PADO协议的技术分析：关于数据真实性和隐私的故事

Fubiao @ PADO Labs

Outlines

- 背景和问题
- 基于MPC和IZK的数据分享
- 与NIZK的结合
- 基于PADO的信用协议建设

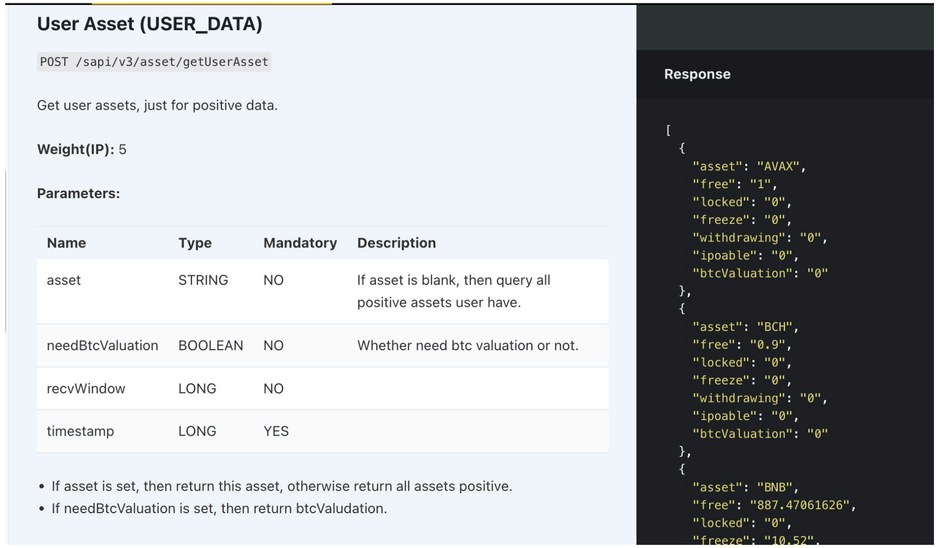
PADO

1

背景和问题

PADO

如何分享你的数据?



User Asset (USER_DATA)

`POST /sapi/v3/asset/getUserAsset`

Get user assets, just for positive data.

Weight(IP): 5

Parameters:

Name	Type	Mandatory	Description
asset	STRING	NO	If asset is blank, then query all positive assets user have.
needBtcValuation	BOOLEAN	NO	Whether need btc valuation or not.
recvWindow	LONG	NO	
timestamp	LONG	YES	

- If asset is set, then return this asset, otherwise return all assets positive.
- If needBtcValuation is set, then return btcValudation.

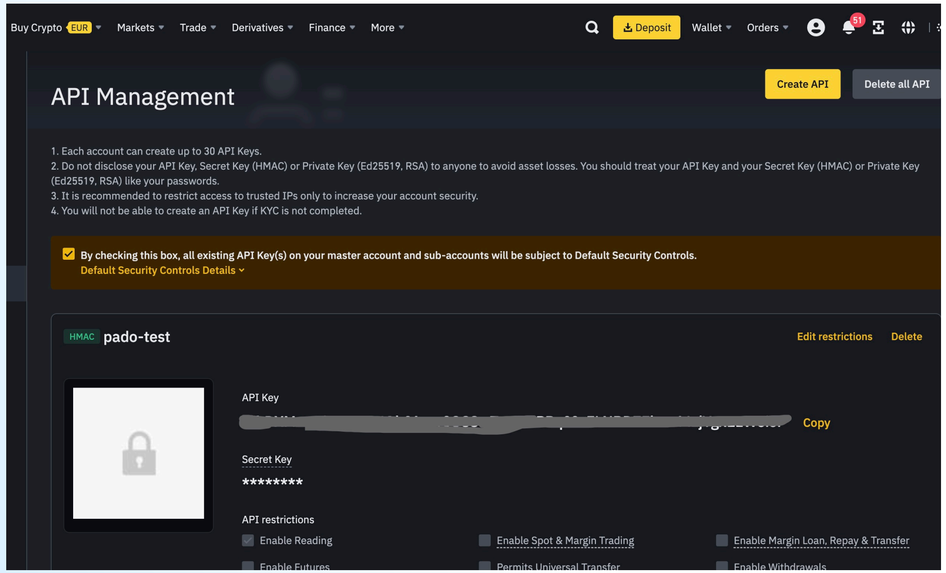
Response

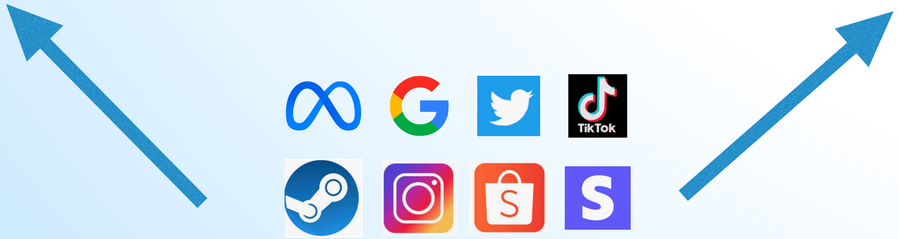

```
[
  {
    "asset": "AVAX",
    "free": "1",
    "locked": "0",
    "freeze": "0",
    "withdrawing": "0",
    "ipoable": "0",
    "btcValuation": "0"
  },
  {
    "asset": "BCH",
    "free": "0.9",
    "locked": "0",
    "freeze": "0",
    "withdrawing": "0",
    "ipoable": "0",
    "btcValuation": "0"
  },
  {
    "asset": "BNB",
    "free": "887.47061626",
    "locked": "0",
    "freeze": "10.52",
```


- 告诉别人我是一个VIP?
- 展示我是一个资深的OTA用户?
- 向别人证明我有很多Cryptocurrencies?

PADO

基于API的数据分享



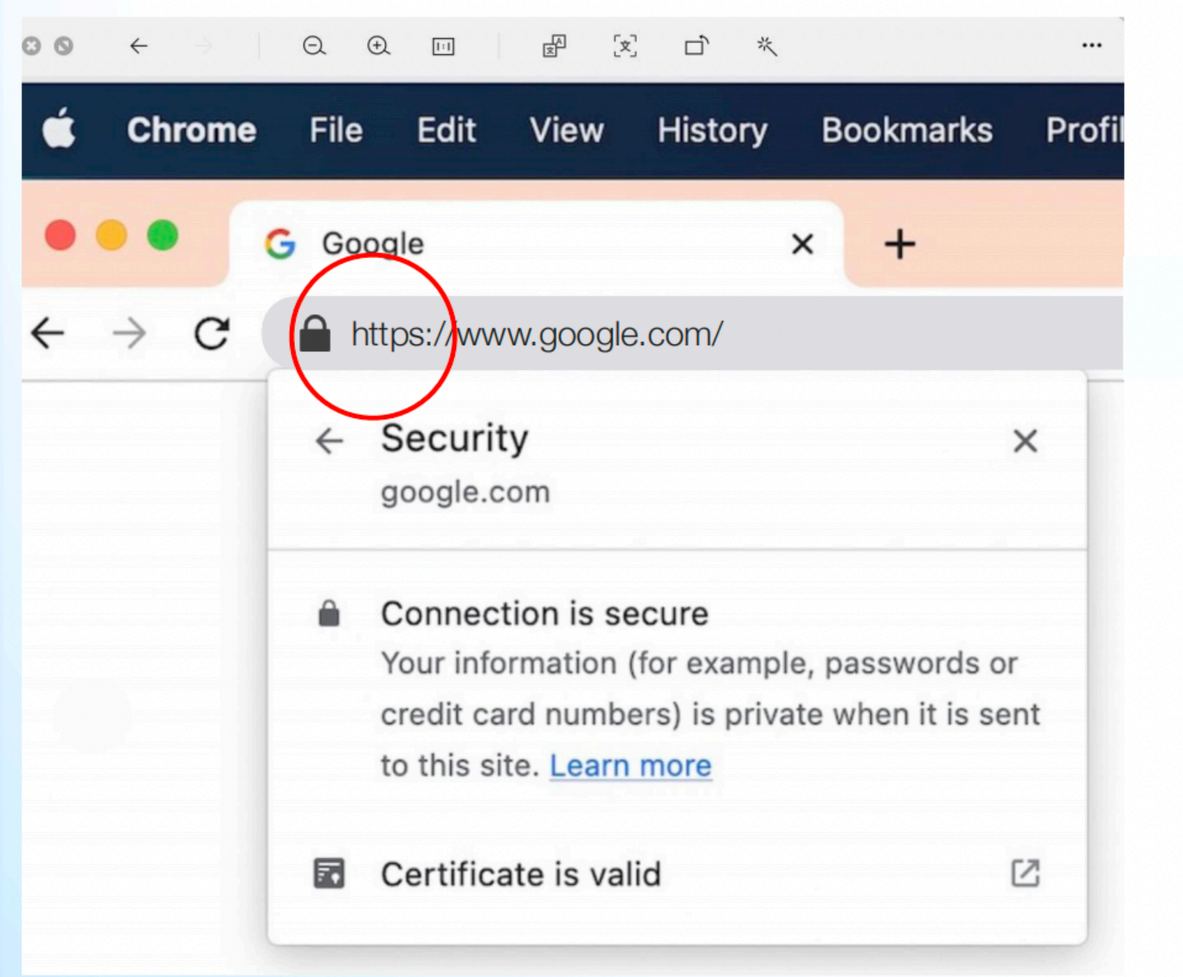


存在的问题:

- 用户会相信数据（API响应），但是无法转移事实
- 数据真实性 (authenticity)：提供的数据，其来源正确（authentication），未被篡改 (integrity)
- 隐私保护（privacy-preserving）：用户的敏感数据不可泄露

PADO

TLS协议



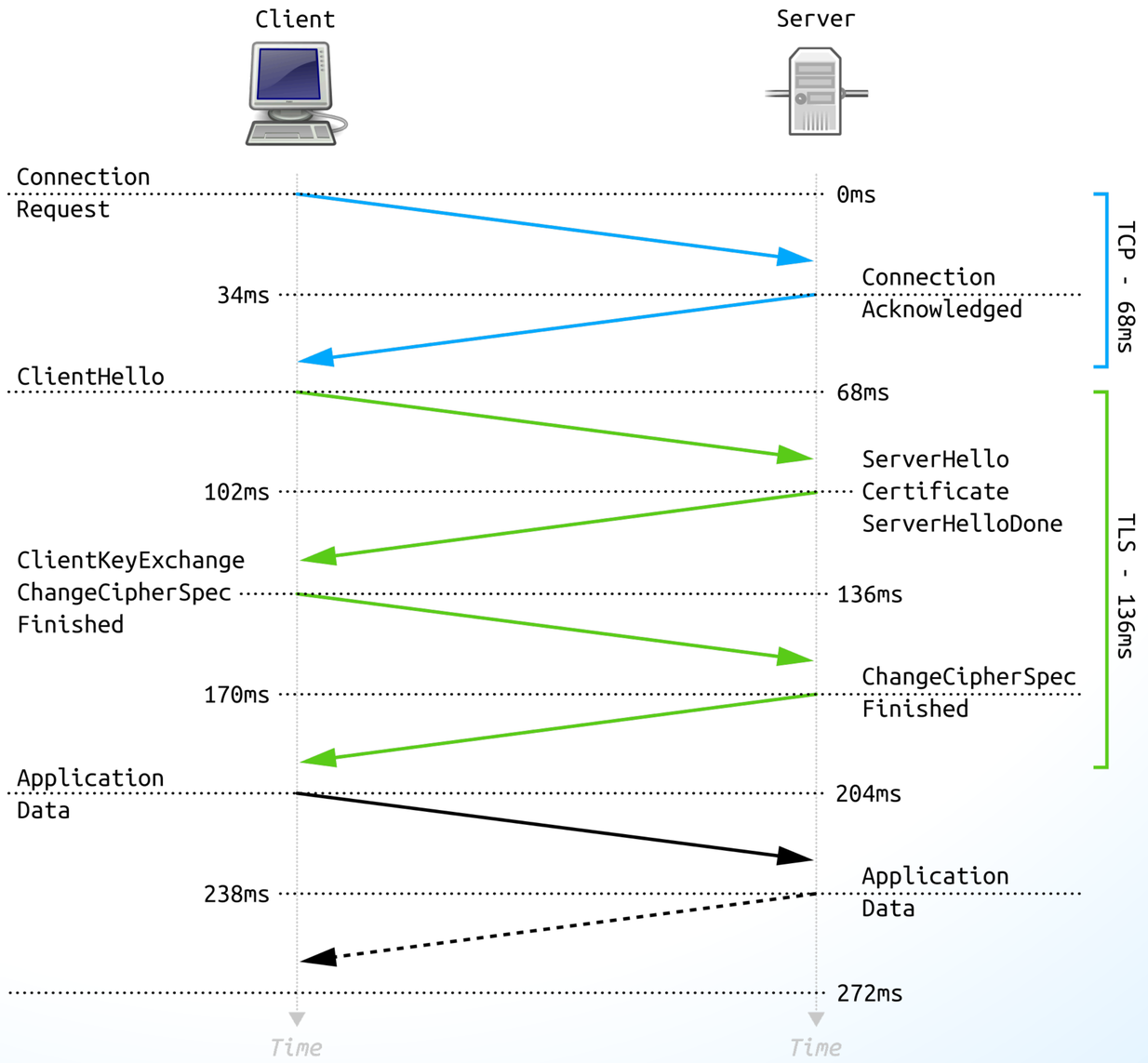
- Transport Layer Security (TLS) 是一个基于密码学的安全传输协议，广泛用于邮件、消息通信、语音电话，以及最主要的Web通讯 （HTTPs）。

- TLS协议主要保护了服务器和客户端之间的通信安全，包括机密性（隐私），完整性和真实性。内部涉及了多项密码学技术，如证书、签名、加密、密钥协商、认证码等。

PADO

TLS协议

- TLS Handshake (握手协议)
- TLS Record （记录协议）
- TLS Version:
  - TLS 1.2 — 99.7%支持
  - TLS 1.3 — 54.2% 支持
- CipherSuites:
  - ECDHE_RSA_AES128_GCM_SHA256,
  - ECDHE_ECDSA_AES128_GCM_SHA256,
  - …



是否有办法分享经由TLS协议传输的数据，确保数据的真实性，以及兼顾隐私安全?

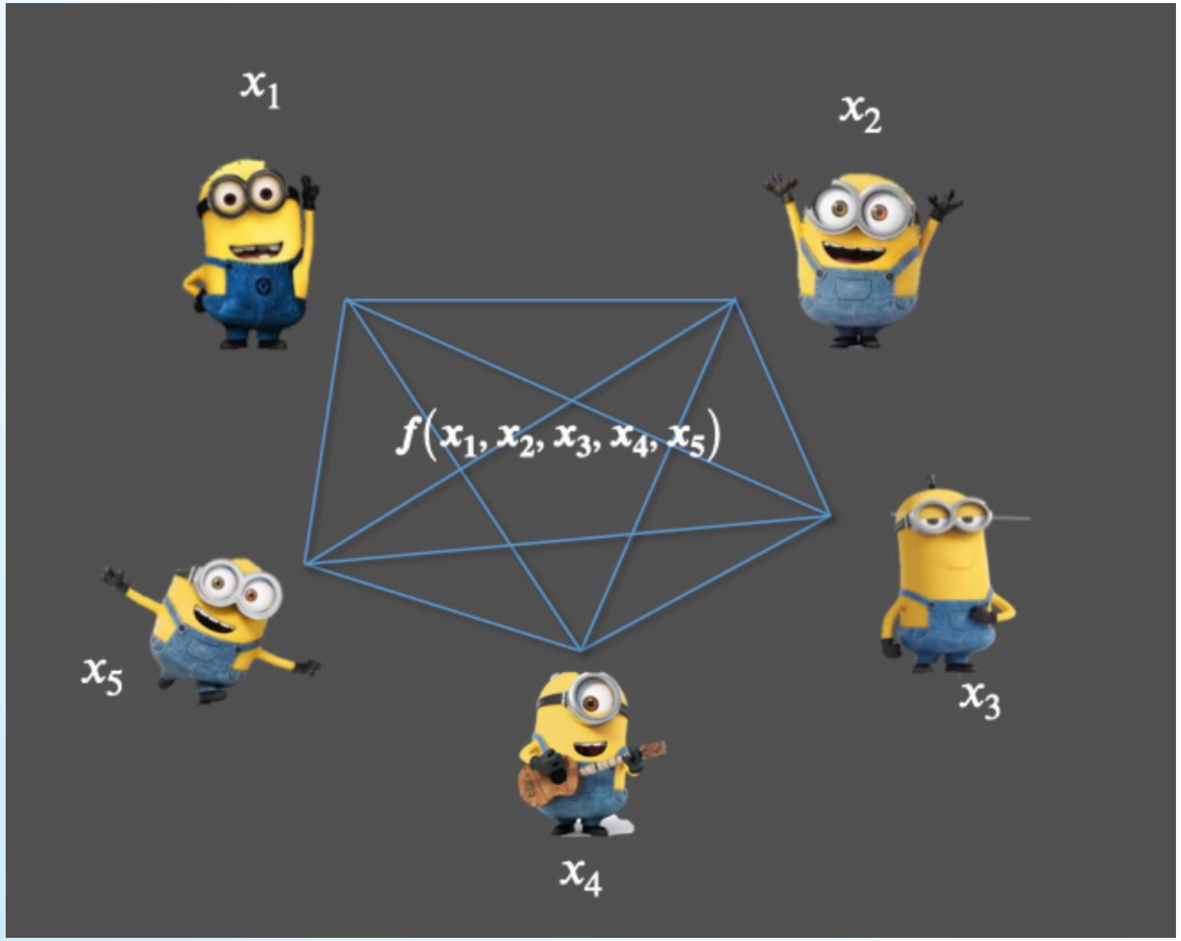
PADO

2

基于MPC和IZK的数据分享

PADO

MPC



安全多方计算 secure Multi-Party Computation（MPC）

- 各方不透露各自的隐私输入;
- 除计算结果以外，各参与方无法通过计算过程中的交互数据推断出其他参与方的原始数据;
- 通用 MPC 协议：混淆电路 GC, 秘密分享 SS, 同态加密HE, …
- 专用 MPC 协议: 门限签名 TSS, 隐私求交 PSI, 隐匿查询PIR, 隐私保护机器学习 PPML, …

PADO

IZK

非交互式零知识证明 （NIZK）

- $P(\sigma, x, w) = \pi$
- $V(\sigma, x, \pi) = \{0,1\}$
- 代表：zkSNARKs, zkSTARKs, bulletproofs, …
- 应用：zkRollup, zkEVM, zk合约验证

交互式零知识证明 Interactive Zero-Knowledge Proofs (IZK)

- $\Pi(P^{(w,x)}, V^{x}) = \{0,1\}$
- 代表：Quicksilver, Mystique, …
- 应用：链下zk验证，zk应用

PADO

形式化TLS协议

Protocol TLS 1.2

Inputs. A client $\mathcal{C}$ and a server $\mathcal{S}$ hold the following inputs:

- *Personal inputs:* $\mathcal{C}$ has a query template Query and a private input α for Query. $\mathcal{S}$ holds a secret ke sk_S and a certification cert_S involving a public key pk_S.
- *Common inputs:* The common inputs except for F_x are chosen by $\mathcal{C}$ or $\mathcal{S}$ in the handshake phase.
  - Let F_x be a function mapping an elliptic-curve point to its x-coordinate.
  - Let H be a cryptographic hash function and PRF be a pseudorandom function.
  - $\mathsf{SIG} = (\mathsf{Sign}, \mathsf{Verify})$ is a signature scheme defined by cert_S, where the key-generation algorithm i omitted, Sign is the signing algorithm used to generate signatures, and Verify is the verificatio algorithm that outputs 0 (reject) or 1 (accept).
  - $(\mathbb{G}, p, q, G)$ is an elliptic-curve group, where p is a prime defining the base field that coordinate locate in and G is a generator with a prime order q.
  - $\mathsf{stE} = (\mathsf{Enc}, \mathsf{Dec})$ is a stateful AEAD scheme, where the key-generation algorithm is omitted.

Handshake protocol execution.

1. *Client request:* $\mathcal{C}$ samples $r_C \leftarrow \{0,1\}^{256}$ and sends $\textsc{req}_C := r_C$ to $\mathcal{S}$.
2. *Server response:* $\mathcal{S}$ performs the following steps:
   (a) Sample $r_S \leftarrow \{0,1\}^{256}$ and $t_S \leftarrow \mathbb{Z}_q$, and compute $T_S := t_S \cdot G$.
   (b) Run $\sigma_S \leftarrow \mathsf{Sign}(\mathsf{sk}_S, r_C \| r_S \| T_S)$, and then send $\textsc{res}_S := (r_S, T_S, \mathsf{cert}_S, \sigma_S)$ to $\mathcal{C}$.
3. *Client response:* If cert_S is invalid or $\mathsf{Verify}(\mathsf{pk}_S, r_C \| r_S \| T_S, \sigma_S) = 0$, $\mathcal{C}$ aborts. Otherwise, $\mathcal{C}$ sample $t_C \leftarrow \mathbb{Z}_q$ and computes $T_C := t_C \cdot G$, and then sends $\textsc{res}_C := T_C$ to $\mathcal{S}$. Both parties compute:
   (a) $\mathcal{C}$ computes $\mathsf{pms} := \mathsf{F}_x(t_C \cdot T_S) \in \mathbb{Z}_p$, and $\mathcal{S}$ computes $\mathsf{pms} := \mathsf{F}_x(t_S \cdot T_C) \in \mathbb{Z}_p$.
   (b) $\mathcal{C}$ and $\mathcal{S}$ compute $\mathsf{ms} := \mathsf{PRF}_{384}(\mathsf{pms}, \text{“master secret”}, r_C \| r_S) \in \{0,1\}^{384}$.
   (c) They compute $(\mathsf{key}_C, IV_C, \mathsf{key}_S, IV_S) := \mathsf{PRF}_{448}(\mathsf{ms}, \text{“key expansion”}, r_S \| r_C) \in \{0,1\}^{448}$ with $\mathsf{key}_C, \mathsf{key}_S \in \{0,1\}^{128}$ and $IV_C, IV_S \in \{0,1\}^{96}$.
4. *Client finished:* Let $\textsc{h}_C$ be a header specifying the sequence number, version and length of a plaintext, and ℓ_C be the target ciphertext length. $\mathcal{C}$ performs the following:
   (a) Compute $\tau_C := \mathsf{H}(\textsc{creq} \| \textsc{sres} \| \textsc{cres})$ and $\textsc{ufin}_C := \mathsf{PRF}_{96}(\mathsf{ms}, \text{“client finished”}, \tau_C) \in \{0,1\}^{96}$.
   (b) Initialize $(\mathsf{st}_e^C, \mathsf{st}_d^C) := (IV_C, IV_S)$, and run $\textsc{fin}_C \leftarrow \mathsf{stE.Enc}(\mathsf{key}_C, \ell_C, \textsc{h}_C, \textsc{ufin}_C, \mathsf{st}_e^C)$. Then, send $(\textsc{h}_C, \textsc{fin}_C)$ to $\mathcal{S}$.

   $\mathcal{S}$ initializes $(\mathsf{st}_e^S, \mathsf{st}_d^S) := (IV_S, IV_C)$, and runs $\textsc{ufin}_C \leftarrow \mathsf{stE.Dec}(\mathsf{key}_C, \textsc{h}_C, \textsc{fin}_C, \mathsf{st}_d^S)$. Then, $\mathcal{S}$ checks the validity of $\textsc{ufin}_C$ using ms, and aborts if the check fails.
5. *Server finished:* Let $\textsc{h}_S$ be a header and ℓ_S be the target ciphertext length. $\mathcal{S}$ does the following:
   (a) Compute $\tau_S := \mathsf{H}(\textsc{req}_C \| \textsc{res}_S \| \textsc{res}_C \| \textsc{ufin}_C)$ and $\textsc{ufin}_S := \mathsf{PRF}_{96}(\mathsf{ms}, \text{“server finished”}, \tau_S) \in \{0,1\}^{96}$.
   (b) Run $\textsc{fin}_S \leftarrow \mathsf{stE.Enc}(\mathsf{key}_S, \ell_S, \textsc{h}_S, \textsc{ufin}_S, \mathsf{st}_e^S)$, and then send $(\textsc{h}_S, \textsc{fin}_S)$ to $\mathcal{C}$.

   $\mathcal{C}$ runs $\textsc{ufin}_S \leftarrow \mathsf{stE.Dec}(\mathsf{key}_S, \textsc{h}_S, \textsc{fin}_S, \mathsf{st}_d^C)$ and checks $\textsc{ufin}_S$ using ms, and aborts if the check fails.

Record protocol execution with one-round session.

6. *Client query:* Let $\textsc{h}_Q$ be a header and ℓ_Q be the target ciphertext length. $\mathcal{C}$ and $\mathcal{S}$ execute as follows:
   (a) $\mathcal{C}$ runs $Q := \mathsf{Query}(\alpha)$ and $\textsc{enc}_Q \leftarrow \mathsf{stE.Enc}(\mathsf{key}_C, \ell_Q, \textsc{h}_Q, Q, \mathsf{st}_e^C)$, and sends $(\textsc{h}_Q, \textsc{enc}_Q)$ to $\mathcal{S}$.
   (b) $\mathcal{S}$ runs $Q \leftarrow \mathsf{stE.Dec}(\mathsf{key}_C, \textsc{h}_Q, \textsc{enc}_Q, \mathsf{st}_d^S)$, and generates R according to Q.
7. *Server response:* Let $\textsc{h}_R$ be a header and ℓ_R be the target ciphertext length. $\mathcal{C}$ and $\mathcal{S}$ do the following:
   (a) $\mathcal{S}$ runs $\textsc{enc}_R \leftarrow \mathsf{stE.Enc}(\mathsf{key}_S, \ell_R, \textsc{h}_R, R, \mathsf{st}_e^S)$, and then sends $(\textsc{h}_R, \textsc{enc}_R)$ to $\mathcal{S}$.
   (b) $\mathcal{C}$ gets $R \leftarrow \mathsf{stE.Dec}(\mathsf{key}_S, \textsc{h}_R, \textsc{enc}_R, \mathsf{st}_d^C)$.

Figure 12: **Handshake and record protocols for TLS 1.2.**

PADO

MPC-TLS

- **Handshake**
  - P和V各自选取随机数，一起计算pms （涉及share conversion）；
  - P, V 执行一个garble-then-prove的两方协议，各自能获得handshake流程中的每一步输出参数 (headers, IVs, tags, ciphers)；最终获得session key shares;
- **Record**
  - P和V执行一个两方协议，产生TLS请求Q(含密文和tag) , 发送给S;
  - P接收S的TLS响应R（含tag）, 将(Q,R)发送给V;
  - V将自己的随机数发给P, 后者恢复session key, 包括其他共享的中间值；
  -

1. $\mathcal{P}$ samples and sends REQ_C to $\mathcal{S}$ and gets back RES_S.
2. $\mathcal{P}$ forwards (REQ_C,RES_S) to $\mathcal{V}$, who sends $t_\mathcal{V} \cdot G$ to $\mathcal{P}$. $\mathcal{P}$ picks $t_\mathcal{P}$ and sends $(t_\mathcal{P} + t_\mathcal{V}) \cdot G$ to $\mathcal{S}$. Then $\mathcal{V}$ and $\mathcal{P}$ run an EC-to-Field conversion based on OLE with error so that two parties obtain additive shares of pms.
3. Two parties use Garble-Then-Prove technique so that they obtain 1) h_C, h_S and z_0's as XOR shares, 2) values of IV_C, IV_S, intermediate public value in HMAC, UFIN_C, UFIN_S, and the encryption of UFIN_C. 3) ms, key_S, key_C in garbled format,
4. Two parties compute the tag based on OLEe over $\mathbb{F}_{2^{128}}$. $\mathcal{P}$ assembles FIN_C and sends to $\mathcal{S}$.
5. $\mathcal{P}$ forward (H_S, FIN_S) to $\mathcal{V}$.
6. $\mathcal{V}$ and $\mathcal{P}$ use Garble-Then-Prove to learn the ciphertext that encrypts $\mathcal{P}$'s query and XOR shares of the corresponding z_0. Two parties use OLEe to compute the tag $\mathcal{P}$ sends the the ciphertext and tag to $\mathcal{S}$; $\mathcal{S}$ returns ciphertext and tag of the response to $\mathcal{P}$, who forwards them to $\mathcal{V}$.
7. $\mathcal{V}$ sends $t_\mathcal{V}$ to $\mathcal{P}$ who checks that it is consistent with $t_\mathcal{V} \cdot G$ received earlier. $\mathcal{P}$ then computes $t_\mathcal{P} + t_\mathcal{V}$, and recovers all values in the execution of TLS, including all values revealed previously. If any value is wrong, $\mathcal{P}$ aborts.
8. $\mathcal{V}$ now holds commitments to $\mathcal{P}$'s share of pms and values revealed as XOR shares earlier. $\mathcal{P}$ proves to $\mathcal{V}$ in ZK that these commitments are consistent with the values revealed to $\mathcal{V}$ based on the TLS specification.

核心思想: 让P和V模拟一个TLS client, 拼装相关的协议消息

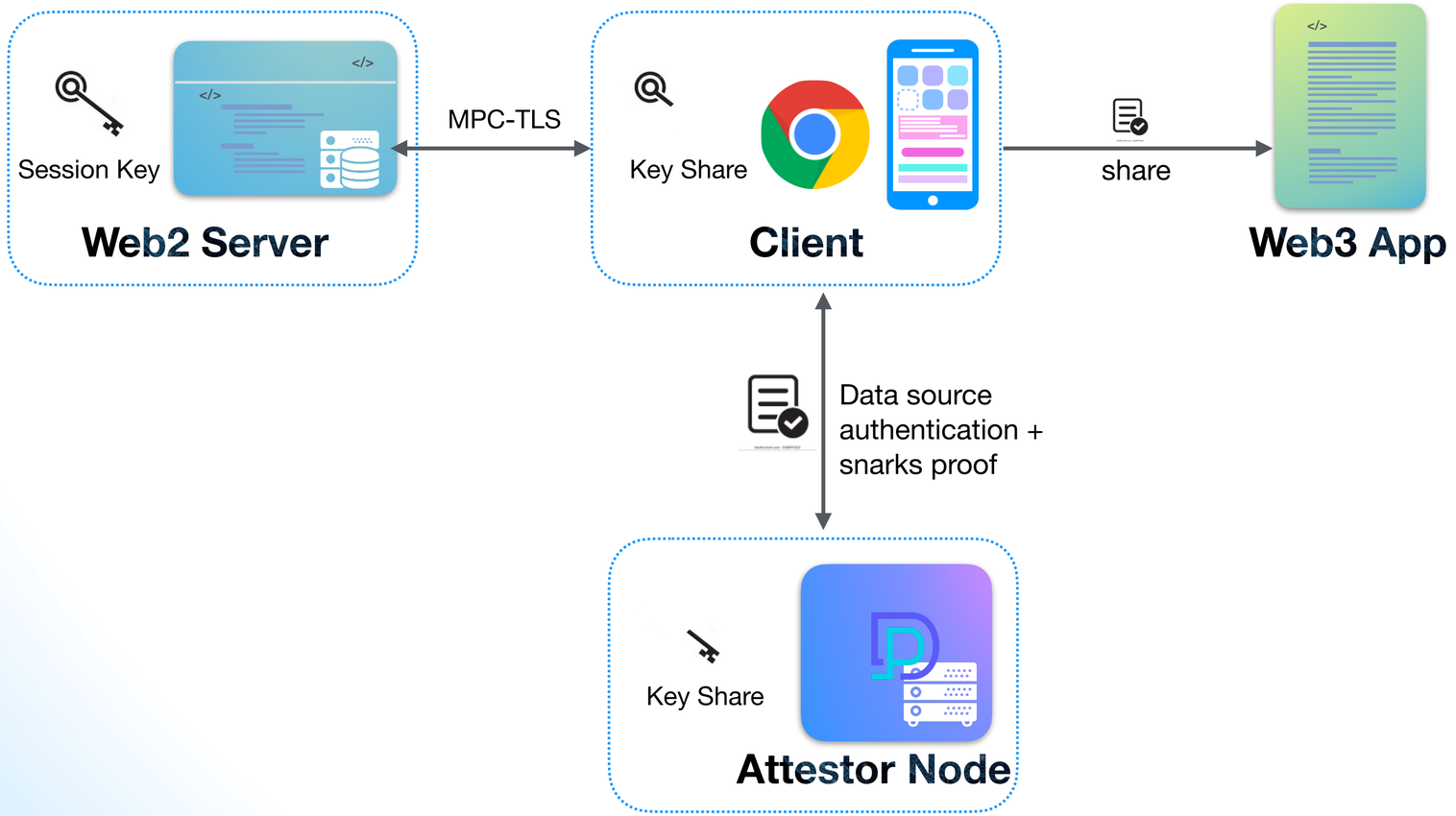
PADO

IZK

- **Handshake**
  - P和V各自选取随机数，一起计算pms （涉及share conversion）；
  - P, V 执行一个garble-then-prove的两方协议，各自能获得handshake流程中的每一步输出参数 (headers, IVs, tags, ciphers)；最终获得session key shares;
- **Record**
  - P和V执行一个两方协议，产生TLS请求Q(含密文和tag) , 发送给S;
  - P接收S的TLS响应R（含tag）, 将(Q,R)发送给V;
  - V将自己的随机数发给P, 后者恢复session key, 包括其他共享的中间值；
- **Post Record （真实性+隐私保护）**
  - P与V执行IZK协议，P向V证明V持有的commiments与V给P的秘密值是一致的；
  - 基于IZK电路，P向V证明数据满足一定的条件;
  - P, V可以将基于IZK的commitments (IT-MACs)通过一些方式转换为snarks-friendly commitments (Pedersen Comm), P计算NIZK证明——数据满足一定的条件；

PADO

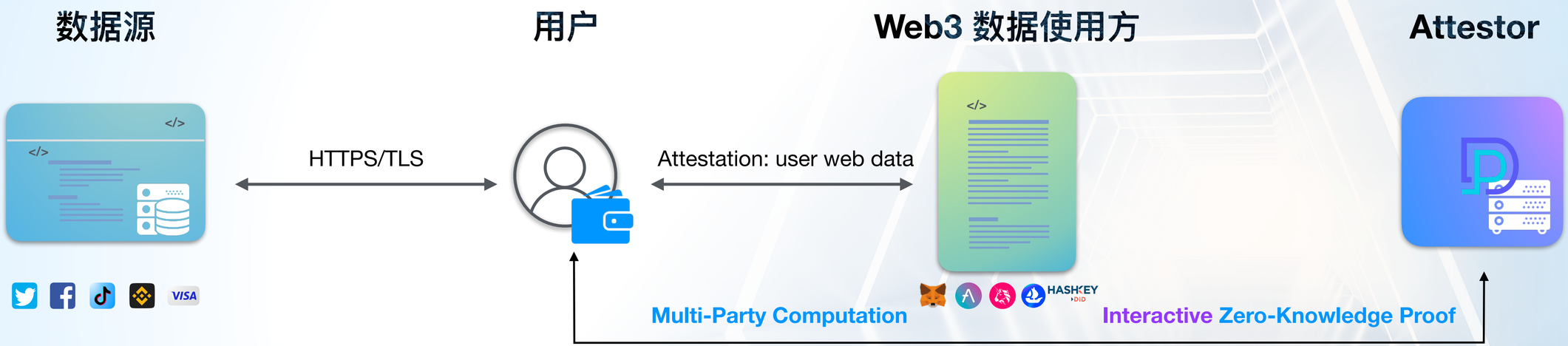
基于MPC-TLS 和 IZK 的TLS数据分享



- TLS通道建立
  - Client, Attestor 执行2PC协议，模拟 TLS Client, 各自获得一片 session key share；
- TLS Query阶段
  - Client, Attestor 执行2PC协议，生成 encrypted query Q ;
  - 收到encrypted response R后，client会获得另一片的key share，来恢复出response明文。并且client 需要以iZK的方式证明 (Q,R) 的来源正确和消息一致性，以及业务数据满足特定的条件, Attestor签名背书；

PADO

Web2到Web3的数据分享模式



用户隐私友好

- 合规，如the right of data portability
- 选择性披露/数据最小化原则

开放式证明者

- 基于MPC/IZK的黑盒式验证
- 无法触碰用户数据
- 支持第三方部署

EAS兼容

- Off-chain web data Attestors
- 扩展Schemes，支持更丰富的链下数据应用模式

PADO

3

与NIZK的结合

PADO

结合NIZK

- 为什么要结合 NIZK？
  - 我们需要公开（链上）可验证性；
  - NIZK 处理算术 (arithmetic) 逻辑 有优势；
- 怎么结合 NIZK?
- Commitment conversion
  - P 向 V 证明 (IZK): $\{com = Comm(m, r) \wedge C = Enc(K, m)\}$ //Comm是一个snarks-friendly的承诺方案
  - V 确认后进行签名$\sigma = Sign(sk, com)$;
  - P 产生一个 NIZK = $\{1 = Verify(pk, com, \sigma) \wedge com = Comm(m, r) \wedge f(m) = 1\}$ // f是一个算术电路

PADO

IZK v.s.NIZK

- 两者互补, 各有所长

	证明时间	验证时间	证明长度	内存	验证者类型	可信设置
IZK	快	快	长	极少	指定验证人	无
NIZK	慢	非常快	短	大	公开	可能需要

PADO

4

基于PADO的信用协议建设

应用场景

早期应用

- 反女巫
- 人类证明
- ...

+ 其他集成设施

成熟应用

- 金融
- 信用
- 物理访问控制
- 治理

GameFi

- 传统游戏平台用户引流
- 链上电竞

DID/SBT

- 丰富用户身份数据维度
- 可验证的链下物理身份

资格/治理

- 职业技能证明
- 活动证明
- 开发者证明

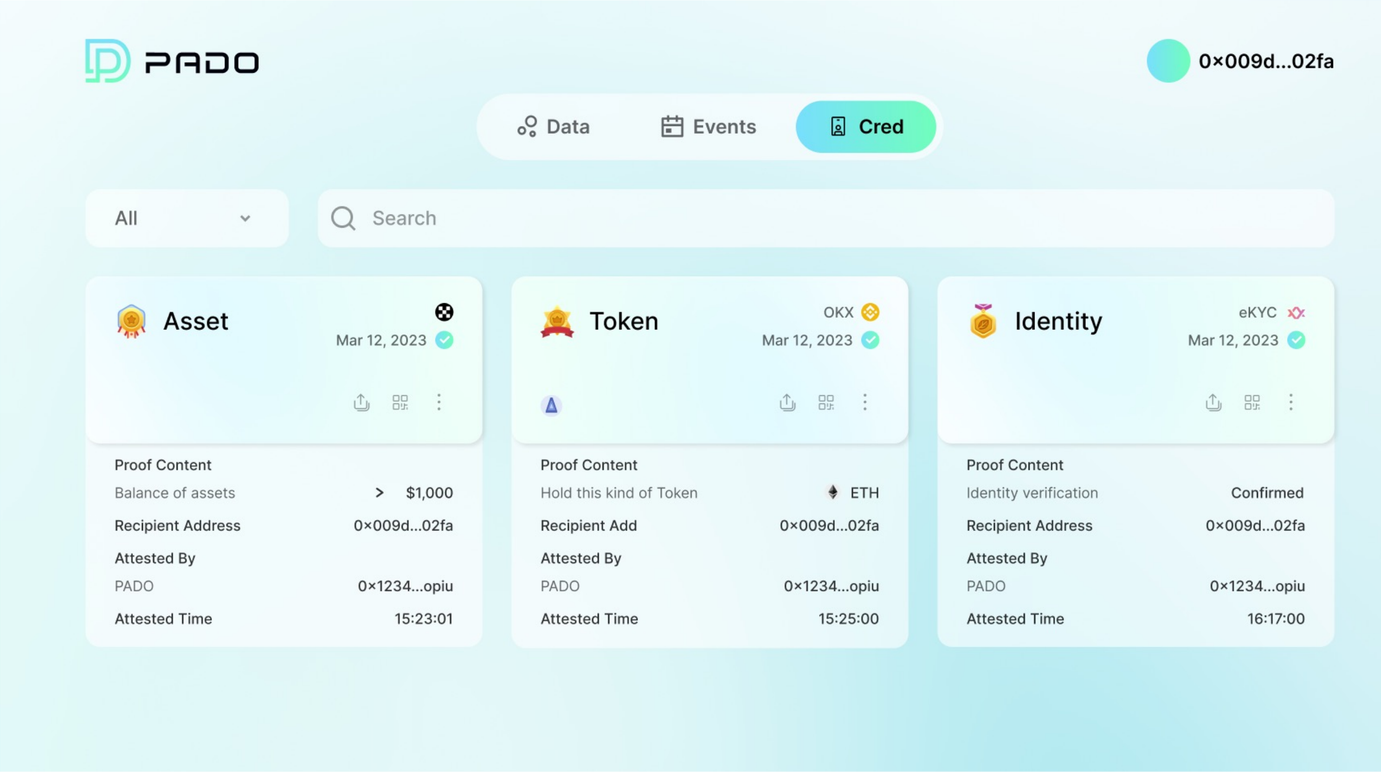
Defi

- 无抵押借贷
- zkKYC
- 合格投资人

PADO

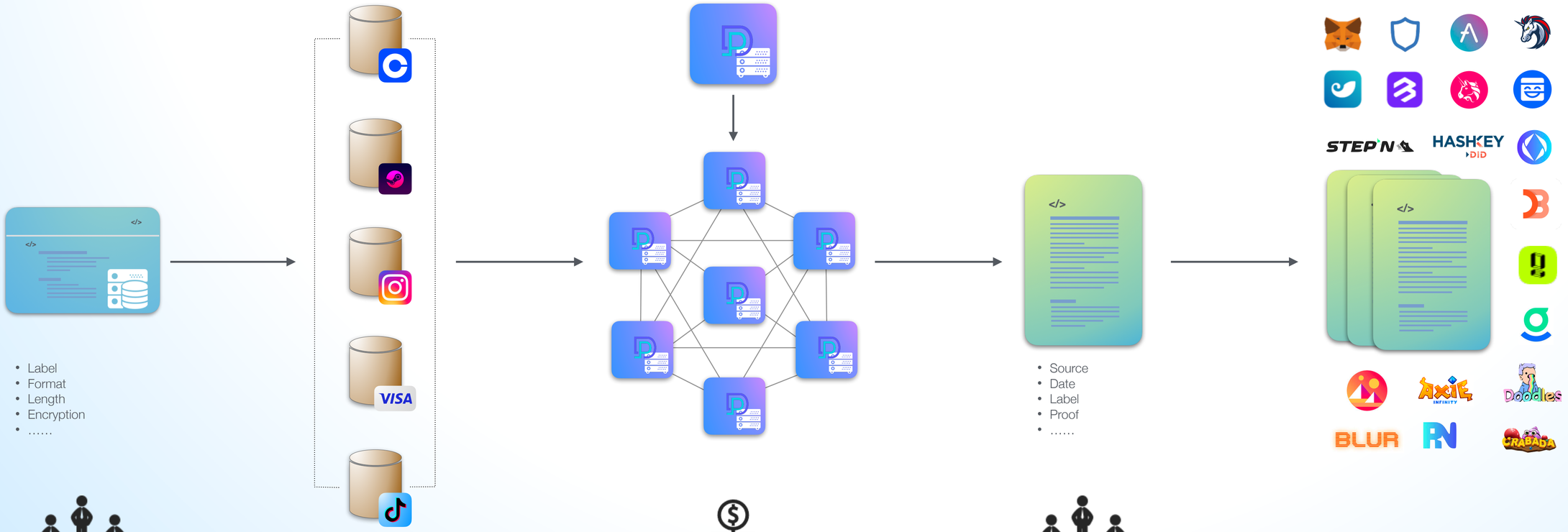
基于PADO的去中心化信用协议

- 用户自主获取链下数据
- 通过PADO协议，用户在本地创建单一信用凭证（credit credentials）
- 用户可以分享给任意方（dApps, institutes, off-chain entites）
- 信用凭证可组合；
- 信用建模/信用分计算 （zkVM/ zkML）



社区化开发

数据源模版	数据源	Attestors	应用侧模版	Web3 应用协议



社区开发者　免费/商业型　去中心化安全性　应用开发者　Game/Social/DID/…

PADO

PADO

算法协议：https://eprint.iacr.org/2023/964