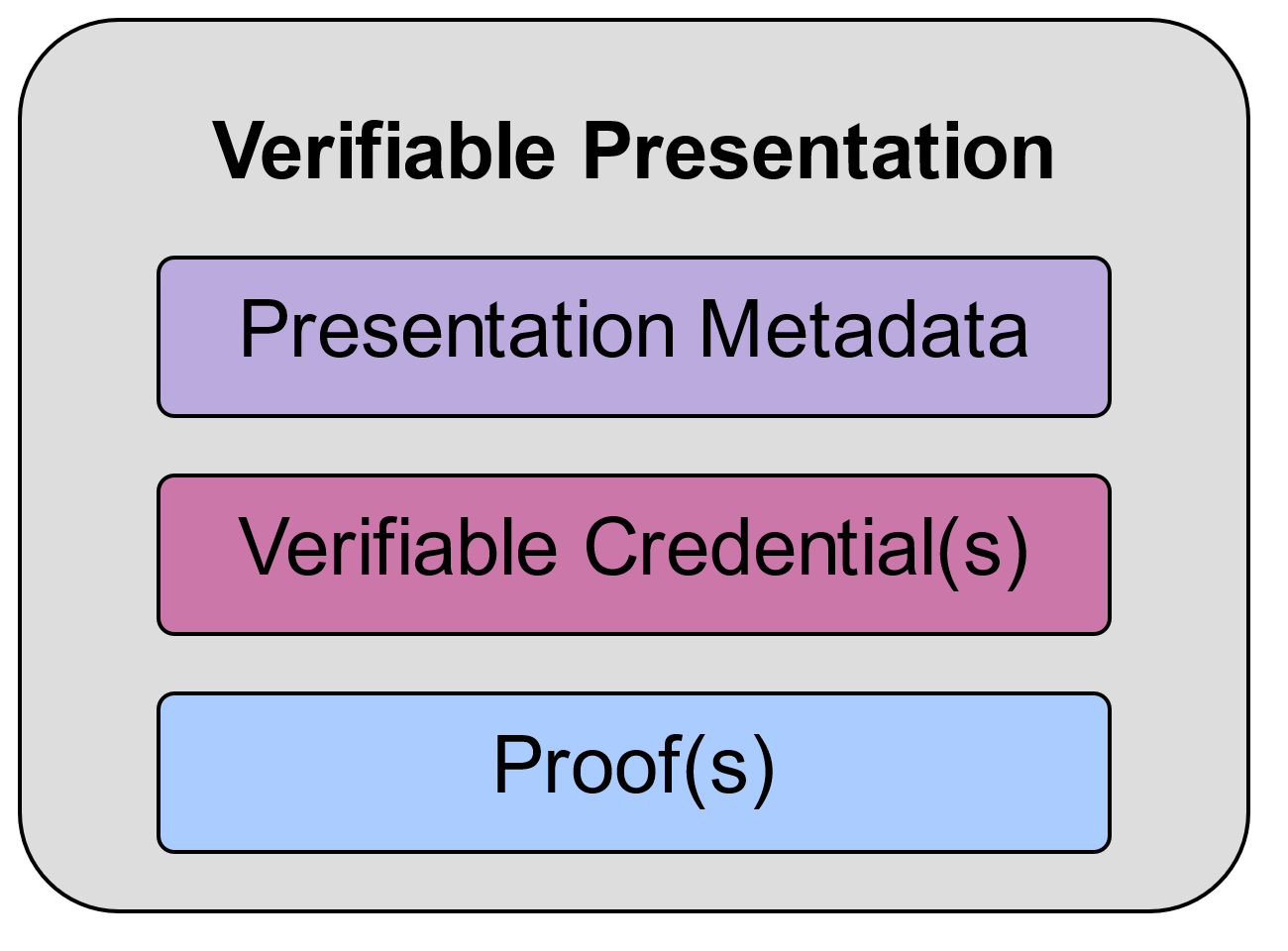
VC通常是json格式，以下是一个示例：



### VP

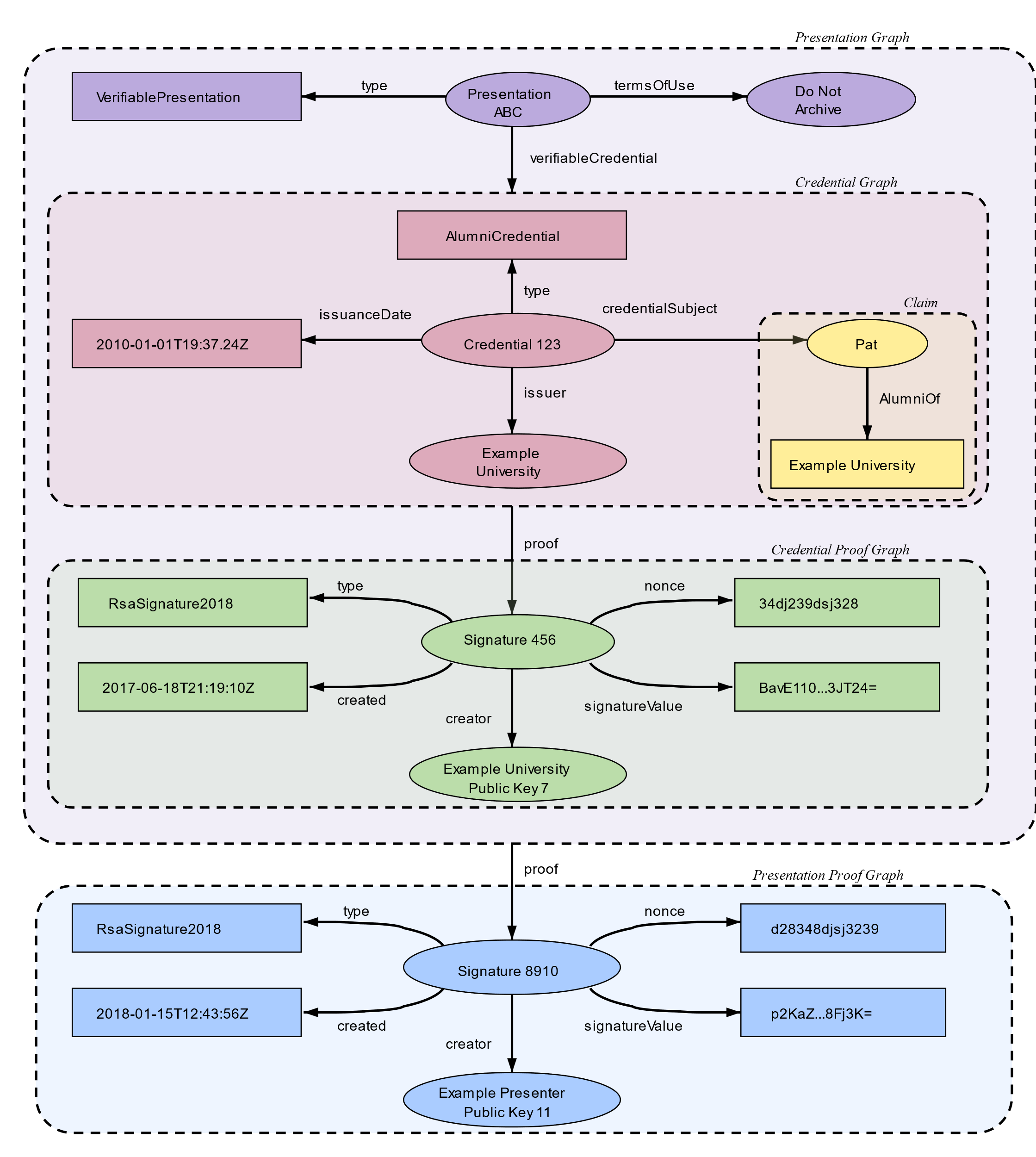
VC是凭证的明文，holder持有，当verifier想验证holder的VC时，holder展示给verifier的并不是VC，而是VP。VP是由VC生成的。

VP的粗略结构如下：



包含**元数据，VC, 签名**三部分。

一个VP示例的结构如下：



从上图可以看出，VP是对VC的进一步封装，包含了VC的内容，然后附加了下面蓝色的presenter签名的部分，即holder签名的部分。这样，VP就包含了两个签名，一个是issuer的签名，一个是presenter的签名。

上图的VP示例表示的是明文VP，而非零知识VP。**零知识VP中不会包含VC的明文，而会包含零知识claim，即statement，所以，verifier在验证VP的时候验证的是其包含的零知识claim，而不是原始信息。**

以上简要介绍了W3C VC/VP的概念，更详细的可以参考：

<https://www.w3.org/TR/vc-data-model/>

W3C VC/VP是协议/标准，其具体实现有一些开源库，此处不赘述。

zkMe的ZK KYC采用了W3C VC/VP，接下来继续介绍zkMe的ZK KYC的技术实现，zkMe文档并没有介绍得很清晰，很多地方融入了我个人的思考。

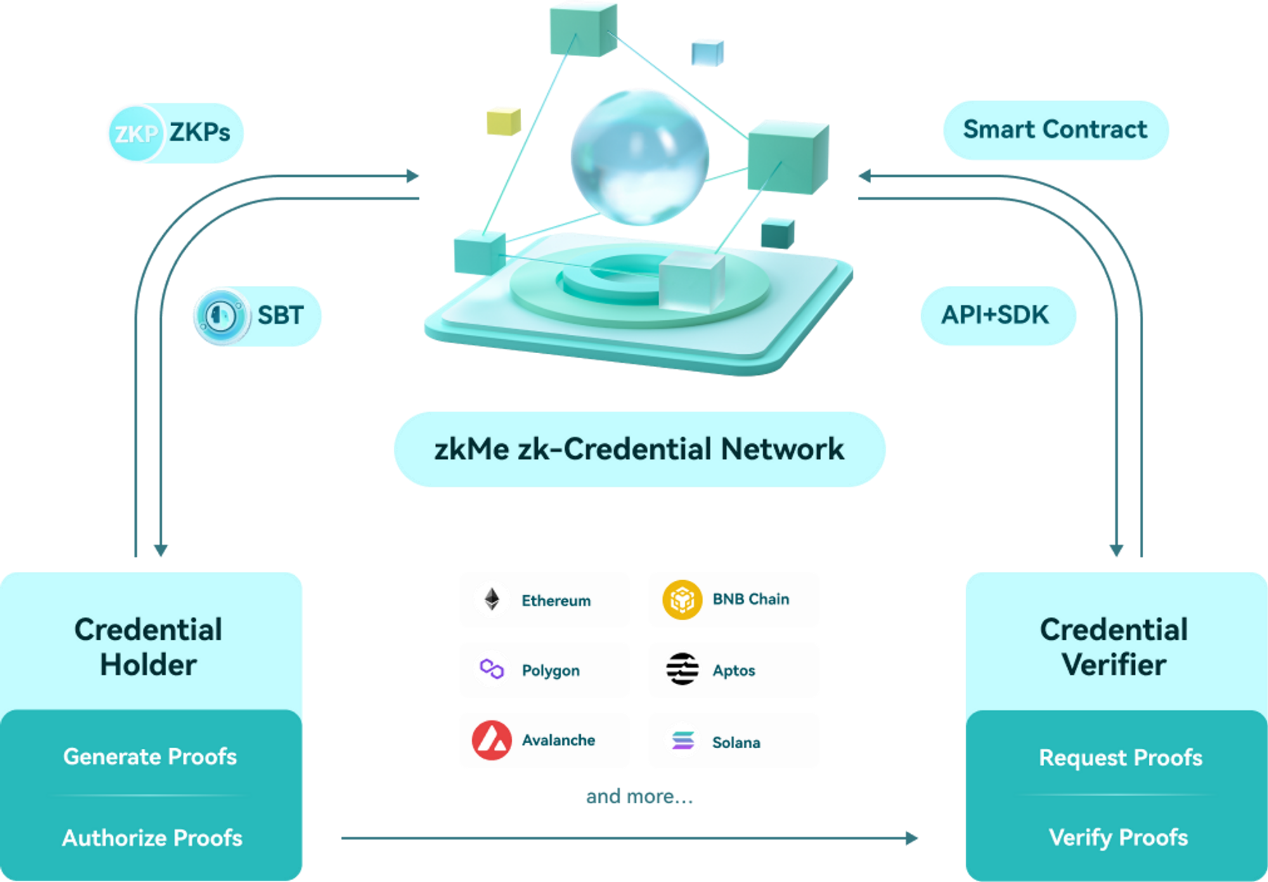
# ZK KYC

## ZK KYC技术栈

粗略来讲，ZK KYC采用的技术主要包括OCR，人脸识别，VC/VP，智能合约，SBT（灵魂绑定代币），零知识证明（groth16）。

技术栈如下：

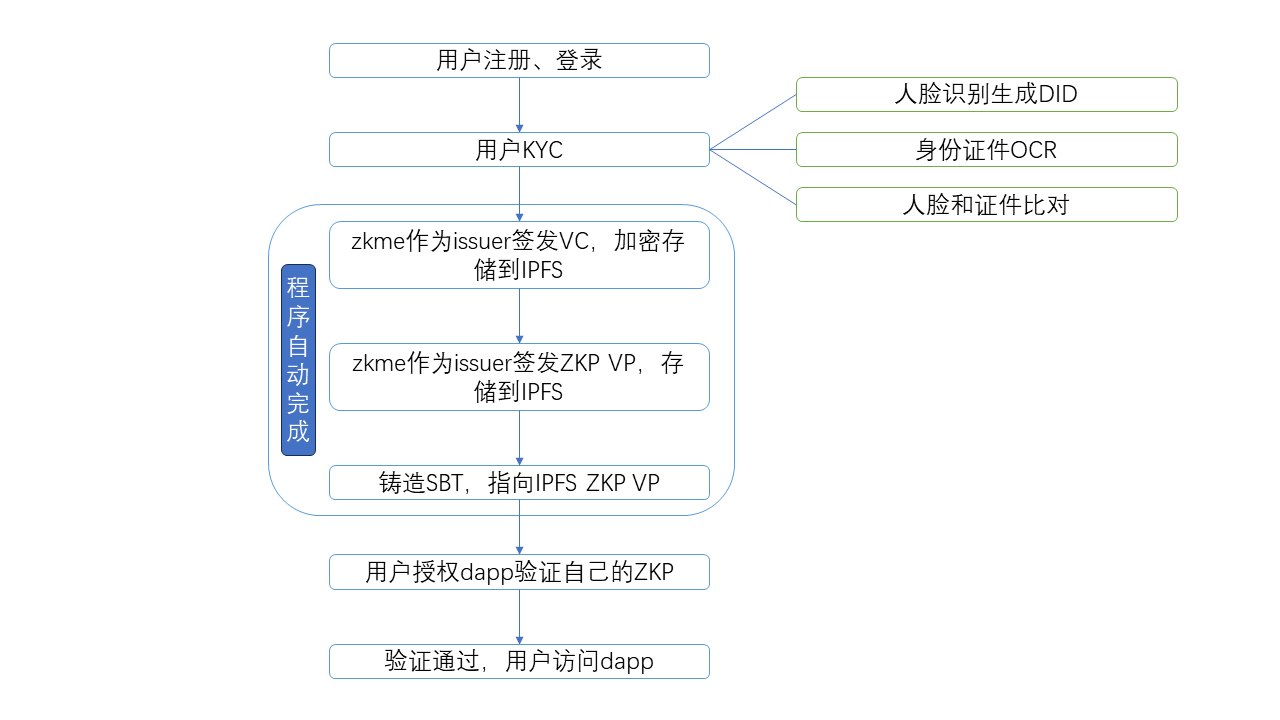




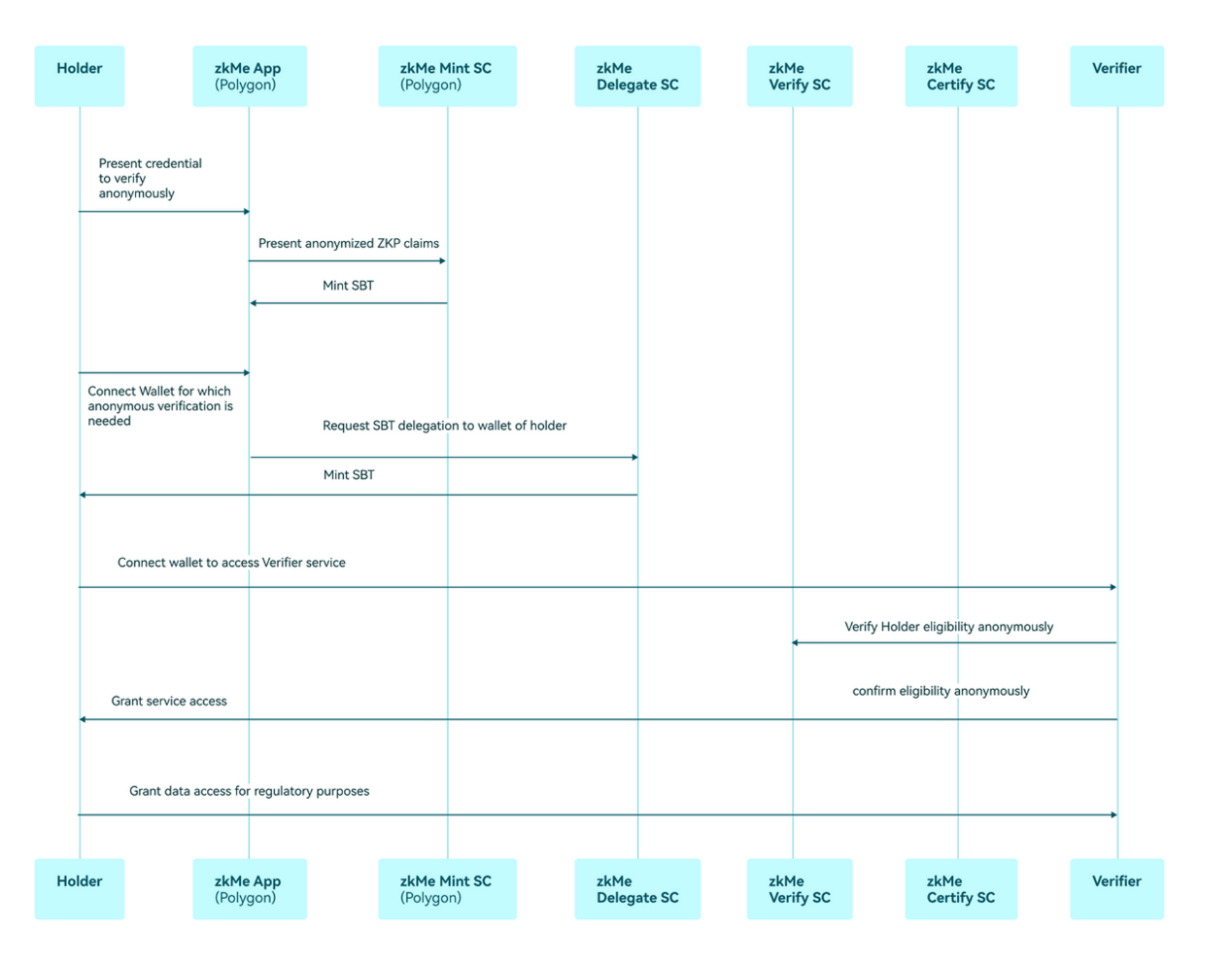
## ZK KYC业务流程

站在**终端用户（比如一个DeFi用户）的使用角度**，ZK KYC的业务流程如下：

流程图：



时序图：

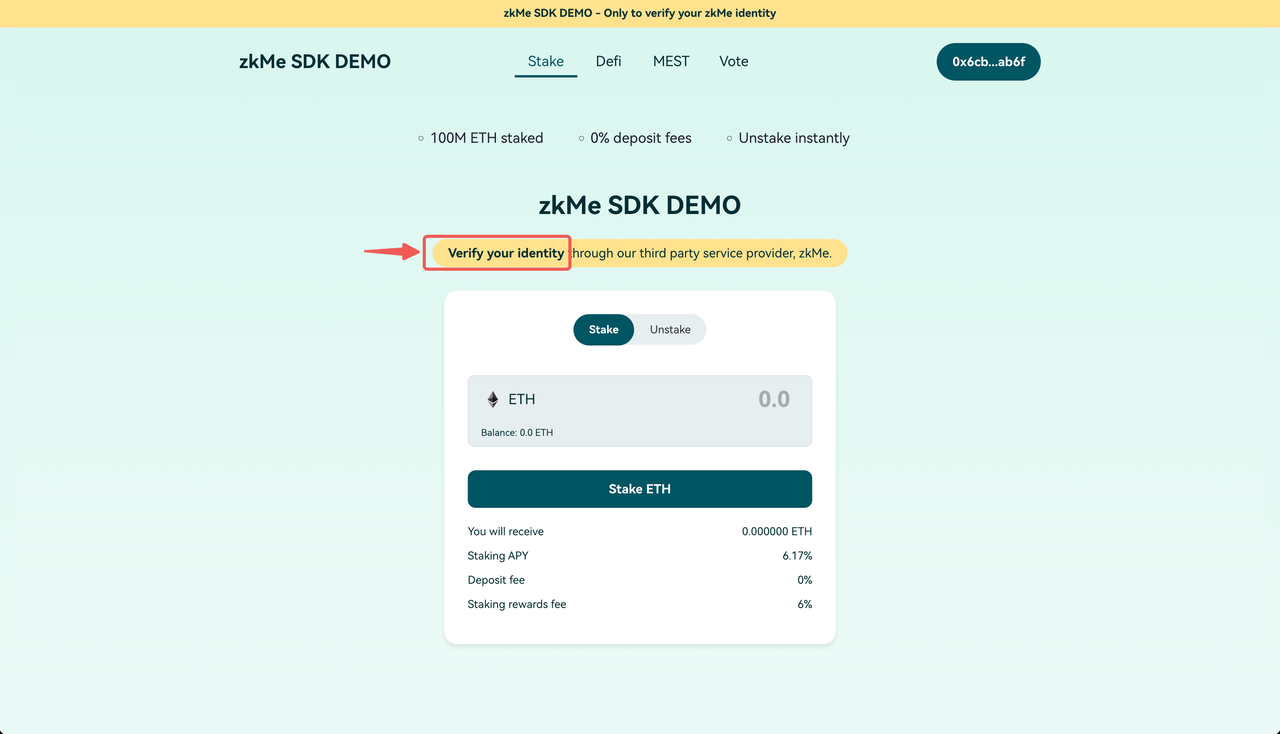


上面是一个粗略的流程图和时序图，有助于了解大概的流程，其中zkMe App也可替换为zkMe H5页面，都负责在本地生成ZKP，并触发ZKP的IPFS存储和SBT的链上铸造。

### 具体步骤如下：

1. **在PC端登录：**

假设用户想和一个DeFi项目交互，用户在PC端进入一个该项目的页面，由于该项目和zkMe ZK KYC已经做了集成，用户在没有通过KYC的情况下是不能直接和项目进行交互的，所以用户会看到一条提示，提示其验证身份，做KYC，如下：

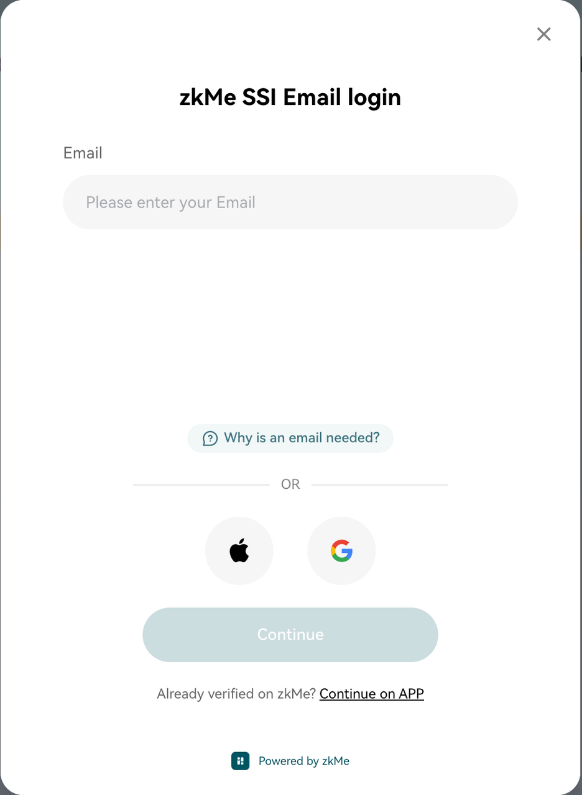
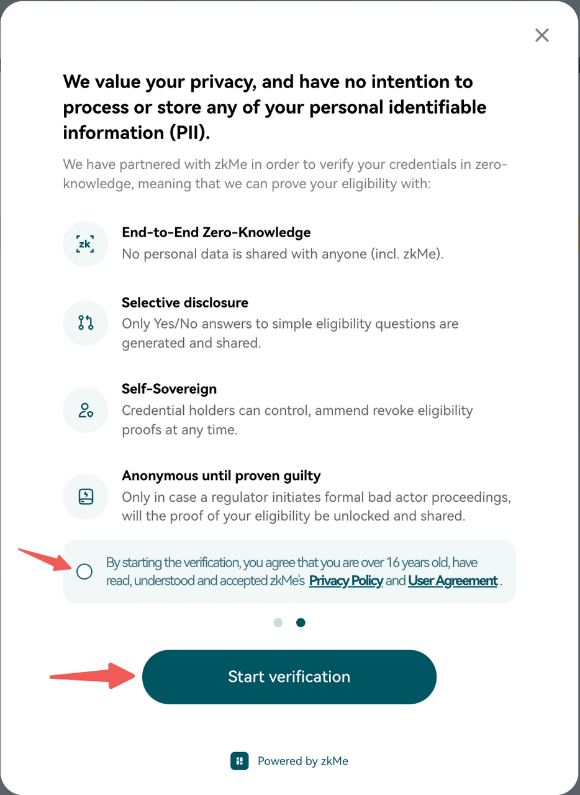


用户点击之后，就会弹出一个zkme的widget，也就是说这个DeFi项目集成了zkme作为KYC服务提供商。由于用户从未使用过zkme，处于未登录状态，首先会提示用户进行注册和登录，用户输入邮箱地址，邮箱收到一条登录链接，用户点击登录链接，登录成功，此时处于已登录状态。除用邮箱登录之外，还可以使用apple ID和Google account进行第三方OAuth2.0登录。

注：widget的地址是

<https://widget.zk.me/?mchNo=YOUR_MERCHANT_NUMBER&name=YOUR_PROJECT_NAME&chainId=HEX_CHAIN_ID&accessToken=ACCESS_TOKEN>

这个zkme的页面通过iframe widget的方式嵌入dapp页面。

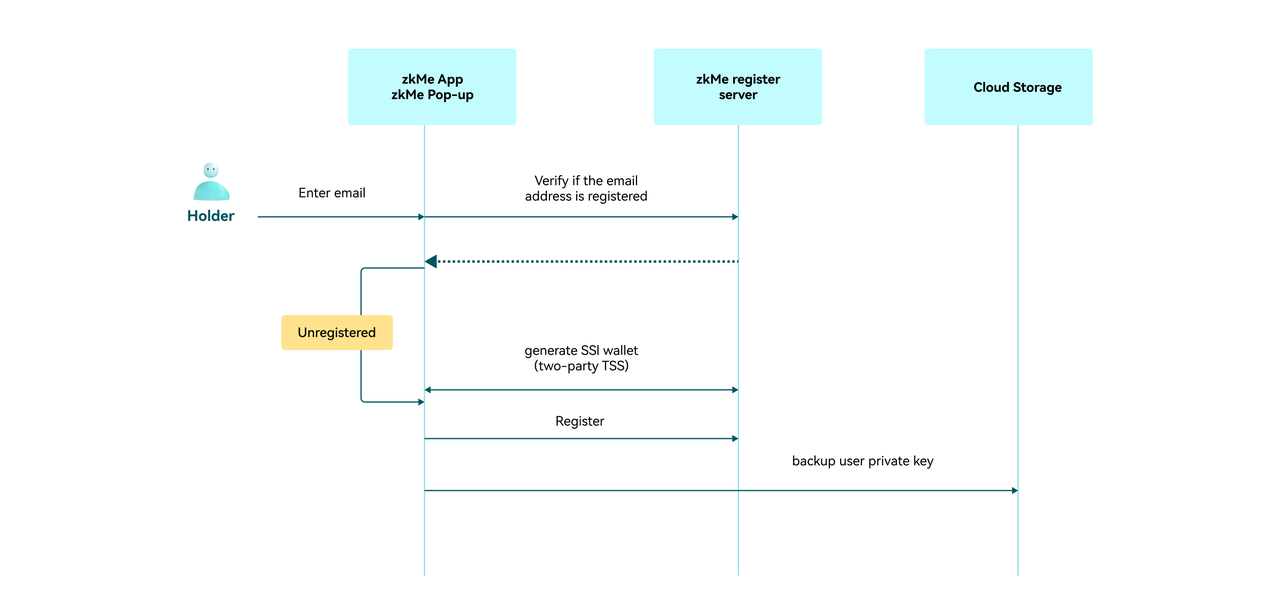


这一步有两点值得注意：

一是注册和登录采用了传统的中心化注册登录，由APP和后台中心化服务通信完成，和区块链还没有关系。

二是匿名性，邮箱可以是未经过实名认证的邮箱。

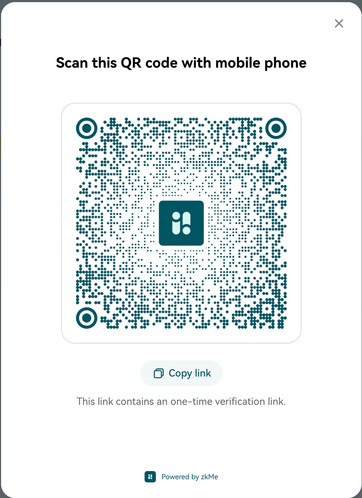
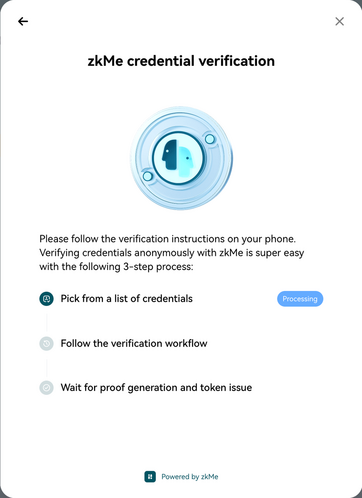
注册登录流程如下：



注：上图中的user private key并不是指用户的区块链私钥（比如metamask私钥），而是指对用户原始身份信息进行加解密所用的私钥的分片之一，后面会详述。

1. **切换到手机端进行KYC**

在PC端网页的widget登录之后，接下来就需要用户做KYC，由于做KYC需要用户对身份证件进行牌照，需要用到手机相机，所以，PC端widget会出现一个二维码（本质上是一个URL），让用户用手机扫描二维码，在手机端浏览器打开这个URL，此时就切换到了手机端浏览器。



用户扫描之后，就在手机端打开了身份验证的URL。



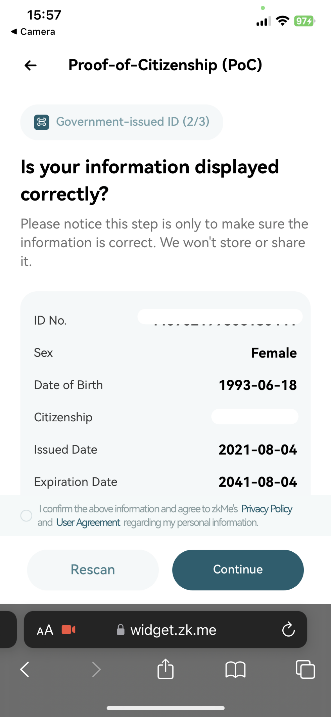
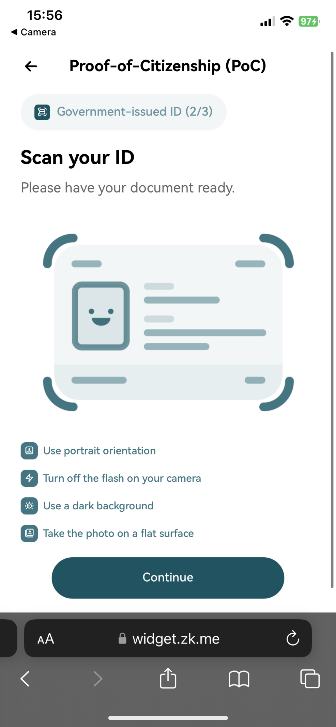
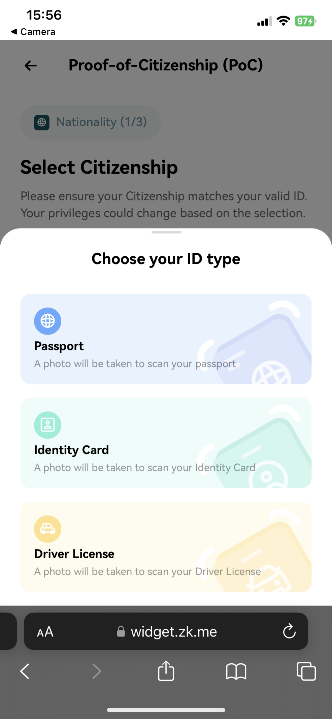
如图所示，身份验证有三个步骤：一是扫描证件，获取信息，二是人脸识别，三是生成零知识证明ZKP，并且在链上铸造SBT（灵魂绑定代币）。

PoC: proof of citizenship

1. **证件OCR：**

用户选择国籍，然后选择证件类型（身份证、护照、驾照等等），然后在手机端调起相机对证件进行牌照，并通过OCR技术对证件的信息进行提取，比如年龄、性别、国籍、生日、证件号、证件颁发日期、证件过期日期等等。

此时这些信息都只在手机端的内存中，还没有上传到服务器，隐私数据没有泄露。

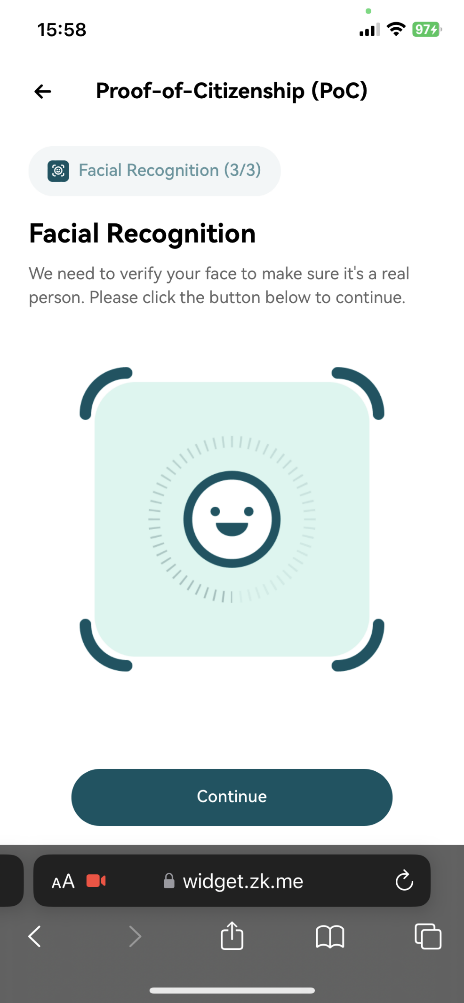
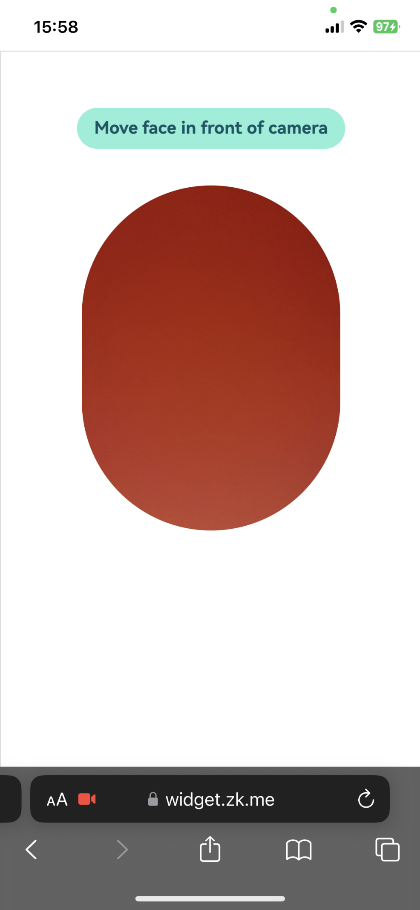


1. **人脸识别：**

用户按提示进行人脸识别，调起相机扫描人脸，人脸识别步骤主要做了三件事情：

一是liveness check（确定是否是活人），二是把人脸和证件OCR中提取的人脸照片进行比对确定是否是同一个人，三是生成了一个独一无二的DID。

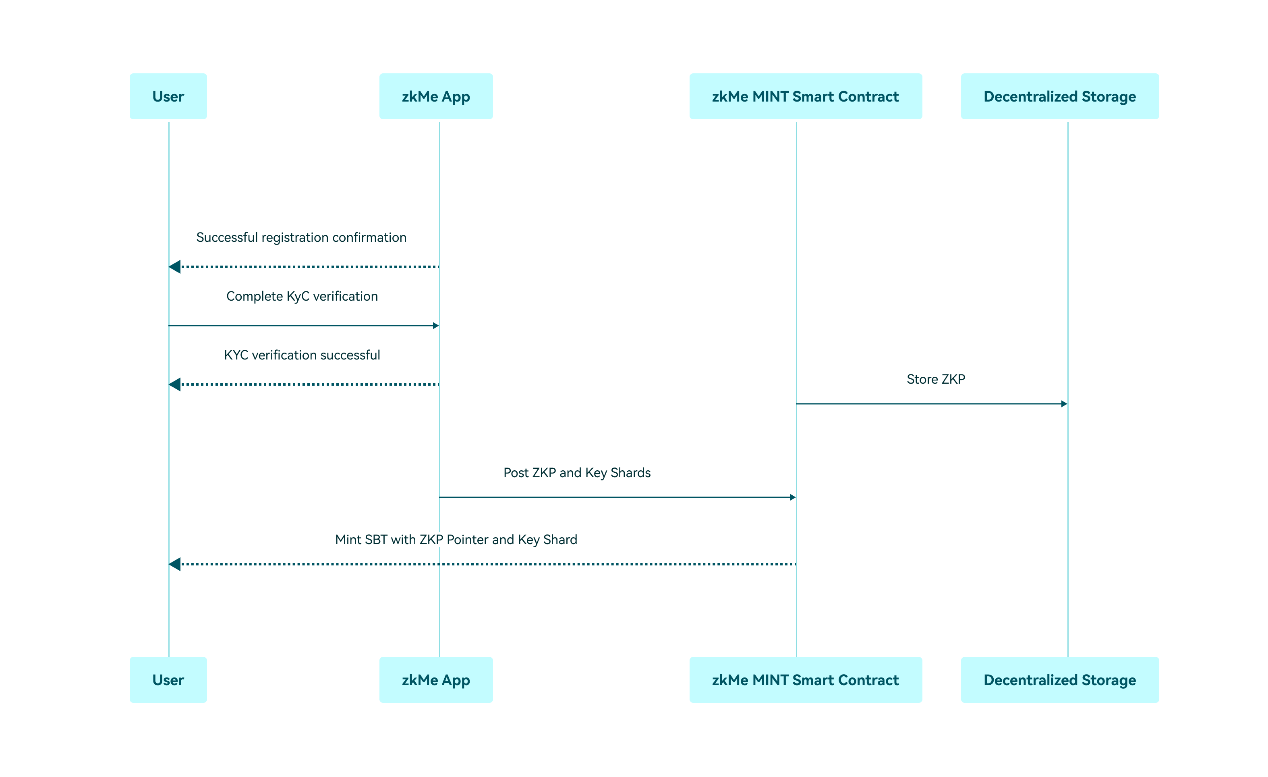
DID值得强调，一张一脸对应一个DID，DID是基于faceprint生成的。

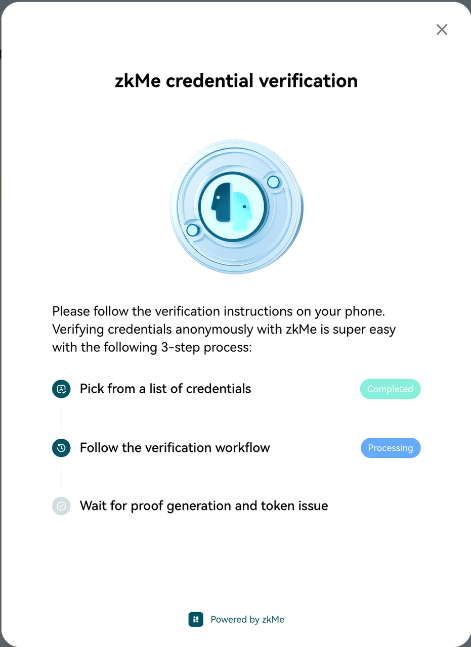


1. **自动生成ZKP、铸造SBT：**

以上1到4步完成后，会在手机端基于证件信息生成VC，并会基于VC生成ZKP VP，包含一些statement，比如用户是否成年，用户国籍等，这些信息并没有暴露用户的隐私比如姓名、证件号等。生成的ZKP（零知识证明）被存储到IPFS去中心化存储中，然后自动在链上铸造SBT（一种非同质化代币，不可转让）代表此用户的身份，SBT保存指向ZKP的指针。

此处是程序自动生成的VC和VP，即由一个写死的私钥对VC和VP进行签名，所以，对应到W3C VC标准，此处VC和VP的issuer就是zkme的一个写死的私钥，后续verifier在验证时可以验证VP是否有此issuer的签名。





1. **用户授权web3 dapp项目验证自己的身份：**

这实际上包含两步：一是授权，二是验证。

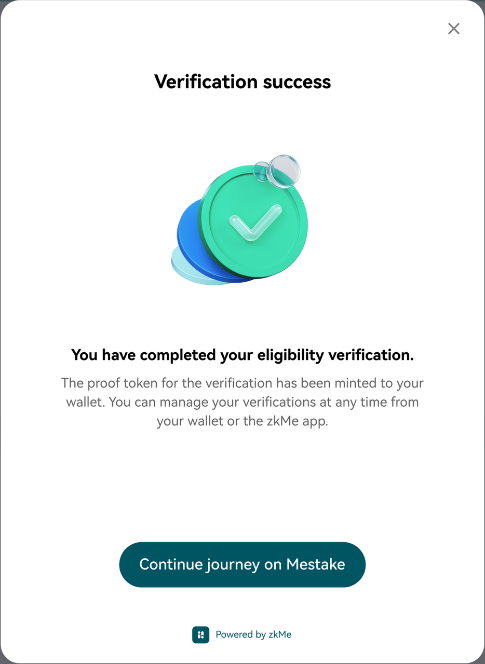
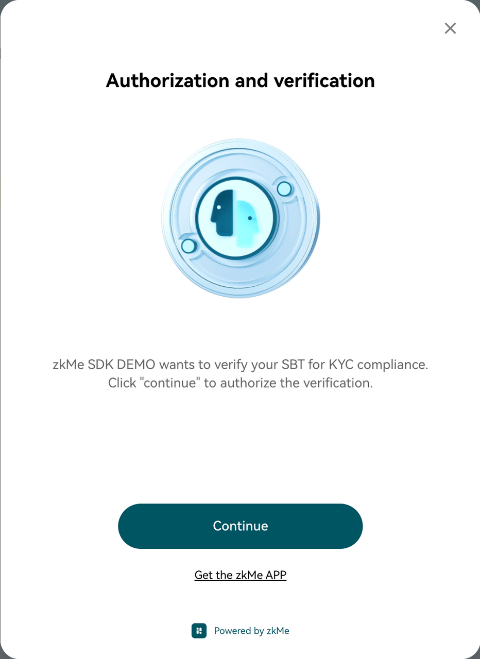
第2到第5步都是在手机端完成的，在第5步完成之后，dapp网页端的widget会自动显示用户已经在手机端完成了KYC和ZKP，SBT的生成，即和身份相关的ZKP已经生成了。

此时用户需要在dapp网页端的widget上点击授权该DeFi dapp对自己的身份进行验证。

授权之后，dapp作为verifier可以通过读取SBT，就能获取相应的ZKP的IPFS链接，进而获取ZKP的内容，发送到链上的ZKP验证智能合约进行验证，验证通过之后，用户才能在dapp的网页上和dapp进行交互。

比如，某些web3 dapp对用户的验证点比较单一，仅仅需要验证一个用户是来自某些特定的国家，或者不是来自某些特定的国家，那么只要ZKP里面包含相关的国籍statement，就可以验证。

注：初始SBT会在polygon铸造，如果要使用其他链的dapp，比如ethereum，就需要多一个用户delegate把SBT复制到ethereum上合约的操作。个人认为这么做是否有必要其实值得商榷。



1. 1-6步用户已经完成了KYC，并自动生成了ZKP和SBT。以后用户再访问该dapp项目，就不用重复做KYC了，也不用重新生成ZKP和SBT了，只需在网页端的widget上进行登录，dapp就可验证用户的ZKP。

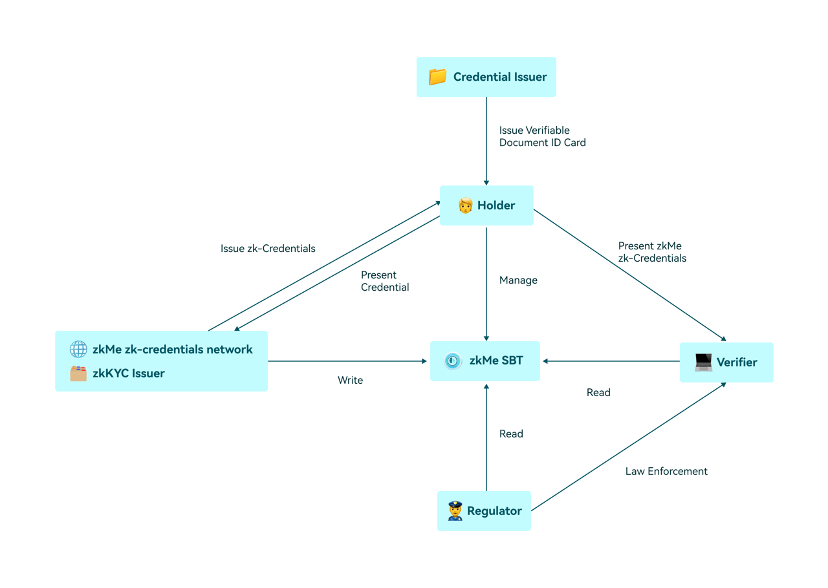
以上就是用户的大致使用流程（不包含技术细节），其中第1,2,3,4,6,7步需要手工操作，第5步的ZKP和SBT的生成是自动的。

更多用户操作图片可以参考：

<https://docs.zk.me/zkme-dochub/getting-started/zkme-components/zkme-widget>

以上用户的使用流程省略了很多的技术细节，以下会一一说明。

### ZK KYC过程涉及到的角色



**Holder**: credential（比如身份证）的拥有者，此处就指代web3 dapp的用户

**Credential Issuer**: 向Holder颁发credential的机构，比如公安部门，通常是权威政府机构

**zkKYC Issuer**: 基于holder持有的credential，进行OCR，签发符合W3C标准的VC和VP的主体，此处就是zkme程序作为issuer

**verifier**: 指web3 dapp，需要对用户进行身份验证，技术上表现为ZKP proof验证。

**regulator**: 监管机构，比如SEC，其可以对恶意行为者的身份信息进行解密。

### ZKP,SBT和用户私钥的解耦

**整个流程都没有和用户的公链私钥（比如metamask私钥）发生直接的联系。**

用户在zkme ZK KYC处的数据有：邮箱地址，DID(和faceprint一一对应)，身份数据，ZKP，SBT。这意味着即使用户ZK KYC和用户的区块链私钥并没有任何关系，用户的ZKP,SBT和用户的区块链私钥在技术上是解耦的。

用户的邮箱地址和DID的映射应该存储在后台数据库中，邮箱地址上链似乎不太合理，因为很多用户的邮箱是经过了实名认证的，上链容易引起隐私泄露。而DID和ZKP, SBT的映射应该在链上。SBT是一个代币，那么其owner就是DID。

所以zkme要验证一个用户的ZKP，就可以通过邮箱地址找到DID，再找到DID对应的SBT，再通过SBT中的ZKP指针找到ZKP，然后验证。由于验证的是ZKP，所以整个过程不会泄露用户的原始数据，且用户对于dapp来讲是匿名的。

### 用户使用的页面汇总

整个流程中用户会接触三个页面：

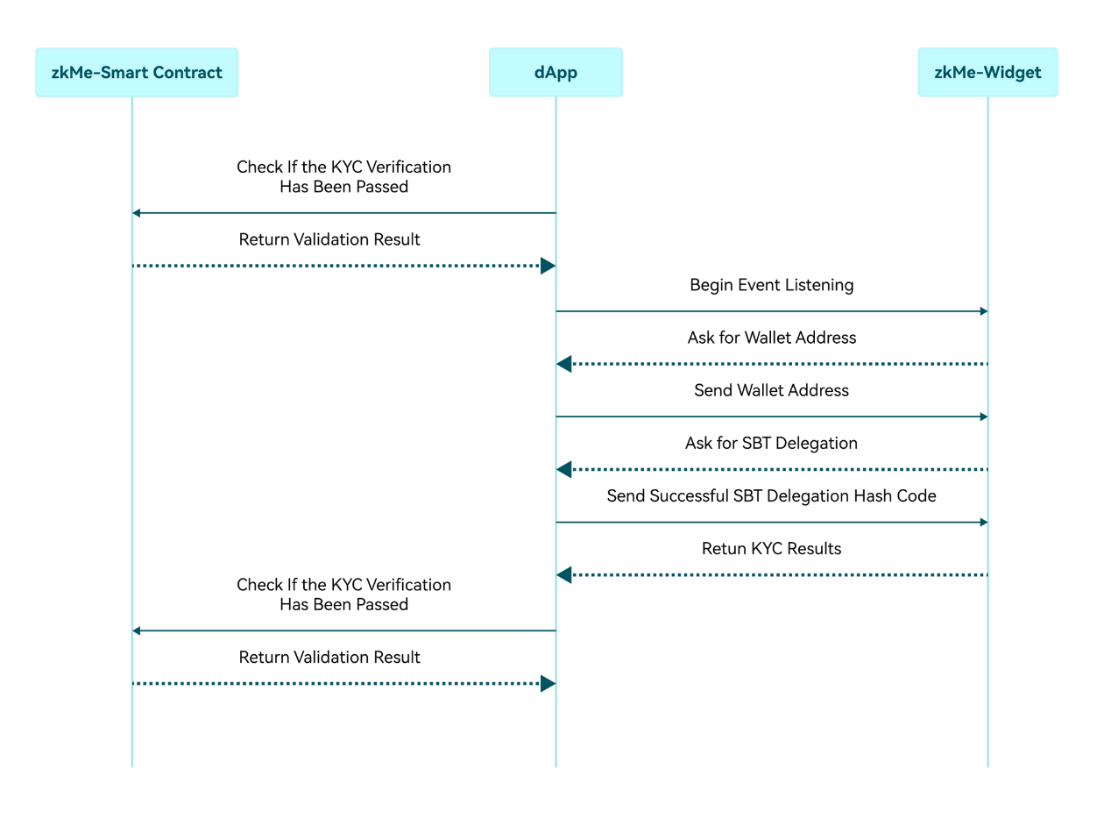


* **dapp页面**：dapp页面通过iframe widget的方式嵌入zkme验证页面，同时dapp页面监听zkme页面传回的用户验证结果（成功/失败）以确定是否让用户使用dapp，dapp页面在监听到zkme页面的验证结果后，还会直接调用链上合约确认验证结果和链上保存的结果是否一致。

dapp页面要集成zkme，就需要添加以上逻辑的代码。

* **zkme PC端页面**：嵌入到了dapp页面中，由zkme开发，dapp开发者只需嵌入即可。主要用于引导用户进行KYC，生成ZKP,SBT，身份验证，以及把验证结果回传给dapp页面。整个身份验证的流程是在zkme页面完成的，本质上是通过zkme的后台完成验证的，然后zkme页面把结果回传给dapp页面。
* **zkme 移动端页面**：用户第一次做KYC，需要从zkme PC端页面跳转到zkme移动端页面进行证件OCR，人脸识别等操作。

### 身份验证过程中dapp H5页面和zkMe-widget H5页面的交互过程



本质上对ZKP的验证是由zkme-widget完成并把结果（true/false）存储在链上合约中，dapp页面做了两件事：一是监听zkme-widget的ZKP验证结果，zkme-widget完成ZKP验证之后会把验证结果通过Window.addEventListener('message')的方式传递给dapp页面；二是收到zkme-widget验证结果之后，直接查询合约上的验证结果进行复核，确保验证的确已经通过。

换句话说，zkme-widget完成了ZKP的验证，并且完成了AML（PEP lists, sanction lists, adverse media）的检查，这些过程一定不是仅靠前端完成的，而是依靠zkme的后端服务完成的，对于dapp来说，这是黑盒。

### 用户身份验证结果的存储和复用

用户在zkme页面完成身份验证后，zkme后台会自动把验证结果存储到链上合约中，基于“appId+user address”进行存储，意思就是某个用户的区块链地址已经在某个dapp通过了身份验证，下次如果该用户用同样的地址访问同样的dapp，就不用再做身份验证了，直接让用户访问dapp。

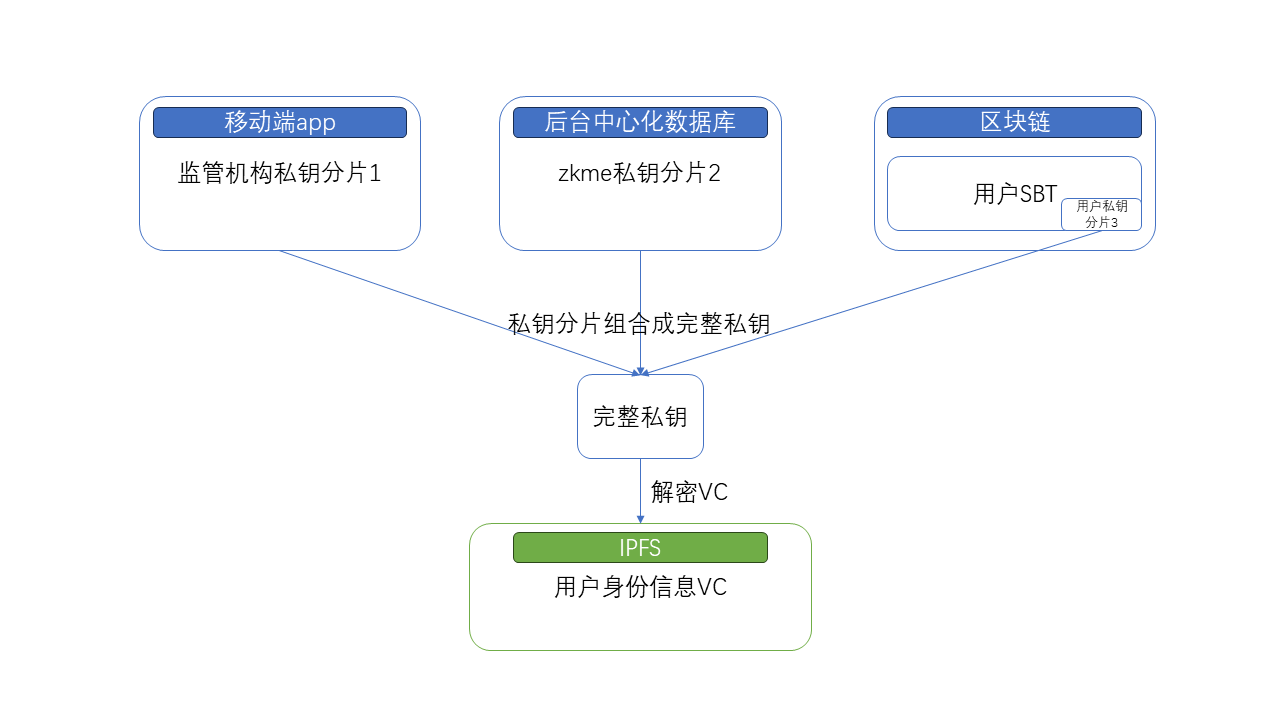
之前讲到用户的DID和地址是解耦的，只有在存储身份验证结果的时候使用到了用户地址。

但需要注意的是，一个用户地址在dapp 1完成了身份验证，其在访问dapp 2的时候还需要再重新做身份验证。这是因为虽然都叫“身份验证”，但dapp1和dapp2的验证点可能有差异，在dapp1通过身份验证并不代表在dapp2就能通过身份验证。

### 用户身份明文的存储和解密

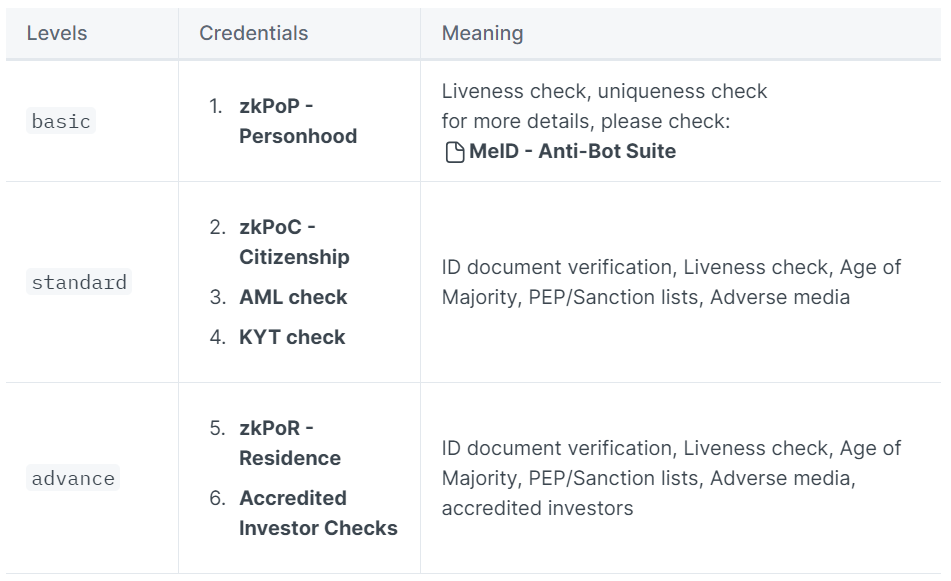
用户在完成KYC流程之后，生成了ZKP，ZKP本质上是W3C VP，其是由VC生成的，VC可以理解为用户身份信息的明文，其存储是值得关注的问题，zkme采用了门限加解密的方式，把一个私钥分成三份，对VC进行加密，存储在IPFS保证持久性，要解密的话需要三份私钥合并在一起才能解密，任何单方都没有办法解密。

正常情况下，不需要对VC进行解密，只有在发现用户有违规操作或身份有问题时，由监管结构发起对用户VC的解密，查看用户的原始身份信息并展开调查。

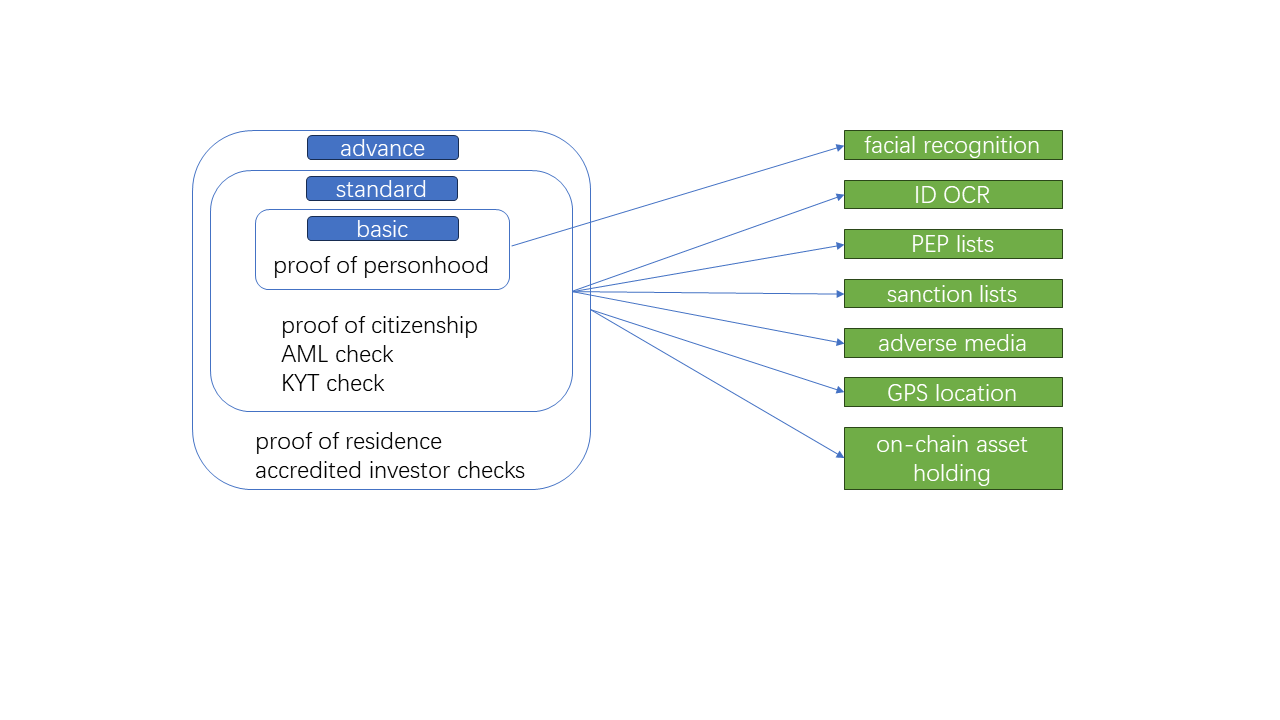


### 身份验证的具体验证内容

在用户完成KYC生成ZKP和SBT后，就是dapp触发zkme对用户身份的验证过程，具体验证的内容是分级的：



具体是验证哪一级，**取决于dapp的需求和选择**，dapp在集成zkme的时候需要选择自己需要哪一级的身份验证。



* **zkPoP**: proof of personhood，只验证是否是一个活人即可。
* **zkPoC**: proof of citizenship，除验证是否是活人外，还要验证身份证件。此处zkme的文档很模糊，介绍得很笼统，没讲到验证细节，从理论上讲，每个dapp对于citizenship的验证点也是不同的，有些dapp验证年龄，有些dapp验证是否来自限制的国家，有些dapp都验证，所以，proof of citizenship应该不是一个统一的验证标准，而是需要对dapp定制化，但zkme的文档并没有介绍相关内容。
* **AML check**：主要是Politically Exposed Persons (PEPs) lists、sanction lists和负面媒体报告的检查，即检查用户是否是政治人物（政治人物需要更加严格的KYC），是否在制裁名单，是否有负面的媒体报道。这些检查涉及到和有公信力的lists的比对，细节不清楚。
* **KYT check**：检查用户当前准备发起的区块链交易是否有风险，zkme文档对此没有做详细介绍，细节不清楚。
* **zkPoR**: proof of residence，一是需要用户通过当前GPS定位生成定位proof，二是需要信用卡账单地址或者公共费用地址（比如水电燃气费）地址确认用户的地址。具体细节zkme文档没有介绍。
* **accredited investor checks**: 合格投资者检查，检查用户在链上是否有足够的资金（比如ETH, USDC），zkme文档也没有详细介绍。

以上的验证点中，zkPoP和zkPoC相对比较清晰，只需要证件OCR和人脸识别即可。其余的检查点zkme只是一笔带过，可能是未来的规划，不知道进展到什么程度了。

### 初始在哪条链铸造SBT&多链支持

第一次KYC通过之后，生成ZKP并铸造SBT，此时SBT会在**polygon**进行初始铸造，因为gas费便宜。如果用户后续使用的dapp是在其他链，则会有一个把SBT从polygon复制到其他链的过程，此过程需要用户手工触发，成为delegate。这是链上操作，但并没有要求用户支付gas费，而是由zkme的后台发起的，由zkme支付gas费，至于zkme会不会把gas费转嫁给dapp项目方，文档没有清晰介绍其商业模式，所以不清楚。

zkme的目标是支持大部分区块链的ZK KYC，这涉及到在多条链上部署合约并进行技术适配。其中关键问题是gas费谁支付，目前没有看到由用户支付的设计，似乎是由zkme项目方承担了gas费。

支持及计划支持的链包括：

**Aptos, Arbitrum, Avalanche, Base, BNB, Ethereum, Fantom, Linea, Manta, Mantle, Optimism, Polygon, Scroll, Sei, Solana, Sui, ZetaChain, zkSync**

### 人脸faceprint的隐私

faceprint不会持久化存储，只会在人脸识别的时候存在于手机端的内存中，一旦结束人脸识别，生成faceprint对应的唯一DID之后，faceprint就被抹去了，所以faceprint不会被泄露。

### 身份证件源头造假的问题

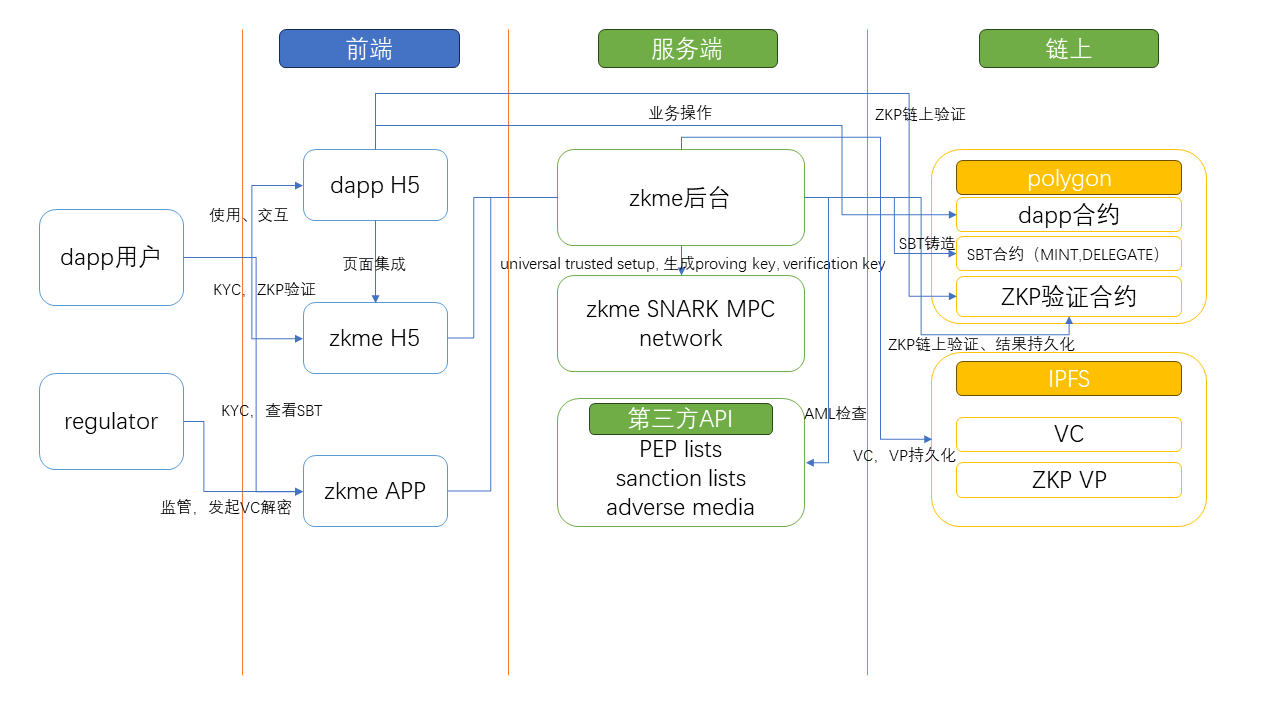
由于支持很多国家、各种类型的身份证件，所以仅凭证件OCR的过程很难避免证件造假，用户出具一张假的身份证，很可能能够通过OCR。然而，整个系统没有对证件进行核验的机制，证件的核验需要通过权威部门的相关接口进行，意味着需要对接各个国家的相关核验接口，这似乎难度很大。所以，如何从源头上规避证件造假是个需要考虑的问题。

## 总体架构

zkme官方架构图如下：

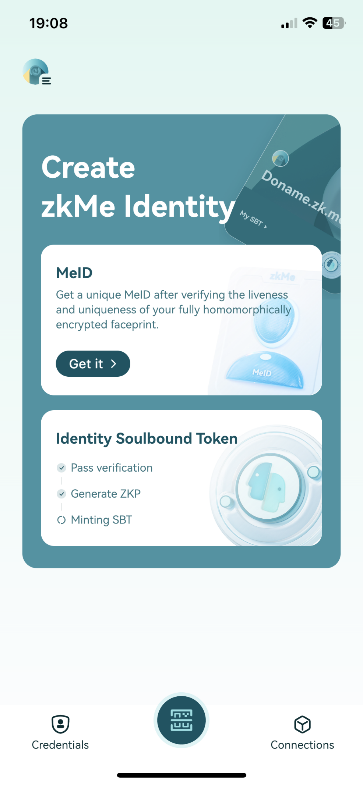


结合文档细节整理的架构图如下：



简要介绍一下各角色/模块的作用：

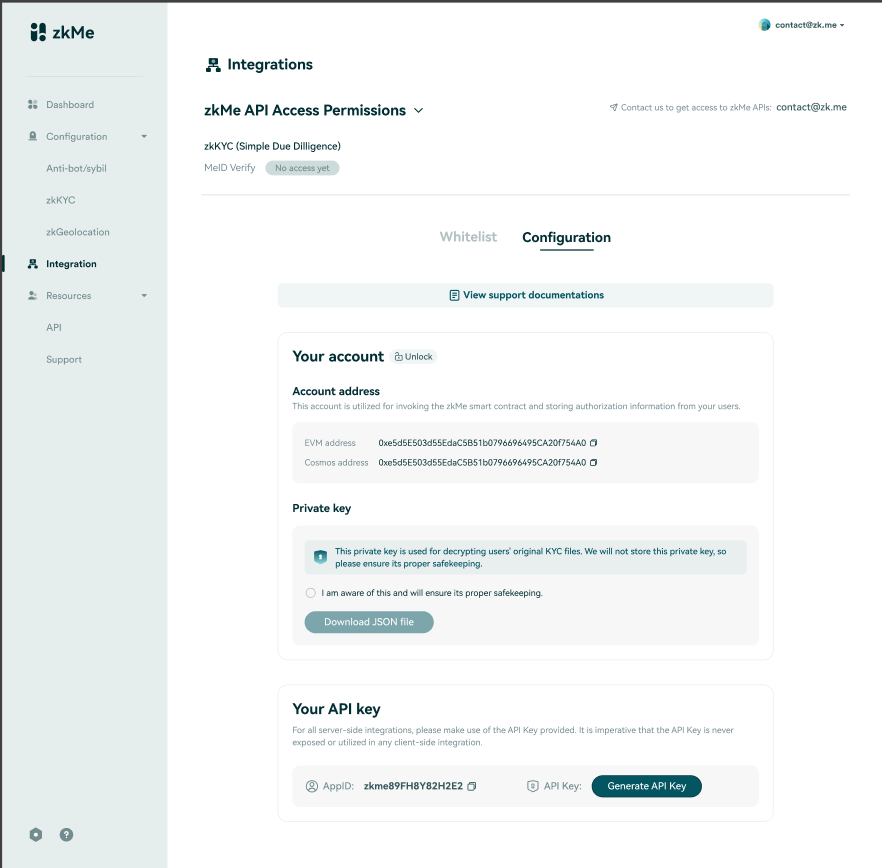
* **dapp用户**：web3 dapp的使用者，也是KYC的对象，SBT的owner
* **regulator**：监管者，当zkme身份认证发现恶意用户之后，regulator可以发起对dapp用户的原始身份信息（IPFS上的VC）的解密，然后进行线下调查和执法等
* **dapp H5**: web3 dapp的前端页面
* **zkme H5**: 分为两部分，一是作为widget嵌入dapp H5的页面，当用户访问dapp时，引导用户做KYC，通过widget集成zkme H5的原因是通过避免打开新的网页降低用户的流失率，二是通过扫码在手机端打开的dapp H5页面，做证件OCR，人脸识别，ZKP生成，SBT铸造。
* **zkme APP**: 用户不仅可以在H5页面上做KYC，也可以在zkme APP上做KYC，可以把zkme APP理解为是H5的替代。用户还可以通过zkme APP查看属于自己的SBT，zkme APP的角色相当于一个钱包，如下图：



* **zkme后台**：ZK KYC业务处理中枢，负责和合约交互铸造SBT合约，和IPFS交互储存VC,VP，和合约交互进行ZKP statement验证等等。
* **zkSNARK MPC network**：zkme采用的零知识证明体系是zkSNARK groth16，有两个trusted setup的过程需要多方参与以降低单一主体作恶的可能，这里涉及到MPC，后面零知识证明部分再详细介绍。
* **第三方API**：主要用于AML检查，比如PEP lists，sanction lists，adverse media，多维度地验证用户的身份。
* **dapp合约**：比如DeFi合约，uniswap, compound, aave合约等等
* **SBT合约**：用于铸造SBT，以及delegate等等。
* **ZKP验证合约**：在合约中实现ZKP验证逻辑，提供verifyProof view函数，可随时随地实现ZKP proof的可信验证，不需要依赖于中心化实体进行proof验证。
* **IPFS**：主要用于存储加密后的VC和ZKP VP。

## dapp和zkme的集成

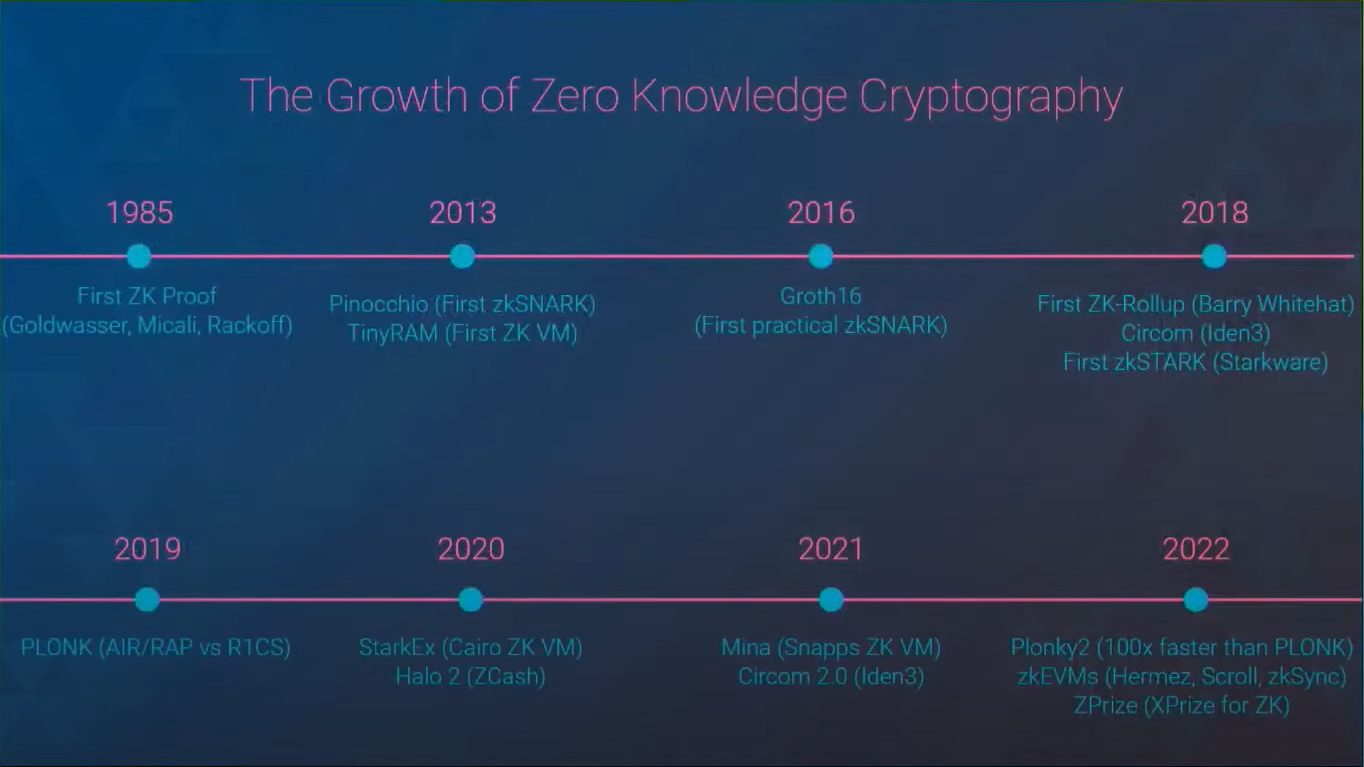
zkme的客户是dapp项目方，zkme提供guideline让dapp集成其ZK KYC功能。为dapp项目方提供dashboard，让其进行配置。



# 零知识证明技术栈

上文已经提及了多次ZKP的验证，接下来详细介绍所使用到的零知识证明技术栈。

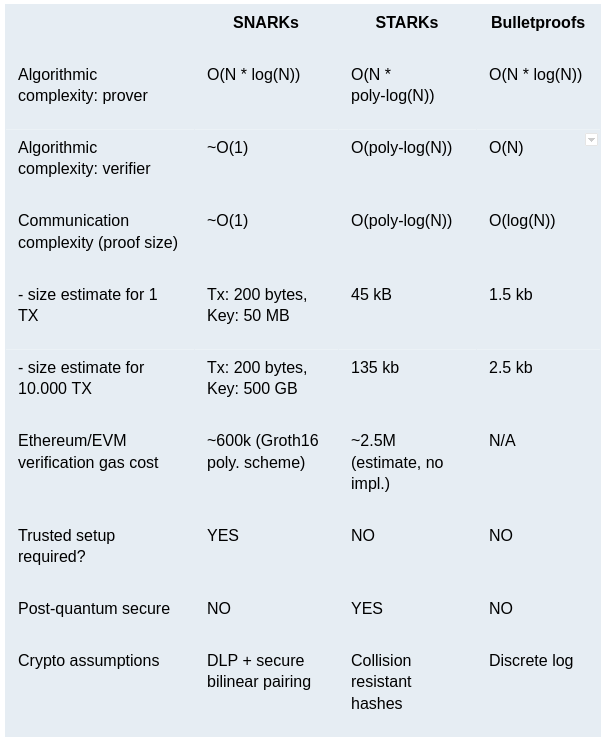
## 零知识证明技术发展的历史脉络



从上图可以看出，零知识证明技术的重大发展基本都是近10年之内取得的，并且受到区块链发展需求的强烈驱动，可以想象，随着区块链技术的持续发展，零知识证明技术也会得到更好的发展。

## 零知识证明的技术流派

目前比较主流的零知识证明技术流派包括SNARK, STARK, Bulletproof，它们的区别如下：



这些区别中，有几点比较关键：

* **Trusted setup:**

SNARK有一个初始化的步骤，就是trusted setup，必须做完这一步才能生成proof，这一步需要多方参与防止作恶，涉及到MPC，所以SNARK的trusted setup比较麻烦和繁琐。STARK和BulletProof是不需要trusted setup的，省略了需要多方参与的MPC，更加简单。所以，在trusted setup这一点上，SNARK看上去是没有优势的。

* **verifier算法复杂度和通信复杂度(proof size)：**

从上图可以看到，**SNARK的verifier算法复杂度和通信复杂度都是O(1)**，是常数，意味着对ZKP的验证效率大大高于STARK和BulletProof。从产品的角度来看，proof的生成是一次性的事情，多花点时间无所谓，但proof的验证可能是需要verifier多次执行的过程，所以**proof验证的效率就显得非常重要**。正是由于SNARK在验证时间和proof size上相对STARK和BulletProof具有非常明显的优势，所以，**SNARK在web3领域面向应用的ZKP系统中几乎是唯一选择，即使它仍然需要trusted setup**。

zkme选择SNARK也是出于以上原因。接下来我们就围绕SNARK做详细介绍。

## SNARK

SNARK这个技术体系下又分为三种不同的算法：

**groth16, PLONK, FFLONK**

* groth16不仅需要universal trusted setup, 还需要circuit specific trusted setup；PLONK, FFLONK都只需要universal trusted setup，流程上比groth16简单一些。
* PLONK的提出晚于groth16，在证明大小和验证速度上与groth16有一定差距，所以groth16的验证效率最高。无论proof有多大，verifier验证proof所需要的工作量都是相同的，计算复杂度是O(1)。
* FFLONK的验证效率如何还未深入调研。

由于universal trusted setup是一次性的工作，如果circuit已经确定了，circuit specific trusted setup也是一次性的工作，所以在ZK KYC的场景中，应该主要考虑proof的验证效率。**zkme选择了groth16，主要还是出于验证效率的考虑。**

## groth16

SNARK groth16的流程图如下：



具体流程细节如下：

1. **universal trusted setup**

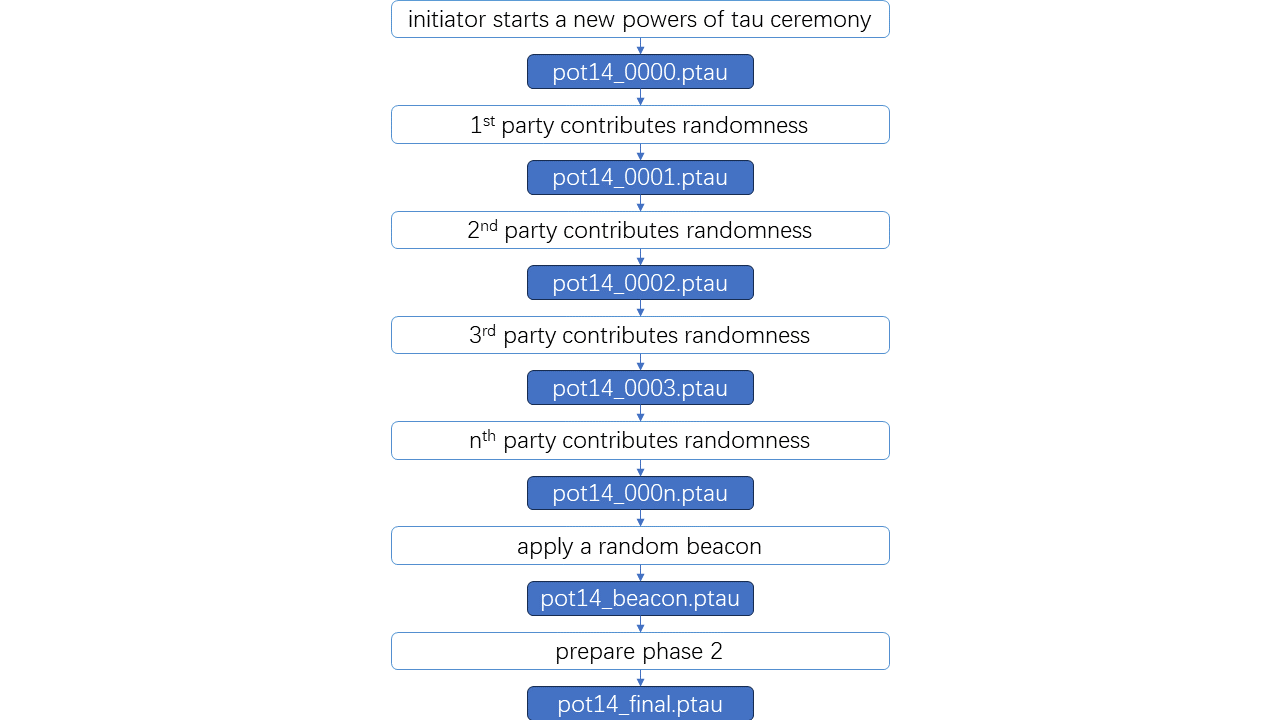
这一步也叫做powers of tau ceremony，是一个全局初始化的操作。生成和验证zkSNARK proof都需要提前生成一个common reference string (CRS)。这个过程可以被认为是创建一个只有系统知道的**秘密(secret)**，任何知道CRS是如何生成的人都可以伪造证明，从而破坏可靠性。

ZK系统的安全性在很大程度上归结为生成CRS的安全性。因此，在所谓的“setup ceremony”中安全地生成CRS的方法非常重要。当然，以可信的、集中的方式生成这些参数是可能的，但与web3去中心化的目标不相容。到目前为止，在zkSNARK setup ceremony中使用的首选技术是多方计算(MPC)。

MPC方案试图确保没有任何一方生成或能够获得CRS底层数学结构的知识。通过要求在尽可能多的独立参与方之间共享生成过程来实现这一点，这样只需要有一小部分人(甚至单个人)诚实行事，就可以确保setup的安全性。

简单来说，universal trusted setup就是需要多个独立参与方通过MPC生成一个CRS。

流程如下：



在具体实操中，有两种方式生成这个CRS：

* **方式一**：找到多个独立的参与方协商，大家一起生成这个CRS，这种方式涉及到线下的沟通。
* **方式二**：既然最终目的是生成一个CRS，只要能够找到一个曾经被多个独立的参与方生成过的CRS，拿过来用即可。例如，Semaphore项目团队正在发起一个多方参与的可信配置第一阶段的活动，这个活动是在Zcash之前已发起过的类似活动的基础上演进而来的。其目的就是为了安全可靠的生成zk-SNARK的电路参数，最大支持的电路门限制数目多达2亿6千多万个（2的28次方）。这也意味着会计算得出约2倍大小的Tau(τ)，大概5亿2千万个。只要任何一个参与者能诚实参与，那么通过这套多方计算系统最终得到的计算结果就是可信的。目前已经生成了很多tau文件，可以认为是可信的，直接拿过来用。

详细情况可以参考：

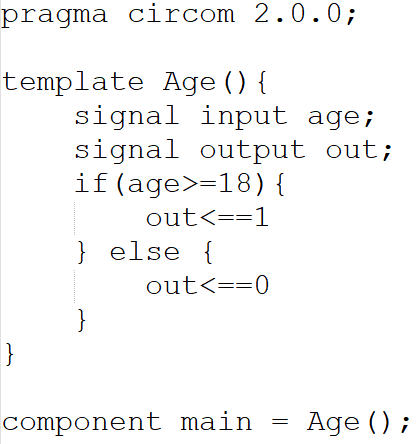
<https://github.com/privacy-scaling-explorations/perpetualpowersoftau/blob/master/README_ch.md>

1. **用circom语言开发circuit**

circuit翻译为电路或回路，可以简单理解为是一个函数，这个函数包含特定的规则。比如定义函数的input是age，判断age是否大于等于18。circuit定义函数的逻辑和参数，但参数值还未传入。

circuit本身是很底层的，但可以用上层语言circom开发，开发好之后，编译为底层circuit。

一个简单的circom circuit示例如下：



以上circuit的逻辑是：

入参为age，默认是private的，即verifier不可见，verifier在验证proof时不会知道具体传入的age是多少。输出是out。如果age大于等于18，即成年，out为1，否则，out为0.

注意，此时还未生成proof，proof是基于circuit和具体的input（此例中为age）生成。

1. **编译circom circuit文件**

可以把circom文件理解为java的源文件，需要编译为更底层的字节码。circom circuit文件编译后生成三个文件：

r1cs文件，wasm文件，sym文件

sym文件作为debug之用，暂且不管。

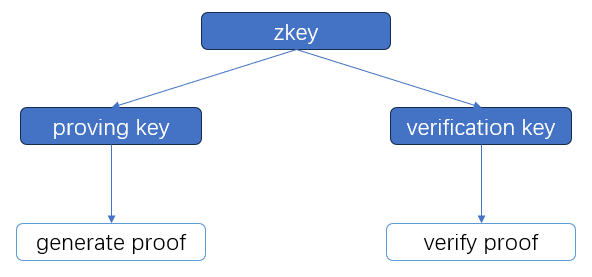
1. **circuit specific trusted setup ceremony**

r1cs文件需要作为特定于这个circuit的trusted setup ceremony的入参，ceremony的输出是zkey。

circuit specific trusted setup ceremony和universal trusted setup ceremony的流程基本一样，都需要多个独立参与方参与MPC。

zkey其实包含两个密钥，一是proving key，用作生成proof；二是verification key，用作verifier验证proof。

从zkey可以导出verification key，给verifier使用。



1. **生成proof**

wasm文件和input.json（此例中包含age的具体值，比如20）一起作为入参生成witness文件，witness又和zkey一起作为入参生成最终的proof（ZKP）和public.json供verifier验证。

proof就是prover生成的零知识证明文件；public.json就是经过计算后生成的public的input和output，在以上circuit的示例中，如果传入age 20，那么out就是1，由于age是private的，out是public的，所以在public.json中，只包含out的值，为1.

此处关键点就是proof的生成有zkey作为入参，更准确地说，是zkey中的proving key作为入参。

1. **验证proof**

verifier用verification key验证proof和public.json，此例中，verifier能够得到的验证结果是：age是大于等于18岁的，但verifier并不知道age的具体值是多少。

能够得到这个结论的原因是public.json中的out等于1，那么根据circuit的逻辑，age必然大于等于18. 如果对public.json造假，比如把out改成0，那么在验证时，会失败，零知识证明本身就能保证数据的防篡改和完整性。

由于零知识证明的验证逻辑是有固定算法的，所以在具体实现上，可以把验证逻辑搬到链上智能合约中，形成一个verifyProof方法，接受verification key, proof, public.json为入参，返回验证结果true或false。比如示例中，public.json中out为1，和proof一起传入verifyProof方法验证，会返回true；如果public.json中out为0，传入verifyProof方法验证，会返回false，所以，verifier都能得出age>=18的结论，同时又没有泄露具体的age值。

## 零知识证明在ZK KYC中的使用

zkme ZK KYC生成proof和验证proof同样遵循上述的groth16流程，但有一些关键点值得注意：

|  |  |
| --- | --- |
| universal trusted setup ceremony | zkme没有明确说明是如何做这一步的，有可能是自己找了一些独立的第三方，通过MPC的方式生成ptau文件，也有可能直接取的社区内已经存在的ptau文件。 |
| circuit specific trusted setup ceremony | 由于circuit是和业务强相关的，比如，判断age可以写一个circuit，判断国籍可以写一个circuit，每一个circuit都需要trusted setup的流程，这就只能通过自己组织多方进行MPC来生成特定于circuit的zkey。多方就形成了一个MPC网络，即zkme network，但zkme的文档没有透露这个zkme network的细节，具体有哪些参与方。 |
| zkme中谁创建circuit | circuit的创建应该是zkme团队提前根据业务需求（比如年龄、国籍）提前开发好了。 |
| zkme中谁生成proof，数据源的信任问题 | proof是基于circuit,input,zkey生成，每个用户在KYC之后都需要生成proof。简而言之，基于一份circuit，生成成千上万份的proof。  zkme中是用户手机端APP或H5在对身份证件OCR之后，得到了原始信息（比如age），然后在手机端直接生成proof，然后传给后端服务，后端服务把proof存储到IPFS，且铸造SBT，指向这个proof。  整个过程都是程序自动完成的，没有人工干预，只要zkme的程序逻辑是值得信任的，那么这个proof的数据来源就是值得信任的，比如程序不会把年龄20乱填成7. |
| zkme中谁验证proof | 嵌入dapp页面中的zkme-widget验证proof，本质上还是通过zkme后端，从IPFS取到proof和output.json之后，作为入参传递到链上验证合约进行验证，verification key是公开的，保存在链上合约中。只要zkme后端不作恶，那么验证结果就是值得信任的。  如果技术极客不信任验证结果的话，可以自己去IPFS取proof和output.json，然后传到链上验证合约验证，即可佐证验证结果。  通过链上合约验证，减少了对中心化服务的依赖，同时更加可靠，7\*24验证。 |

关于SNARK groth16，更专业的知识可以参考如下链接：

<https://medium.com/coinmonks/simplifying-powers-of-tau-and-the-trusted-setup-ceremony-c9a4f1833dc2>

<https://zeroknowledge.fm/the-power-of-tau-or-how-i-learned-to-stop-worrying-and-love-the-setup/>

<https://github.com/iden3/snarkjs>

<https://docs.circom.io/>

<https://medium.com/swf-lab/circom-snarkjs-728e4314e057>

<https://www.youtube.com/watch?v=wyBNZzLEgkA>