## مقدمه

مطالعه پیشینه پژوهش نشان داد که چالش‌های طراحی یک پروتکل رای‌گیری، عمدتا ناشی از این مساله هستند که پروتکل‌های رای‌گیری الکترونیک نیاز دارند به اهدافی دست بیابند که ذاتا متناقض هستند؛ یعنی رسیدن به

* گمنامی (پنهان بودن هویت رای‌دهنده) در عینِ نیاز به شناخته بودن فرد واجد شرایط (احراز هویت[[1]](#footnote-1) رای‌دهنده)؛
* حفظ حریم خصوصی (رمزگذاری) در عین نیاز به توانایی شمارش آرا (رمزگشایی)؛
* شمارش آرا (داشتن آگاهی از تمام آرای ثبت شده) و رعایت انصاف (پنهان بودن نتیجه رای‌گیری تا پایان فرآیند آن)؛ و
* نیاز به بازبینی فردی و عمومی بر پروتکل از طریق خودشمارندگی (تمرکز زدایی از نقش انحصاری برگزارکننده انتخابات)، در عین نیاز به انجام رویه‌های مجازشماری[[2]](#footnote-2) و کنترل دسترسی[[3]](#footnote-3) برای جلوگیری از اقدامات مخرب و دستکاری در روند انتخابات.

بنابراین، بسیاری از پژوهش‌های انجام شده در حوزه رای‌گیری الکترونیک به صورت نامتمرکز، دارای مشکلات زیر هستند:

1. امکان تبانی[[4]](#footnote-4)
2. مشکلات مقیاس پذیری[[5]](#footnote-5)
3. نداشتنِ بازبینی پذیری[[6]](#footnote-6)

## بیان مساله

این پژوهش یک سیستم مبتنی بر بلاک چین را برای افزایش اعتماد و امنیت در شبکه‌های هوشمند، با تمرکز بر تجارت انرژی بین خانوارهای مجهز به کنتورهای هوشمند و قابلیت‌های ذخیره انرژی، پیشنهاد می‌کند. این سیستم از قراردادهای هوشمند برای خودکار کردن تجارت انرژی و تضمین شفافیت استفاده می‌کند و یک محیط غیرمتمرکز و بدون اعتماد را برای تبادل انرژی همتا به همتا فراهم می‌کند

## شمای نظریِ پروتکل

در ابتدای این فصل به شمای نظری پروتکل می‌پردازیم. در این بخش، با استفاده از نمودار توالی[[7]](#footnote-7)، روند پروتکل توضیح داده شده است.

### فازهای پروتکل

نمودار توالی پروتکل را نشان می‌دهد که تمامی فازهای پروتکل را به نمایش گذاشته است. مخزن کد پروژه نیز موجود است تا در صورت نیاز به آن مراجعه شود.

با این حال، به صورت کلی، فازهای پروتکل به ترتیب زیر انجام می‌شوند:

1. راه‌اندازی

* استقرار قرارداد هوشمند:

قرارداد SmartHomeFactory بر روی بلاک چین مستقر شده است. این قرارداد کارخانه به عنوان یک نقطه مرکزی برای ایجاد و مدیریت قراردادهای SmartHome فردی است که هر یک نماینده یک خانواده شرکت کننده در شبکه هوشمند است

* ایجاد بازار ریزشبکه: قرارداد MicrogridMarket منعقد شده است که به عنوان یک بازار غیرمتمرکز عمل می کند که در آن خانوارها می توانند پیشنهادات و درخواست های انرژی را ارسال کنند. این قرارداد تطابق خریداران و فروشندگان انرژی را تسهیل می‌کند و فرآیند تجارت شفاف و کارآمد را تضمین می‌کند

1. ثبت نام:

* ثبت‌نام خانوار:‌هر خانوار شرکت‌کننده، از طریق مالک تعیین‌شده خود، با قرارداد SmartHomeFactory تعامل می‌کند تا یک قرارداد SmartHome ایجاد کند. این فرآیند خانوار را در سیستم ثبت می‌کند و ظرفیت انرژی اولیه و جزئیات مالکیت آن را مشخص می‌کند
* ادغام کنتور هوشمند: کنتور هوشمند در هر خانوار به قرارداد SmartHome مربوطه آن مرتبط است. کنتور هوشمند به‌طور مداوم داده‌های مصرف انرژی و تولید را کنترل می‌کند، که سپس برای به‌روزرسانی وضعیت قرارداد SmartHome استفاده می‌شود که منعکس‌کننده تقاضای انرژی فعلی خانوار، عرضه، و انرژی اضافی موجود برای تجارت است.

1. مرحله سفارش برق

* تجارت انرژی: خانوارهایی که انرژی مازاد دارند می توانند سفارشات فروش ("پرسش") را به قرارداد MicrogridMarket ارسال کنند و قیمت و مقدار انرژی را که مایل به فروش هستند مشخص کنند. برعکس، خانوارهایی که به انرژی اضافی نیاز دارند، می توانند سفارش خرید ("مناقصه") را با قیمت و مقدار مورد نظر خود ارسال کنند
* تطبیق و اجرای سفارش: قرارداد MicrogridMarket به طور خودکار پیشنهادات و درخواست ها را بر اساس قیمت و اولویت زمانی مطابقت می دهد. هنگامی که یک تطابق پیدا می شود، قرارداد انتقال انرژی بین خریدار و فروشنده را تسهیل می کند، وضعیت های قرارداد SmartHome مربوطه را به روز می کند و تراکنش را روی بلاک چین ثبت می کند
* پرداخت امن: این سیستم از ارز دیجیتال بومی بلاک چین برای پرداخت های ایمن و شفاف استفاده می کند. پس از انتقال موفقیت آمیز انرژی، قرارداد SmartHome خریدار به طور خودکار مبلغ توافق شده را به قرارداد SmartHome فروشنده منتقل می کند و تسویه حساب فوری و بدون دستکاری را تضمین می کند.

1. مرحله رتبه بندی اعتماد:

در حالی که در سیستم فعلی پیاده سازی نشده است، مکانیسم رتبه بندی اعتماد به عنوان یک پیشرفت آینده پیشنهاد شده است. این مکانیسم امتیازهای اعتماد را به خانوارهای شرکت کننده بر اساس رفتار گذشته آنها در بازار انرژی اختصاص می دهد، مانند:

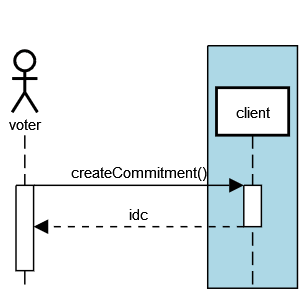
* تراکنش های موفق: تکمیل معاملات انرژی بدون مشکل به امتیاز اعتماد خانوار کمک می کند.
* تراکنش های ناموفق: لغو سفارش ها یا عدم ارائه انتقال انرژی مورد توافق بر امتیاز اعتماد تأثیر منفی می گذارد.
* پاسخگویی: به موقع بودن پاسخ ها به پیشنهادات و درخواست ها نیز می تواند در رتبه بندی اعتماد نقش داشته باشد. این سیستم رتبه بندی اعتماد می تواند چندین مزیت ارائه دهد
* افزایش امنیت امنیت: خانوارهایی که امتیازات اعتماد بالاتری دارند می‌توانند در فرآیند تطبیق سفارش اولویت‌بندی شوند و خطر تعامل با بازیگران غیرقابل اعتماد یا مخرب را کاهش دهند.
* مشوق های مبتنی بر شهرت: امتیازات اعتماد می تواند برای تشویق رفتار خوب و جلوگیری از فعالیت های متقلبانه در بازار انرژی استفاده شود.
* قیمت‌گذاری پویا: امتیازات اعتماد می‌تواند بر قیمت‌گذاری پیشنهادها و درخواست‌های انرژی تأثیر بگذارد، با نمرات اعتماد بالاتر به طور بالقوه بر قیمت‌های برتر یا دریافت رفتار ترجیحی. با گنجاندن این مکانیسم رتبه‌بندی اعتماد، سیستم پیشنهادی می‌تواند اعتماد و امنیت را در شبکه هوشمند افزایش دهد و محیط تجارت انرژی قابل اعتمادتر و کارآمدتری را برای همه شرکت‌کنندگان ایجاد کند.

1. **فاز راه اندازی:**

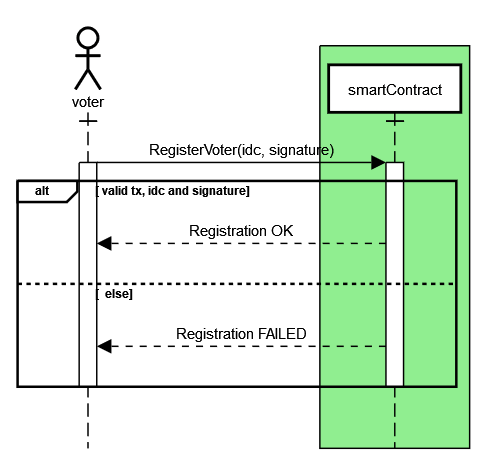
در فاز راه‌اندازی، ابتدا قرارداد هوشمند متعلق به شبکه اصلی هوشمند برق بر روی بلاکچین قرار می‌گیرد. این قرارداد هوشمند مسئول ایجاد راه ارتباطی برای مشترکان شبکه برق هوشمند به منظور تبادل انرژی است.

به علاوه، این مرحله به ازای هر مشترک شبکه هوشمند اجرا می‌شود. هر مشترک بایستی حساب هوشمند[[8]](#footnote-8) خود را ایجاد کند تا بتواند به تبادل انرژی بر بستر بلاکچین بپردازد.

1. **فاز ثبت نام:** در فاز ثبت نام، ابتدا هر رای‌دهنده با استفاده از کلاینت رای‌گیری بر روی دستگاه خود، تعهد هویت خود را تولید و مقادیر خصوصی خود را به صورت امن ذخیره می‌کند (شکل ‏4–1). سپس مرحله احراز هویت انجام می‌شود. در این پژوهش، به دلیل خارج بودن موضوع احراز هویت و قابلیت انجام آن به شکل‌های مختلف (فرض احراز هویت در بخش 1-3-2)، به سادگی فرض می‌کنیم که فرد واجد شرایط پس از انجام احراز هویت، امضای معتبری بر روی تعهد هویت خود دریافت می‌کند. سپس وی می‌تواند با داشتن این امضا به قرارداد هوشمند مراجعه کند و ثبت نام خود را انجام دهد (شکل ‏4–2).

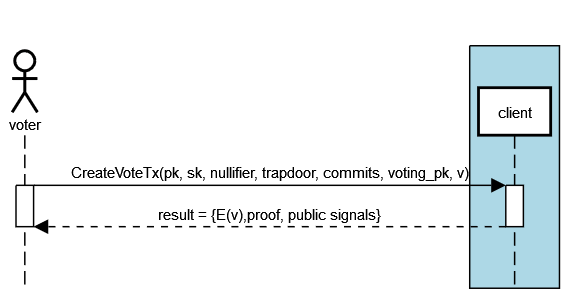


شکل ‏4–1 ایجاد تعهد هویت توسط رای‌دهنده



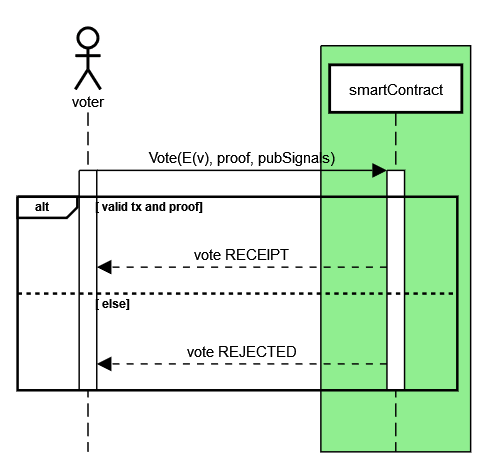
شکل ‏4–2 ارسال تعهد هویت توسط رای‌دهنده احراز هویت شده به قرارداد هوشمند

1. **فاز رای‌گیری:** رای‌گیری به دستور برگزارکننده (برگزارکنندگان) انتخابات شروع می‌شود. از این نقطه تا پایان زمان رای‌گیری، افراد ثبت نام کرده می‌توانند رای خود را ثبت کنند. برای این کار ابتدا هر فرد با استفاده از کلاینت خود، تراکنش مجاز رای‌گیری را ایجاد می‌کند (شکل ‏4–3).



شکل ‏4–3 ایجاد اثبات با دانایی پنهان مورد نیاز برای رای دهی

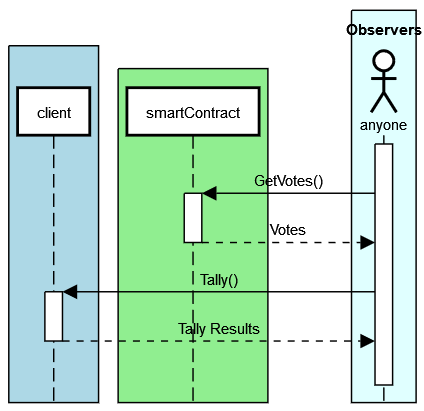
پس از انجام این مرحله، رای‌دهنده مقادیرِ اثبات، مقادیر عمومی و رای رمز شده را در قالب تراکنش ثبت رای به قرارداد هوشمند پروتکل تحویل می‌دهد (شکل ‏4–4).



شکل ‏4–4 ارسال اثبات با دانایی پنهان و رای رمز شده به قرارداد هوشمند

در نهایت، با فرا رسیدن پایان رای‌گیری، برگزارکنندگان انتخابات دستور پایان را به قرارداد هوشمند صادر می‌کند. از این نقطه به بعد، قرارداد هوشمند هیچ رای جدیدی را نخواهد پذیرفت.

1. **فاز شمارش:** در ابتدای فاز شمارش، برگزارکنندگان انتخابات باید با استفاده از DKG، کلید خصوصی را با رسیدن به حد نصاب تولید و سپس به صورت عمومی منتشر کنند. هر کسی با داشتن دسترسی به یک گره کامل بلاکچین اتریوم (چه مستقیم و چه غیر مستقیم از طریق کلاینت‌های سَبُک[[9]](#footnote-9)) می‌تواند تمام آرای ثبت شده را از قرارداد هوشمند دریافت کند، و سپس با استفاده از الگوریتم شمارشِ قرار گرفته در کلاینت، شمارش آرا را انجام دهد (شکل ‏4–5).



شکل ‏4–5 انجام شمارش آرا

### الگوریتم راه اندازی

الگوریتم راه اندازی در شکل ‏4–6 آمده است. ورودی این الگوریتم:

* **پارامتر امنیتی**[[10]](#footnote-10) است که تعیین می‌کند اندازه مساله سخت برای رسیدن به **سطح امنیتی**[[11]](#footnote-11) مورد نیاز چه قدر باشد. سطح امنیتی تعیین می‌کند که تعداد عملیات مورد نیاز برای اجرای بروت‌فورس بر روی یک الگوریتم یا سیستم رمزنگاری[[12]](#footnote-12)، چه قدر است. در پیوست ب، به طور مفصل بحث شده است که برای رسیدن به سطح امنیتی 128بیتی، در حال حاضر بر روی اتریوم بایستی از منحنی Baby Jubjub استفاده کنیم که پارامتر امنیتی آن 256 است.

مراحل این فاز عبارتند از:

1. **انتخاب گروه برای عملیات رمزنگاری:** الگوریتمی از زمان چند جمله‌ای که در خروجی آن، و دو عدد اول بسیار بزرگ به گونه‌ای که و گروهی از مرتبه‌ی ، و و سازنده‌های تصادفی آن و ، و در نهایت، زیرگروهِ و حل مساله‌ی لگاریتم گسسته در سخت باشد. در پیوست ب، عنوان شده که بهتر است از منحنی‌های بیضوی استاندارد به جای تولید تصادفی زیرگروهِ استفاده کرد. با این حال برای سازگاری با ادبیات موضوع و ماژولار در نظر گرفتن پروتکل، در بخش نظری به گروه یا منحنی بیضوی خاصی اشاره نشده است.
2. **انتخاب الگوریتم هش:** بر اساس نیاز پروتکل و ابزارهای در دسترس، بایستی الگوریتم هش انتخاب شود. در پیوست ج، بحث شده است که مصالحه بین انتخاب مناسب‌ترین الگوریتم هش چه فاکتورهایی در بر دارد.
3. **قرار دادن قرارداد هوشمند بر بلاکچین**: در بخش 1-3-3 بحث شد که فرض می‌کنیم پروتکل DKG مانند [4] وجود دارد که تولید کلید خصوصی پروتکل و کلید عمومی برگزارکنندگان را انجام و قرارداد هوشمند رای‌گیری را بر روی شبکه اصلیِ اتریوم قرار می‌دهد. همراه با بارگذاری قرارداد هوشمند، تابع سازنده[[13]](#footnote-13) آن اجرا می‌شود و مقادیر زیر را به عنوان ورودی می‌گیرد:

* عمق درخت مرکل (تعیین کننده حداکثر تعداد آرا)،
* کلید عمومی برگزارکنندگان انتخابات،
* حد نصاب لازم برای برگزارکنندگان به منظور اجرای فرمان‌های مدیریتی.

این سازنده مقداردهی متغیرها، آرایه‌ها و نگاشت‌های قرارداد هوشمند را انجام می‌دهد که توضیح هر کدام در شکل ‏4–6 آمده است.

### الگوریتم‌های کلاینت

الگوریتم‌های کلاینت می‌تواند توسط برگزارکنندگان و رای‌دهندگان مورد استفاده قرار بگیرد. از دیدگاه نرم‌افزاری، الگوریتم‌های کلاینت یک ماژول است که کلاینت‌های تولیدشده برای فرآیند رای‌گیری، از آن استفاده می‌کنند.

|  |
| --- |
| Setup   * Inputs: * Outputs:    2. Initialize Smart Contract parameters : * Empty solidity array, , pubkey of all trustees * Empty solidity array, , all id commitments stored in merkle tree as leafs * Empty solidity mapping, , mapping voter pubkey to its encrypted votes * Empty solidity mapping, , a map showing if a nullifier has been seen * Empty Solidity mapping, , a map showing if a merkle root has been seen * Empty solidity mapping, , a map showing if a trustee has complained * Empty solidity array, * Empty solidity array, * , a Boolean used for termination of voting process * , m is the minimum of n trustee needed to act as admin |

شکل ‏4–6 الگوریتم راه اندازی

1. : این تابع با استفاده از پارامترهای عمومی پروتکل، یک جفت کلید، و دو مقدارnullifier و trapdoor را به صورت تصادفی انتخاب می‌کند. هشِ این سه مقدار، تعهد هویت رای‌دهنده را می‌سازد.

|  |
| --- |
| * Inputs: * Output: |

شکل ‏4–7 الگوریتم CreateCommitment

1. . این تابع با اثبات داناییِ پنهان، اثبات می‌کند که مقادیر nullifier و trapdoor و کلید خصوصی را می‌داند، به طوری که هش مقادیر nullifier، trapdoor و کلید عمومی، ایجاد کننده تعهدی است که در انباشتگر قرار گرفته است. به علاوه، این تابع ثابت می‌کند که هش رای رمز شده توسط کلید خصوصی رای‌دهنده، امضای دیجیتال شده است.

تابع ایجاد تراکنش رای:

* + یک جفت کلید موقت[[14]](#footnote-14) تولید می‌کند؛
  + کلید مشترک بین رای‌دهنده و پروتکل با استفاده از تولید می‌شود؛
  + رای با این کلید مشترک رمزنگاری می‌شود؛
  + مقدار هشِ رای رمز شده را با کلید خصوصی رای‌دهنده امضا می‌کند؛
  + با استفاده از تابع و به صورت محلی، درخت مرکل (پیوست ث) قرار گرفته در قرارداد هوشمند را شبیه سازی و اثبات مرکل شامل مسیر مرکل و عناصر آن را تولید می‌کند؛ و
  + با استفاده از ، ‌اثبات با داناییِ پنهان و مقادیر عمومی لازم برای اثبات حق رای دهی را تولید می‌کند.

|  |
| --- |
| * Inputs:   , list of all idc   * Outputs: |

شکل ‏4–8 الگوریتم CreateVoteTx

1. : این تابع بر اساس گروه رمزنگاری انتخاب شده در راه اندازی سیستم، مقدار را با کلید رمزگذاری می‌کند.
2. : این تابع بر اساس گروه رمزنگاری انتخاب شده در راه اندازی سیستم، پیام رمز شده را با کلید رمزگشایی می‌کند.
3. : این تابع، ثابت می‌کند که مقدار در مجموعه قرار دارد. مقادیر مجموعه در انباشتگر قرار گرفته اند و خروجی این تابع برای اثبات عضویت در انباشتگر به شکل داناییِ پنهان به کار می‌رود. اطلاعات بیشتر در رابطه با این تابع در پیوست ث آمده است.
4. : این تابع، پیام را با استفاده از کلید خصوصی امضای دیجیتال می‌کند. پروتکل امضای EdDSA در پیوست ت بررسی شده است.
5. : این تابع، پس از دریافت مقدار هویتی رای‌دهنده شامل کلید عمومی، nullifier، trapdoor، شاهدِ عضویت تعهد هویت در مجموعه انباشت شده، هشِ رای رمز شده و امضای دیجیتال رای‌دهنده بر روی هش رمز شده، به شکل داناییِ پنهان اثبات می‌کند که:
   * تعهد هویت رای‌دهنده در مجموعه تمام تعهدهای ثبت شده در قرارداد هوشمند وجود دارد؛
   * امضای رای‌دهنده بر هش رای رمز شده معتبر است

خروجی این تابع، شامل مقادیر عمومیِ:

* + مقدار انباشتگر (ریشه درخت در صورت استفاده از درخت مرکل)؛
  + مقدار هش شده nullifier (مکانیسم امنیتی برای جلوگیری از دوبار رای دادن)؛
  + و مقدارِ ‌اثبات با داناییِ پنهان حاصل است.

1. : شمارش را به صورت خودمختار و خارج از بلاکچین[[15]](#footnote-15) انجام می‌دهد. پس از پایان انتخابات، برگزارکنندگان کلید خصوصی رای‌گیری را آشکار می‌کنند. این تابع با استفاده از این کلید خصوصی می‌تواند تمامی آرا را رمزگشایی کند. زیرتابع ( که به ازای هر رای‌گیری به صورت مجزا تعریف می‌شود) بر اساس هر رای رمزگشایی شده، تصمیم می‌گیرد که آن رای به کدام کاندیدا (یا آرای باطله) تعلق دارد.

|  |
| --- |
| * Inputs: protocol secret key, list of all votes * Output: |

شکل ‏4–9 الگوریتم Tally

1. : این تابع بر اساس پروتکل تبادل کلید دیفی هلمن، کلید مشترک را محاسبه می‌کند. این الگوریتم در بخش ‏4-1-1 مورد بحث قرار گرفته است.

### الگوریتم‌های قرارداد هوشمند

لازم به ذکر است که بر اساس «فرض وجود مکانیسم بررسی امضای تراکنش‌های بلاکچین»، نیازی به بررسی جداگانه امضای دیجیتال تراکنش‌های فراخواننده توابع زیر نیست. توابع عمومی به کار رفته در این قرارداد هوشمند، به ترتیب قرارگیری در فرآیند رای‌گیری عبارتند از:

1. : ورودی این تابع شامل کلید عمومی ثبت نام کننده، تعهد هویتی رای‌دهنده یا ، و اثبات احراز هویت شدن وی است. برای سادگی، فرض کردیم که رای‌دهنده پس از احراز هویت شدن، یک امضا، ، از ثبت نام کننده‌ای با کلید عمومیِ ، دریافت کرده و قرارداد هوشمند قابلیت تشخیص صحت این امضا را دارد. این تابع با قراردادن مقدار تهعد در انباشتگر، رای‌دهنده را ثبت نام می‌کند.

|  |
| --- |
| * Inputs: , , |

شکل ‏4–10 الگوریتم RegisterVoter

1. : ورودی این تابع شامل کلید عمومی برگزارکننده و داده‌های اختیاری، همانند دلیل ثبت شکایت است. این تابع ابتدا بررسی می‌کند که آیا کلید عمومی فرستنده، به عنوان برگزارکننده انتخابات ثبت شده است. می‌توان در پیاده‌سازی مبتنی بر اتریوم، از کلید عمومی فرستنده تراکنش، استفاده و یا مستقیما کلید را به عنوان پارامتر ورودی تابع دریافت کرد. در صورت صحت تراکنش، شکایت ثبت می‌شود. هنگامی که تعداد شکایات ثبت شده به حد نصاب برسد، فرآیند رای‌گیری متوقف خواهد شد. با این حال تنها تا قبل از شروع فرآیند ثبت آرا، باید برگزارکنندگان شکایات خود را وارد کنند.

|  |
| --- |
| * Inputs: , arbitrary (e.g. reason of complaint) |

شکل ‏4–11الگوریتم Complain

1. : این تابع تنها توسط برگزارکنندگان انتخابات قابل اجراست. هر برگزارکننده برای رای دادن به شروع انتخابات، این تابع را فراخوانی می‌کند. در صورت رسیدن آرا به حد نصاب، رای‌گیری شروع می‌شود. تا قبل از شروع رای‌گیری، قرارداد هوشمند هیچ گونه تراکنش رای را پردازش نمی‌کند.

|  |
| --- |
| * Inputs: |

شکل ‏4–12 الگوریتم StartVoting

1. : ورودی‌های این تابع شامل ‌‌اثبات با داناییِ پنهان، ، مقادیر عمومی اثبات دانایی، ، و رای رمزشده، است. تابع ابتدا در مقادیر عمومی ارسالی بررسی می‌کند که:
   * + - ریشه مرکل قبلا ثبت شده باشد؛ و
       - هشِ nullifier قبلا ثبت نشده باشد.

سپس، ‌‌اثبات با داناییِ پنهان ارایه شده را بررسی می‌کند. در صورت درست بودن ورودی‌ها، رای رمز شده در حافظه قرارداد هوشمند قرار می‌گیرد.

|  |
| --- |
| * Inputs: |

شکل ‏4–13الگوریتم Vote

1. : این تابع تنها توسط برگزارکنندگان انتخابات قابل اجراست. هر برگزارکننده برای رای دادن به پایان انتخابات، این تابع را فراخوانی می‌کند. در صورت رسیدن آرا به حد نصاب، رای‌گیری پایان می‌یابد. بعد از اتمام رای‌گیری، قرارداد هوشمند هیچ گونه تراکنش رای را پردازش نمی‌کند.

|  |
| --- |
| * Inputs: |

شکل ‏4–14الگوریتم EndVoting

1. : این تابع صحت امضای دیجیتال، ، توسط کلید عمومی بر روی پیام را بررسی می‌کند. در این پژوهش از امضای EdDSA استفاده شده است که در پیوست ت به صورت مفصل مورد بررسی قرار گرفته است. خروجی این تابع در صورت صحیح بودن عدد یک یا مقدار «صحیح» و در غیر این صورت، صفر یا «غلط» خواهد بود.
2. : این تابع به عنوان ورودی، مقدار تعهد هویتِ رای‌دهنده را می‌گیرد و آن را به انباشتگر اضافه می‌کند. در صورت استفاده از درخت مرکل افزایشی به عنوان انباشتگر، پیوست ث، ریشه جدیدِ درخت مرکل باید در آرایه از قرارداد هوشمند ثبت شود.

|  |
| --- |
| * Inputs: identity commitment  1. Add to merkle tree |

شکل ‏4–15الگوریتم InsertCm

1. : ورودی این تابع شامل ‌اثبات با داناییِ پنهان ارائه شده توسط رای‌دهنده، مقادیر عمومی تولید شده در فرآیند ‌اثبات با داناییِ پنهان و رای به صورت رمز شده است. در صورت استفاده از درخت مرکل به عنوان انباشتگر، پیوست ث، مقادیر عمومی تولید شده شامل ریشه درخت نیز خواهد بود. قرارداد هوشمند باید تمامی ریشه‌های تولید شده در حین ثبت نام رای‌دهندگان را ثبت کرده باشد و ابتدا بررسی کند که آیا ریشه ارایه شده در مقادیر عمومی قبلا در حافظه قرارداد هوشمند ثبت شده است یا خیر سپس به بررسی اثبات عضویت در انباشتگر بپردازد که عملی پرهزینه بر روی بلاکچین است. به علاوه معتبر بودن اثبات تنها نشان می‌دهد که رای‌دهنده یک تعهد هویتیِ را در درخت مرکلی می‌داند که ریشه آن در مقادیر عمومی قرار گرفته است. اما اگر این ریشه برای قرارداد هوشمند آشنا نباشد، به این معناست که درخت مرکل در سمت اثبات‌کننده، اشتباه محاسبه شده یا اینکه رای‌دهنده قصد داشته یک اثبات به ظاهر سالم ولی بدون داناییِ تعهد هویت در درخت مرکلِ قرارداد هوشمندِ رای‌گیری ارایه دهد. در بخش ‏4-4، پیاده‌سازی پروتکل، به طور ویژه درباره این تابع صحبت شده است.
2. : این تابع بررسی می‌کند که آرایه ارسالی به آن، به حد نصاب برگزارکنندگان رسیده است یا خیر. از این تابع می‌توان برای بررسی شروع و پایان رای‌گیری استفاده کرد.

|  |
| --- |
| * Inputs: any type of  1. (m is the quorum required) |

شکل ‏4–16الگوریتم Quorum

1. تابع کمکیِ : یک مکانسیم امنیتی که در صورت برآورده نشدن شرط مورد نیاز آن، اجرای ادامه تابع را متوقف می‌کند. این مکانیسم در EVM به صورت قطعی پیاده‌سازی شده است.

## پیاده‌سازی نرم افزاری پروتکل

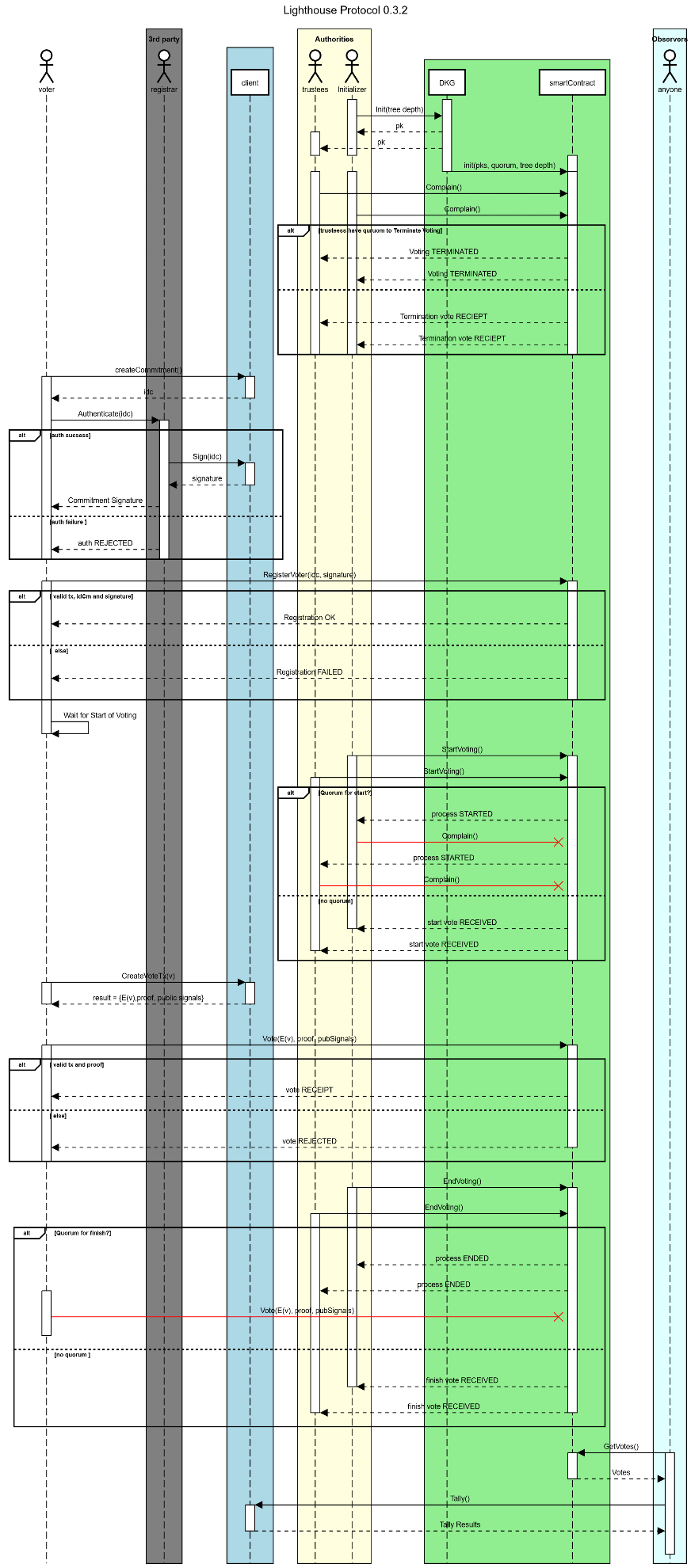
در پیاده‌سازی این پروتکل از چند ابزار و کتابخانه متن باز استفاده شده است. در ابتدای این بخش، توضیحی راجع به ابزارها ارایه و سپس پیاده‌سازی پروتکل توسط این ابزارها و کتابخانه‌ها بررسی می‌شود.

### مدارهای محاسبات عددی پروتکل

مدار اصلی پروتکل ‌اثبات با داناییِ پنهان به کار رفته در این پژوهش از بخش‌های متعددی تشکیل شده است:

#### محاسبه تعهدِ هویت

برای رای دادن، رای‌دهنده باید ثابت کند که تعهد هویت وی در درخت مرکل قرارداد هوشمند قرار گرفته است. بنابراین، یکی از زیرمراحل اثبات این قضیه به شکل دانایی صفر، محاسبه تعهد هویت توسط خود رای‌دهنده است. مدار شکل ‏4–12، کلید عمومی،id\_nullifier و id\_trapdoor را به عنوان ورودی دریافت می‌کند و به عنوان خروجی، تعهد هویت را تحویل می‌دهد. این مدار به عنوان یک قالب کمکی (پیوست چ) در مدار اصلی مورد استفاده قرار می‌گیرد. دلیل اینکه مدار اصلی تعهد هویت را مستقیما از رای‌دهنده نمی‌پذیرد و وی را وادار به تولید مجدد آن می‌کند این است که رای‌دهنده باید ثابت کند که یک کلید عمومی، id\_nullifier و id\_trapdoor‌ را می‌داند که هش حاصل از این سه مقدار برابر با یکی از تعهدهای قرار گرفته



شکل ‏4–17 نمودار توالیِ پروتکل

در درخت مرکلِ قرارداد هوشمند رای‌گیری است. برای محاسبه هش در این مدار از هش پوسایدون استفاده شده است.

1. include "./poseidon/poseidonHasher.circom";
3. template CalculateIdentityCommitment() {
4. signal input identity\_public\_key\_subgroup\_element;
5. signal input identity\_nullifier;
6. signal input identity\_trapdoor;
8. signal output out;
10. component hasher = PoseidonHashT4();
11. hasher.inputs[0] <== identity\_public\_key\_subgroup\_element;
12. hasher.inputs[1] <== identity\_nullifier;
13. hasher.inputs[2] <== identity\_trapdoor;
14. out <== hasher.out;
15. }

شکل ‏4–18 مدار محاسبه تعهد هویت

#### مدار بررسی عضویت در درخت مرکل

برای پیاده‌سازی پروتکل از درخت مرکلِ افزایشی[[16]](#footnote-16) استفاده شده است. مزیت استفاده از این ساختمان داده، بهینه بودنِ اضافه کردن برگ‌های جدید به درخت است؛ با این حال از به روز رسانیِ برگ‌ها پشتیبانی نمی‌کند. مدار شکل ‏4–11، با دریافت یک برگ، مسیر مرکل آن برگ و عناصر قرار گرفته در مسیر مرکل، ریشه و اثبات عضویت برگ در درخت مرکل را تولید می‌کند. تابع HashLeftRight به کار رفته در این مدار، از هش پوسایدون برای محاسبه هش دو گره همسایه در درخت مرکل استفاده می‌کند.

#### محاسبه Nullifier

Nullifier یا باطل کننده، یک مکانیسم امنیتی است که از پژوهش ساندر و همکاران [7] الهام گرفته شده است. این مکانیسم در پژوهش ساندر و همکاران به نام «شماره سریال» شناخته می‌شود و نام Nullifier، نامی است که پژوهش شکل دهنده بلاکچین زی‌کش بر آن گذاشته است. فلسفه این مقدار این است که اجازه ندهد رای‌دهنده از یک اثبات معتبر دو یا بیشتر از آن استفاده کند، به همین دلیل طوری طراحی شده که مقدار آن هیچ اطلاعاتی راجع به تعهد هویت انتخابی رای‌دهنده نشت ندهد و در عین حال، وابسته به id\_nullifier و مکان قرارگیری تعهد هویت در درخت مرکل باشد. این امر باعث می‌شود که به ازای هر تعهد هویت معتبر، تنها یک باطل کننده تولید شود. قرارداد هوشمند رای‌گیری می‌تواند در صورت معتبر بودن اثبات ارایه شده، این باطل کننده را ثبت کند تا دیگر کسی با آن اجازه رای دادن نداشته باشد.

1. include "../node\_modules/circomlib/circuits/mux1.circom";
2. include "./poseidon/poseidonHasher.circom";
4. template MerkleTreeInclusionProof(n\_levels) {
5. signal input leaf;
6. signal input path\_index[n\_levels];
7. signal input path\_elements[n\_levels][1];
8. signal output root;
10. component hashers[n\_levels];
11. component mux[n\_levels];
13. signal levelHashes[n\_levels + 1];
14. levelHashes[0] <== leaf;
15. for (var i = 0; i < n\_levels; i++) {
16. // Should be 0 or 1
17. path\_index[i] \* (1 - path\_index[i]) === 0;
19. hashers[i] = HashLeftRight();
20. mux[i] = MultiMux1(2);
22. mux[i].c[0][0] <== levelHashes[i];
23. mux[i].c[0][1] <== path\_elements[i][0];
25. mux[i].c[1][0] <== path\_elements[i][0];
26. mux[i].c[1][1] <== levelHashes[i];
28. mux[i].s <== path\_index[i];
29. hashers[i].left <== mux[i].out[0];
30. hashers[i].right <== mux[i].out[1];
32. levelHashes[i + 1] <== hashers[i].hash;
33. }
35. root <== levelHashes[n\_levels];
36. }

شکل ‏4–19 مدار بررسی عضویت در درخت مرکل

1. include "./poseidon/poseidonHasher.circom";
3. template CalculateNullifierHash() {
4. signal input external\_nullifier;
5. signal input identity\_nullifier;
6. signal input path\_index\_num;
8. signal output out;
10. component hasher = PoseidonHashT4();
11. hasher.inputs[0] <== external\_nullifier;
12. hasher.inputs[1] <== identity\_nullifier;
13. hasher.inputs[2] <== path\_index\_num;
14. out <== hasher.out;
15. }

شکل ‏4–20 مدار محاسبه Nullifier

#### مدار اصلی

مدار اصلیِ پروتکل، شکل ‏4–13 مداری است که رای‌دهنده از آن برای تولید ‌اثبات با داناییِ پنهان و ارایه به قرارداد هوشمند رای‌گیری استفاده می‌کند.

در ابتدا، مدار اصلی وظیفه دارد تا با قالب ارایه شده در بخش 4-4-1-1، تعهد هویت رای‌دهنده را بر اساس مقادیر ورودی به مدار محاسبه کند. سپس با استفاده از قالبِ بخش 4-4-1-2، اثبات عضویت این تعهد هویت در درخت مرکل را بر اساس مسیر مرکل و عناصر مسیر مرکل داده شده به عنوان ورودی حساب کند. رای‌دهنده باید مسیر مرکل و عناصر مسیر مرکل را با استفاده از الگوریتم‌های کلاینت از پیش محاسبه کند و به عنوان ورودی در اختیار مدار بگذارد. سپس از قالب بخشِ 4-4-1-3 برای محاسبه Nullifier استفاده می‌شود. در نهایت، امضای رای‌دهنده بر روی هشِ رای رمز شده بررسی می‌شود. در صورت اتمام موفقیت آمیز مدار، ‌اثبات با داناییِ پنهان و مقادیر عمومی را به عنوان خروجی تحویل می‌دهد.

1. // n\_levels must be < 32
2. template Lighthouse(n\_levels) {
3. // BEGIN signals
5. signal input signal\_hash;
6. signal input external\_nullifier;
8. signal private input fake\_zero;
10. // poseidon vector commitment
11. signal private input identity\_pk[2];
12. signal private input identity\_nullifier;
13. signal private input identity\_trapdoor;
14. signal private input identity\_path\_elements[n\_levels];
15. signal private input identity\_path\_index[n\_levels];
17. // signature on (external nullifier, signal\_hash) with identity\_pk
18. signal private input auth\_sig\_r[2];
19. signal private input auth\_sig\_s;
21. // poseidon hash
22. signal output root;
23. signal output nullifiers\_hash;
24. // END signals
26. // BEGIN constants
28. var IDENTITY\_PK\_SIZE\_IN\_BITS = 254;
29. var NULLIFIER\_TRAPDOOR\_SIZE\_IN\_BITS = 254;
30. var EXTERNAL\_NULLIFIER\_SIZE\_IN\_BITS = 254;
32. // END constants
34. fake\_zero === 0;
36. component verify\_identity\_pk\_on\_curve = BabyCheck();
37. verify\_identity\_pk\_on\_curve.x <== identity\_pk[0];
38. verify\_identity\_pk\_on\_curve.y <== identity\_pk[1];
40. component verify\_auth\_sig\_r\_on\_curve = BabyCheck();
41. verify\_auth\_sig\_r\_on\_curve.x <== auth\_sig\_r[0];
42. verify\_auth\_sig\_r\_on\_curve.y <== auth\_sig\_r[1];
44. // BEGIN identity commitment
46. component verifyPkOnCurve = VerifyPkOnCurve();
47. verifyPkOnCurve.identity\_pk[0] <== identity\_pk[0];
48. verifyPkOnCurve.identity\_pk[1] <== identity\_pk[1];
50. component pk2SubgroupElement = Pk2SubgroupElement();
51. pk2SubgroupElement.identity\_pk[0] <== identity\_pk[0];
52. pk2SubgroupElement.identity\_pk[1] <== identity\_pk[1];
54. component identity\_commitment = CalculateIdentityCommitment();
55. identity\_commitment.identity\_public\_key\_subgroup\_element <== pk2SubgroupElement.out;
56. identity\_commitment.identity\_nullifier <== identity\_nullifier;
57. identity\_commitment.identity\_trapdoor <== identity\_trapdoor;
59. // END identity commitment
61. // BEGIN tree
62. component tree = MerkleTreeInclusionProof(n\_levels);
63. tree.leaf <== identity\_commitment.out;
64. for (var i = 0; i < n\_levels; i++) {
65. tree.path\_index[i] <== identity\_path\_index[i];
66. tree.path\_elements[i][0] <== identity\_path\_elements[i];
67. }
68. root <== tree.root;
69. // END tree
71. // BEGIN nullifiers
72. component bit2num = Bits2Num(n\_levels);
73. for (var i = 0; i < n\_levels; i ++) {
74. bit2num.in[i] <== identity\_path\_index[i];
75. }
76. component calculateNullifierHash = CalculateNullifierHash();
77. calculateNullifierHash.external\_nullifier <== external\_nullifier;
78. calculateNullifierHash.identity\_nullifier <== identity\_nullifier;
79. calculateNullifierHash.path\_index\_num <== bit2num.out;
81. nullifiers\_hash <== calculateNullifierHash.out;
82. // END nullifiers
84. // BEGIN verify sig
85. component msg\_hasher = HashLeftRight();
86. msg\_hasher.left <== external\_nullifier;
87. msg\_hasher.right <== signal\_hash;
89. component sig\_verifier = EdDSAPoseidonVerifier();
90. (1 - fake\_zero) ==> sig\_verifier.enabled;
91. identity\_pk[0] ==> sig\_verifier.Ax;
92. identity\_pk[1] ==> sig\_verifier.Ay;
93. auth\_sig\_r[0] ==> sig\_verifier.R8x;
94. auth\_sig\_r[1] ==> sig\_verifier.R8y;
95. auth\_sig\_s ==> sig\_verifier.S;
96. msg\_hasher.hash ==> sig\_verifier.M;
98. // END verify sig
99. }

شکل ‏4–21 مدار اصلی پروتکل

### قراردادِ هوشمند پروتکل

دو تابع حایز اهمیت در قرارداد هوشمندِ پروتکل، توابع و هستند. در قرارداد هوشمند نوشته شده به زبان سالیدیتی، به شکل ‏4–14تعریف شده است. مطابق با مطالبی که در رابطه با فرض احراز هویت گفته شد، در اینجا تنها در نظر گرفته شده که این تابع توسط دسترسی برگزارکننده (برگزارکنندگان) انتخابات قابل اجرا باشد. اگر فراخواننده تابع یکی از برگزارکنندگان باشد، این تابع تعهد هویت را به عنوان برگی از درخت مرکل افزایشی اضافه می‌کند. تابع کمکی ، عمل اضافه کردن برگ به درخت مرکل و ذخیره سازی ریشه جدید به دست آمده پس از اضافه شدن برگ جدید را انجام می‌دهد و برای اختصار، در متن ذکر نشده و در کُد منبع برنامه قابل دسترسی است (بخش4-4-3).

تابع در شکل ‏4–15 آمده است؛ به دلیل اختصار، توابع کمکی در اینجا آورده نشده ولی در مخزن کُد منبع پروژه (بخش4-4-3) قابل دسترس هستند.

1. function registerVoter(uint256 idCommit) public onlyOwner returns (uint256) {
2. // insertIdentity makes sure the idCommit is not registered before
3. return insertIdentity(idCommit);
4. }

شکل ‏4–22 تابع ثبت نام رای‌دهنده در قرارداد هوشمند

1. function vote(
2. Cyphertext memory \_encryptedVote,
3. uint256[8] memory \_proof,
4. uint256 \_root,
5. uint256 \_nullifiersHash,
6. uint232 \_externalNullifier) public {
7. // Check whether each element in \_proof is a valid field element
8. require(
9. areAllValidFieldElements(\_proof),
10. "Lighthouse: invalid field element(s) in proof"
11. );
12. // Check whether the nullifier hash has been seen
13. require(
14. nullifierHashHistory[\_nullifiersHash] == false,
15. "Lighthouse: nullifier already seen"
16. );
17. // Check whether the given Merkle root has been seen previously
18. require(rootHistory[\_root], "Lighthouse: root not seen");
19. uint256 voteHash = hashSignal(\_encryptedVote);
20. uint256[4] memory publicSignals =
21. [\_root, \_nullifiersHash, voteHash, \_externalNullifier];
23. (uint256[2] memory a, uint256[2][2] memory b, uint256[2] memory c) =
24. unpackProof(\_proof);
25. require(
26. verifyProof(a, b, c, publicSignals),
27. "Lighthouse: invalid proof"
28. );
29. // Store the nullifiers hash to prevent double-voting
30. nullifierHashHistory[\_nullifiersHash] = true;
32. // record the encrpyted vote
33. beacons.push(\_encryptedVote);
34. }

شکل ‏4–23 تابع ثبت رای در قرارداد هوشمند

### مخزنِ کُد پروتکل

کُد منبع این پژوهش به آدرس [https://github.com/creepteks/Lighthouse](https://github.com/creepteks/lighthouse) و به صورت متن‌باز در اختیار پژوهشگران و علاقمندان قرار داده شده است. کُد مرجعِ پژوهش فعلی با برچسب v0.3.2 در این آدرس در دسترس است و انشعابِ اصلی توسعه در این مخزن، با نام develop، یا سایر انشعاب‌ها و برچسب‌ها، ممکن است به مرور زمان دچار تغییرات و به روز رسانی شوند و با آن چه در این متن آمده، تفاوت داشته باشند.

### اجرای سناریوی آزمایشی

با ورود به صفحه اینترنتی کُد منبع پروژه، می‌توان نحوه شیوه اجرای یک سناریوی شبیه سازی شده رای‌گیری را مشاهده کرد. برای اجرای این سناریو لازم است تا مراحل نصب ابزارهای لازم انجام شود. این مراحل به طور کامل در فایل README پروژه بر روی مخزن کُد آورده شده است. در اینجا نیز به طور خلاصه به بررسی این مراحل پرداخته شده است:

1. برای آماده‌سازی پروژه، باید دستور زیر در ریشه پروژه اجرا شود:

$ lerna bootstrap

این دستور تمامی کتابخانه‌های مورد نیاز پیاده‌سازی را نصب و آماده می‌کند.

1. سپس بایستی مدارهای ‌اثبات با داناییِ پنهان ایجاد شود؛ به این منظور، در پوشه circuits، یک اسکریپت با نام fast\_build\_circuits.sh قرار گرفته است تا به صورت خودکار عملیات کامپایل و مرحله trusted setup را انجام دهد، کلیدهای لازم برای اثبات و اعتبار سنجیِ یک ‌اثبات با داناییِ پنهان را ایجاد کند و در نهایت قرارداد هوشمندِ استاندارد و ارایه شده توسط اتریوم برای اعتبارسنجی zkSNARKs بر روی بلاکچین را تولید ‌کند.
2. در مرحله بعد، لازم است که از یک بلاکچین تستی استفاده کنیم؛ در اینجا برای سادگی، از یک بلاکچین آزمایشی اتریوم به نام ganache استفاده می‌شود. در پوشه ethcode، کافی است اسکریپت start\_ganache.sh اجرا شود تا به صورت خودکار، یک بلاکچین تستی با کلیدهای خصوصی مناسبِ این سناریو اجرا شود.
3. در مرحله بعد لازم است که تمام قراردادهای هوشمند کامپایل شوند. اسکریپت compile\_sol.sh در پوشه ethcode، به طور خودکار تمام قراردادهای هوشمند مورد نیاز برای پروتکل را کامپایل می‌کند.
4. مرحله بعد، اجرای سناریوی رای‌گیری است. در این مرحله، تمام فازهای پروتکل اجرا و نتایج به نمایش در می‌آیند. کافی است در پوشه ethcode، دستور زیر وارد شود:

$ node ./scripts/votingScenario.js

این دستور، سناریو را به اجرا در می‌آورد. سناریوی نوشته شده، ابتدا فاز راه اندازی را انجام می‌دهد و تمام قراردادهای مورد نیاز پروتکل را بر روی بلاکچین آزمایشی، قرار می‌دهد[[17]](#footnote-17). در مرحله بعد، یک برگزارکننده، یک ثبت‌نام کننده و 40 فرد واجد شرایط را به صورت فرضی در نظر می‌گیرد. مطابق با فازهای پروتکل، ابتدا مرحله ثبت نام انجام می‌شود، و سپس افراد واجد شرایط، رای خود را ثبت می‌کنند. در نهایت مرحله شمارش انجام می‌شود و حاصل شمارش آرا چاپ می‌شود.

1. Authentication [↑](#footnote-ref-1)
2. Authorization [↑](#footnote-ref-2)
3. Access Control [↑](#footnote-ref-3)
4. Collusion [↑](#footnote-ref-4)
5. Scalability [↑](#footnote-ref-5)
6. Auditability [↑](#footnote-ref-6)
7. Sequence Diagram [↑](#footnote-ref-7)
8. Smart Account [↑](#footnote-ref-8)
9. Light clients [↑](#footnote-ref-9)
10. Security parameter [↑](#footnote-ref-10)
11. Security level [↑](#footnote-ref-11)
12. Cryptosystem [↑](#footnote-ref-12)
13. Constructor [↑](#footnote-ref-13)
14. Ephemeral key [↑](#footnote-ref-14)
15. Off-chain [↑](#footnote-ref-15)
16. Incremental Merkle Tree [↑](#footnote-ref-16)
17. Deploy [↑](#footnote-ref-17)