

HacdsDeFi 白皮书

2026-10-21 第二版

写在前面

hacds 是 Hacash 主网上的一个链上资产，serial 2000。

它跑在 Hacash 的第一份 DeFi 合约上。Hacash 主网的 HVM（图灵完备智能合约）在区块高度 765,432 激活，

而这份合约是主网上第一个把 DeFi 四件事——兑换做市、质押、借贷、金库——整套做进单份合约的实现。

我们的部署清单当时就是按“主网首个图灵完备智能合约”写的。预言机、索引器、钱包也是我们自己写的，跑在自己的服务器上。

本文分四部分：上一版的问题，这一版的设计，四个特点，当前状态。

上一版那份《Hacds 铭文开发白皮书》已经作废。那套方案我们做完过、上线过，也停掉了。

一、上一版

1.1 原来的方案

上一版把 hacds 定义成刻在 HACD 钻石上的铭文。方案分四块。

发币靠铭文。每颗 HACD 刻一条 Hacds 铭文，一颗钻石只有一条有效。普通钻石刻一张给 10000 个 hacds，特殊钻石给多少（爱心 50 万、四连字母 30 万、异形 20 万）由社区投票决定。

交易靠插件。去中心化交易所做成一个叫 MoneyNex 的插件钱包，用户装插件、用插件的钱包进出场。

上所当目标。白皮书里写的第一个上所目标是 xeggex.com。

检索靠扫链。每个客户端自己扫全网铭文，第三方服务器可以另外提供检索。

1.2 我们做完了，也上所了

这些不是纸面方案。铭文系统做出来了，MoneyNex 插件跑起来了，hacds 先后上了 xeggex.com 和 nonkyc.io 两家交易所。

然后停牌了。

1.3 问题出在哪

症结在第一条，其余三条都是它的后果。

币刻在钻石上，就动不了。持有 1 万个 hacds 意味着持有一批特定的 HACD，转账要连钻石一起转。一颗钻石只能刻一条铭文，总量因此被钻石数量锁死，拆不开，也拆不细。拆不开的东西当不了货币，在交易所里也做不了正常的充提和深度。

价格没有锚。发行比例靠投票，等于每类钻石值多少个 hacds 随时可以被投票改掉。交易所拿不到可复现的换算依据，做市商给不出稳定报价，深度也就起不来。

交易环节押在别人身上。去中心化交易所是第三方插件，插件停更就是交易所停摆。中心化交易所更直接，成交和流通都在 xeggex 和 nonkyc 手里。

没有退路。这两家所是全部的流通渠道，它们一停，链上再没有别的地方能交易。

连起来看：币不能自由流通，就没法定价；没法定价，就没法做市；没法做市，就只能依赖交易所；交易所一停就断。

上不上所不是重点。上了所，设计上的毛病也不会消失，只是推迟到停牌那天才暴露。这是这一版改设计的直接原因。

1.4 这一版改了什么

只做了一件事：把币从钻石上拿下来。

hacds 改成标准链上资产，不依附任何钻石，能自由拆分和转账。金融逻辑从插件和交易所搬进合约。流通不靠上所，链上 AMM 自己就能交易，上所以后只是可选项，不再是活下去的前提。

二、这一版的设计

2.1 发行量怎么来的

hacds 的总量不是拍的，是从全网钻石算出来的。

每颗 HACD 按 HIP-5 多维评分（形状、样式、颜色、命名、编号），1 分换 1 个 hacds（合约里 `SCORE_SCALE = 1e8`，也就是 1 分 = 1e8 atoms）。总额取全网钻石的精确均值：

均值评分 × HACD 总发行量

= 3.55440306 × 16,777,216

= 59,632,987 个 hacds

= 5,963,298,700,000,000 atoms

这个数谁都能自己复算：拉一遍浏览器接口，逐颗算分，结果一样。上一版缺的就是这个：一个不依赖任何人拍板的基准。

评分在链下算，算完签名；合约收到证书后在链上重算一遍才认。链下省 gas，链上保正确，两边用同一份算法，结果必然一致。我们在 600 多颗真实钻石上复算过，对得上。

2.2 资产规格

项	值
serial	2000
ticket / name	HACDS / hacds
decimal	8
总供给	5,963,298,700,000,000 atoms

项	值
issuer	VKNwXFFMCEn9XfMrhyHhd5rHqBnEyM3m1 (合约)
发行成本	8 HAC (协议费, 随出块奖励浮动)

链上 #1028 的 ticket 也是 HACDS (它的 name 是 "HACD Fragments", 别人的资产)。协议不检查 name 和 ticket 是否重名, 只看 serial 有没有被占。认 serial 2000。

三、四个特点

3.1 开源

合约 (fitsh 源码)、预言机、索引器、钱包, 四部分源码都会公开。

DeFi 协议里跑的是别人的钱, 用户能自保的唯一办法是自己读得懂代码、算得清账。上一版的钱包是第三方插件, 用户只能选择信或不信; 这一版把逻辑摆在明面上。

代码公开之后, 有两件事可以直接做:

- 评分可独立复算。谁都能拉一遍全网钻石、按同一套算法重算评分, 看发行量对不对。我们自己在 600 多颗钻石上做过。
- 合约行为可审计。管理员能做什么、不能做什么, 函数清单里写着, 不必听我们怎么说。

3.2 去中心化

这一条要讲边界。笼统说"完全去中心化"没有意义。

已经做到的:

规则和执行都在链上。AMM 的定价、借贷的放行条件、清算的门槛、手续费的分账, 全是合约代码。参与者按同一套规则交易, 我们不在中间, 也插不进手。

合约不可升级, 没有后门。整个合约没有自毁函数, 没有升级函数, 没有针对单个用户的冻结函数。部署上去是什么, 以后就一直是。

管理员动不了用户的钱。这一点可以从函数清单直接验证: 管理员能调的六个函数里, 没有提款, 没有改余额, 没有改抵押, 没有改债务, 没有转移任何资产的能力。它改的只是参数。

管理员权限有硬上限。不是靠承诺, 是靠代码里的判断:

操作	上限
<code>set_fee</code> 改 AMM 费率	≤ 300 bps (3%)
<code>set_loan_rate</code> 改借贷利率	≤ 2000 bps (20%)
<code>set_liquidate_bonus</code> 改清算奖励	≤ 1000 bps (10%)

抵押率不是参数。 `COLL_RATIO` 是合约常量, 写死为 2, 管理员改不了。所谓"抵押率会不会被偷偷调低"这个问题不存在。

管理员不可更换。部署时由 `init_admin` 固化, 合约里没有 `set_admin`, 地址改不了。

参与无需许可。兑换、做市、质押、借贷，任何人都能来，没有白名单，没有审批，不需要我们同意。金库地址也是写死的常量。

资产自己拿得住。hacds 是标准链上资产，放在用户自己的地址里。我们不托管，也冻结不了。

还没做到的，一并说清楚：

预言机目前是单点，由我们运行。它决定不了用户的资产，合约会在链上重算评分；但兑换能不能用由它决定，它下线，兑换就停。

管理员手里有 `set_paused`，可以全局暂停合约。他停不了用户的资产，也拿不走任何人的钱，但能让协议暂时不接受操作。

这两条就是当前去中心化程度的实际边界。要消掉需要改合约设计（多预言机、时间锁、多签），和下一次合约更新一起做。

3.3 安全

分两头：用户的钱怎么保管，协议的规则怎么保证。

私钥不触网。钱包分冷热两端，离线机构建交易并签名，联网机只做查询和广播。私钥只在主进程内存里出现过一次，构建时通过环境变量注入子进程，不落盘。这套流程能成立是因为 Hachash 的交易没有 nonce、没有时间戳过期，离线签好的交易放很久再广播依然有效。

签名在本地做。交易由随包的 Rust 程序在本地签，不经过任何节点。节点只用来翻译、校验、广播。节点的 `/util/transaction/sign` 接口要求把私钥通过 HTTP 发过去，我们没用它。

钱包文件是加密的。口令用 `scrypt` ($N=2^{15}$, $r=8$, $p=1$) 派生，加密用 AES-256-GCM，AAD 绑定格式版本。

界面进程没有系统权限。渲染进程开了 `contextIsolation`、关了 `nodeIntegration`，只能通过 preload 的白名单调主进程。渲染层即使拿到恶意内容，也碰不到文件系统和私钥。

不盲信预言机。预言机签的证书只是传递凭证，合约收到后会用自己的评分引擎在链上重算一遍再比对。预言机被攻破也伪造不了分数。

资产守恒。链上恒等式：用户持有的量加合约持有的量等于总供给。所有操作都保持这个式子，谁都能随时核对。

防呆也做了。单桶 gas 限制导致一笔只能存一颗钻石，钱包里做了硬拦截，勾多颗自动拆笔；批量完成后反向按钮有冷却，防止刚赎回又误点存入。

3.4 DeFi 完备

上一版的"金融"只有两件事：发币，和把币挂到交易所。没有金融原语，不能做市，不能抵押，不能借，收益只能靠涨。

这一版补齐了，一个合约里有四块：

模块	内容
AMM	HAC 和 hacds 双向兑换。恒定乘积 $x \times y = k$ ，手续费 0.5%，LP 50% / 质押池 30% / 金库 20%
质押	LP 份额锁进质押池分手续费，按 rpu 定点分配，惰性结算，不领也不会丢

模块	内容
借贷	用 hacds 做抵押借 HAC。抵押率 50%，年化 5%，可以还款，抵押率过低会被清算
金库	手续费沉淀在这里，定期清扫

另有两样工具。

原子交易。多个操作并进同一笔交易，要么全成要么全败，协议上限 200 个动作。比如"存入钻石 + 立刻领 hacds"可以是一笔。

清算机制。借贷有明确的清算线和清算奖励，任何人发现抵押率低于 200% 都能执行清算，不需要我们参与。为了让大家找得到可清算的账户，我们写了索引器（见 4.3），因为合约本身枚举不出借款人。

利息用惰性计算：不写定时任务，不依赖 keeper，有人跟合约交互时按块高差把利息补上。省 gas，也不会因为没人维护就停摆。

四、系统组成

整体分四层：链上的合约和资产，服务端的节点、预言机、索引器，客户端的钱包，以及冷热分离的使用方式。



技术依据：Hacash 交易无 nonce、无时间戳过期 → 离线签好的交易长期有效，联网后再广播即可。

⑤ 接口契约（钱包 ↔ 服务端）

接口	用途	要点
GET /cert?name=X (:8123)	HACD 评分证书	126B；兑换/赎回必需
sandboxcall (:8082)	读合约状态	每 IP 并发 ≤ 2，需串行
GET /watchlist (:8124)	借款人台账 + 健康度	一次性快照，替代本地扫链
POST /submit/transaction	广播	ret:0 仅代表进内存池；出块 300s

口径唯一出处：wallet-app/设计文档.md §0 事实总纲 | 单位约定：HAC 用 HAC、hacds 全端用 raw atoms（不做 ÷1e8 换算） | VM 三桶上限：compute 18,009 / resource 6,100 / storage 111,911

4.1 合约

合约用 fitsh 写在 Hacash VM 上。四个模块的规则前面讲过，这里补几条实现上的取舍。

抵押品用的是 due 配额，也就是钻石存进合约换来的权利，不是钻石本身。用户不必为了借钱把钻石锁住。

抵押品的估值直接用 AMM 的即时池价（ $coll_to_hac = coll \times amm:hac / amm:ds$ ），不引入外部价格源，省掉一整套喂价基础设施，也少了被喂价攻击的面。

金库地址写死在合约里，改不了。

合约没有暴露读配置的接口，所以管理界面看不到当前的费率和利率。这是有意的：为了回显去升级一份已经上线的合约不划算。

有个硬限制要提前知道。Hacash VM 的 gas 分三个桶，各自独立，跟总预算 gas_max 无关：

桶	上限
compute	18,009
resource	6,100
storage	111,911

resource 那条直接决定了单笔存入最多 1 颗 HACD。每颗钻石的验签要链上重算评分，大约吃掉 3,000 resource，两颗就是 6,101，超过 6,100 一点点。这跟 gas_max 和手续费都没关系，调大也没用，只能拆笔。它是共识参数，改它等于分叉。

4.2 预言机

合约要验证钻石值多少分，但评分里有 SHA3 和五维打分，在链上全量跑太贵。办法是链下算、链上验。

调用 GET /cert?name=KWTVSH，服务端取钻石基因、派生 visual_gene、算五维分，打包成 126 字节的证书（消息 29 字节加签名 97 字节），用自己的私钥签好返回。合约收到后链上重算一遍评分做比对，签名用来防止伪造。

预言机跑在 43.251.101.101:8123，只依赖 Python 标准库（hashlib.sha3_256 是内置的）。私钥只在服务端，客户端只拿证书。

合约里的 `cfg:oracle` 只认一个固定地址。用户自己跑一个预言机，签出来的证书合约不认。这是有意设计的，否则评分就能随便伪造。

4.3 索引器

这里有个绕不开的问题：合约枚举不出借款人。借款账目是按地址拼键存的（`loan:debt:<地址>`），合约里没有列表，不写日志，节点也没法按前缀去列合约的 KV。所以“现在谁可以清算”这个问题，链上查不出来。办法是在链下扫链建台账：扫区块，挑出命中了借贷函数 selector 的合约调用，取其交易的 main 地址当借款人，增量记 lastScanned，台账落盘。要快照时，对台账里每个地址查一遍 `loan_status`。

钱包只需要调一个接口：`GET /watchlist`，拿回概况和带健康度的台账。其余 `/snapshot`、`/health`、`/sync`、`/watch/add`、`/watch/remove` 是辅助。

索引器必须放服务端。实测扫链速度是串行 2.6 块/秒、并发 4 是 6.6 块/秒，回溯 500 块要 15 到 75 秒、上千次 HTTP 请求。手机做这个，流量和电量都受不了。放到服务端后，钱包只发一次请求。附带两个好处：Python 标准库有 SHA3（Electron 的 Node 运行时没有，见下面钱包那节）；台账只有一份，PC 和手机看到的是同一个。

4.4 钱包

钱包是一个 Electron 应用，冷热共用：离线机构建和签名，联网机查询和广播。

界面用原生 HTML/CSS/JS 写，没有前端框架。里面有资产总览、协议状态、借贷状态常驻卡、AMM 实时报价、HACD 收藏、离线签名、原子交易、清算台、管理员页。文案做了中英土三语，479 条。

主进程负责 IPC 白名单、钱包 keystore、手续费迭代估算，以及和三个服务端之间的 HTTP。

交易构建和签名交给随包的 Rust 程序 `hacds_txbuild`，八种模式。不重写是因为这块逻辑复杂（200 多个动作、Type-3 线格式、手续费迭代），重写风险高，而且这套代码已经在跑。

钱包里有几处值得说：

- 借贷页在输入框下面实时显示可借上限和这笔需要的抵押，首页另有一张常驻的借贷状态卡。以前用户直接点借出，只会拿到合约的 return 40，看不出问题在哪。
- HACD 多选后批量操作，自动拆成每笔一颗。
- 离线签名：冷端构建加签名，出分片二维码或存文件，热端扫码广播。
- 原子交易：多个动作并进一笔。

五、新旧对照

	旧版	现在
代币形态	HACD 上的铭文	标准链上资产（serial 2000）
供给	受钻石数量限制，一钻一铭文	一次性发行 59,632,987，可自由拆分
价值来源	社区投票	钻石评分算法，可复算
交易场所	MoneyNex 插件钱包	链上 AMM 合约
上所	上了 xeggex.com 和 nonkyc.io，后停牌	不需要上所即可交易；要上也凭标准资产接口

	旧版	现在
金融原语	无	AMM / 质押 / 借贷 / 金库
评分来源	客户端自算	预言机签证书 + 合约链上重算
数据检索	每个客户端扫全网铭文	服务端索引器，钱包调一次
钱包	第三方插件	自建 Electron 控制台
私钥	交给插件	本地自持，不触网
源码	不公开	计划全量公开
谁能改规则	插件作者、交易所	只有链上合约，管理员仅能改有上限的参数

六、现状

已经上线的东西：

项	位置
合约	VKNwXFFMCEn9XfMrhyHhd5rHqBnEyM3m1
hacds 资产	serial 2000, 供给 5,963,298,700,000,000 atoms
全节点	43.251.101.101:8082
预言机	43.251.101.101:8123
索引器	43.251.101.101:8124
钱包	HacdsDeFi-Setup-0.1.0-x64.exe / HacdsDeFi-0.1.0-x64.exe

验证过的几件事：

- 评分算法跟预言机对了 152 颗，跟合约链上引擎对了 610 颗，全部一致
- 单笔存 1 颗，广播成功
- 单笔存 2 颗，报 `OutOfGas(31): resource ... used 6101 > limit 6100`，跟预期一致
- 批量转入真机跑 3 颗，3/3 成功；同一时间戳构建的多笔交易 hash 互不相同
- 打包自检 `RESULT = ALL OK`，八种引擎模式齐全
- 三语各 479 条词条，键集合一致

七、已知限制

有几件事不是没做，是权衡之后先这样。写在这里，免得后面的人踩。

预言机是单点。它由我们持有，地址写死在合约里，换不了。我们选了评分可信而不是评分离散：分数客观、可复算，合约还会链上重算兜底，代价是这台机器必须一直在线。

管理员可以暂停合约。他动不了资金，但能让协议暂时不接受操作。参数权限也集中在管理员手里，虽有数值上限。

单笔存入限 1 颗。原因是前面 gas 那节讲的。大额操作要拆成多笔，界面上会提示。

出块 300 秒。广播成功不等于上链，`ret:0` 只是进了内存池。钱包按出块节奏补刷几次（15、45、90、180、320 秒）。

利息不单独显示。合约惰性计息，计息的起点没有对外接口，算不出当前的应付利息，只在还款时显示链上结算的结果。

前两条是去中心化程度的边界，其余三条来自已经上线的合约。要改就得重新部署合约、重新发资产，代价远大于收益，所以先留着，写清楚比藏着好。

八、接下来

Android 端在做，Kotlin 加 Compose，复用同一份 Rust 引擎保证交易字节一致。Rust 交叉编译到 aarch64 已经打通。

去中心化这块，下一步是消掉上一节的前两条：预言机做多实例，关键参数加时间锁。这两项都要动合约设计，和下一次合约更新一起做。

其余在排队：提交前的参数即时校验、出块后自动刷新余额、交易历史、批量操作。

不再考虑的方向：本地拉起预言机（合约不认）、多节点广播、自动更新。

附录 A：地址与参数

项	值
部署者 / 管理员	15fv6cNEapTNqYhigMPPr38pC1XRKn3Wh5q
合约	VKNwXFFMCEn9XfMrhyHhd5rHqBnEyM3m1
金库	1FbwjELGMmaomNQq3ZZ6hdgQ17GRGwzAzK
预言机账户	1H3sFZ53HzNVUVN97mjpC9JT8s2PqKMyRu
hacds serial	2000 (decimal 8, ticket HACDS)
SCORE_SCALE	1e8 (1 分 = 1 个 hacds)
COLL_RATIO	2 (50% 抵押率, 清算线 200, 合约常量, 不可改)
AMM 手续费	0.5%, LP 50% / 质押 30% / 金库 20%
借贷利率 / 清算奖励	年化 5% / 2%
管理员参数上限	费率 3% / 利率 20% / 清算奖励 10%
证书长度	126 字节 (消息 29 + 签名 97)
VM 三桶上限	compute 18,009 / resource 6,100 / storage 111,911

项	值
出块目标	300 秒

附录 B：单位

hacds 是 decimal 8 的资产。链上那个整数是 atoms，写 UI 的时候很容易把单位和读数搞混。

钱包里的做法很简单：hacds 一律显示 atoms，不多做一层换算。余额、池子、抵押、输入框、报价，看到的都是 atoms，跟浏览器和交易所接口一个口径。输入框的标签写明 atoms，默认值给 100000000，让用户直接看到 1 个 hacds 对应多少 atoms。

要做个数换算时自己除 1e8。7000000000 atoms 就是 70 个 hacds。

三条规矩：

1. 链上、交易所接口、钱包，一律用 atoms，不换算法。消费方需要个数就自己除 1e8。
2. 别按除以 1e8 的读数去填输入框。用户看见 70 就填 70，实际只转了 70 atoms，差 1e8 倍。输入框旁边必须标 atoms。
3. 总供给 5,963,298,700,000,000 已经超过 JS `Number.MAX_SAFE_INTEGER` 的 66%，JS 里用 `BigInt`，Kotlin 里用 `BigInteger`，`Long` 不够。

HAC 是另一套：链上单位是 zhu (1 HAC = 1e8 zhu)，界面上一律用 HAC。所以

`add_liquidity(hac_amt, ds_amt)` 两个参数的量纲不一样，一个是 zhu，一个是 atoms。这是最容易犯 1e8 倍错误的地方，我们自己在首注时就犯过一次。