Децентрализованный протокол для “Root-Of-Trust” чипов: гомоморфные и доверенно-прозрачные вычисления

Глеб Русяев

Летово

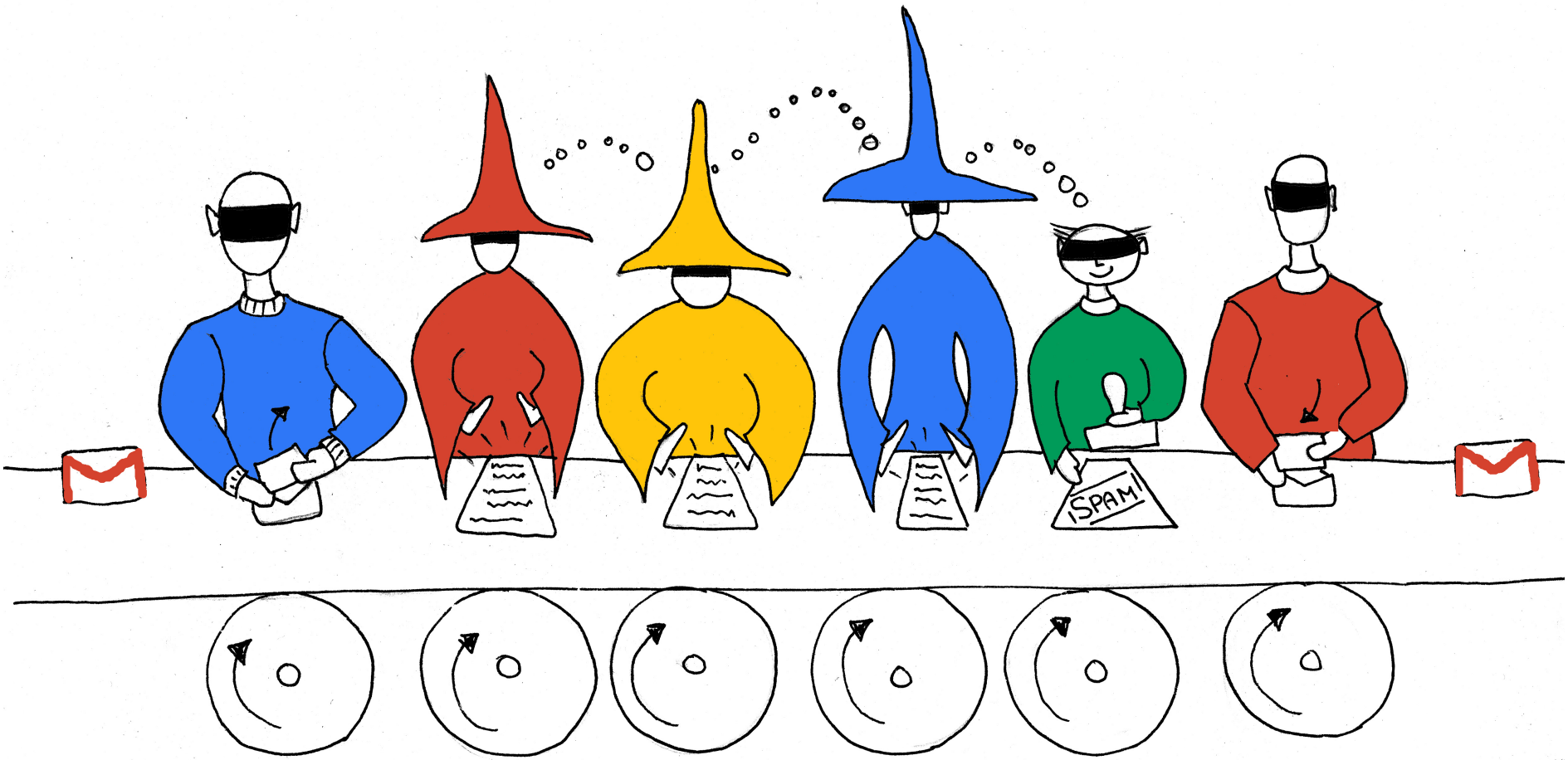
January 01, 1337

Обзор

- Описание проблемы и прошлая работа в этом направлении
- Cooler than “Root-Of-Trust”: PHCM
- Основные функции протокола и Blockchain/Torrent сеть
- Оракулы на службе у open-source и инфобеза
- Приминения и перспективы

Доверенные вычисления

- Гоморфное шифрование
- Anti-Tampering: Root-Of-Trust DRMs



Гомоморфное шифрование

- Craig Gentry: “Fully homomorphic encryption using ideal lattices”, 2009
- Что мешает применять это прямо сейчас? Проблемы с эффективностью

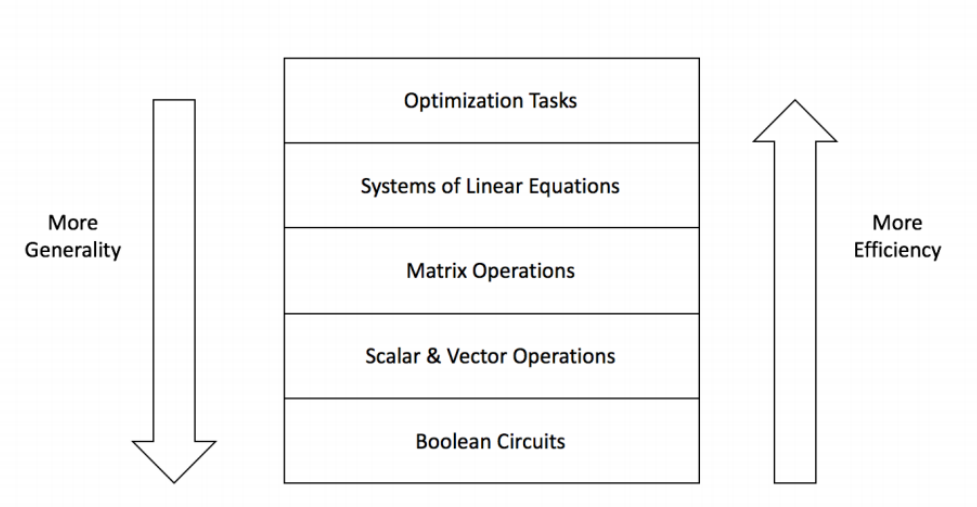


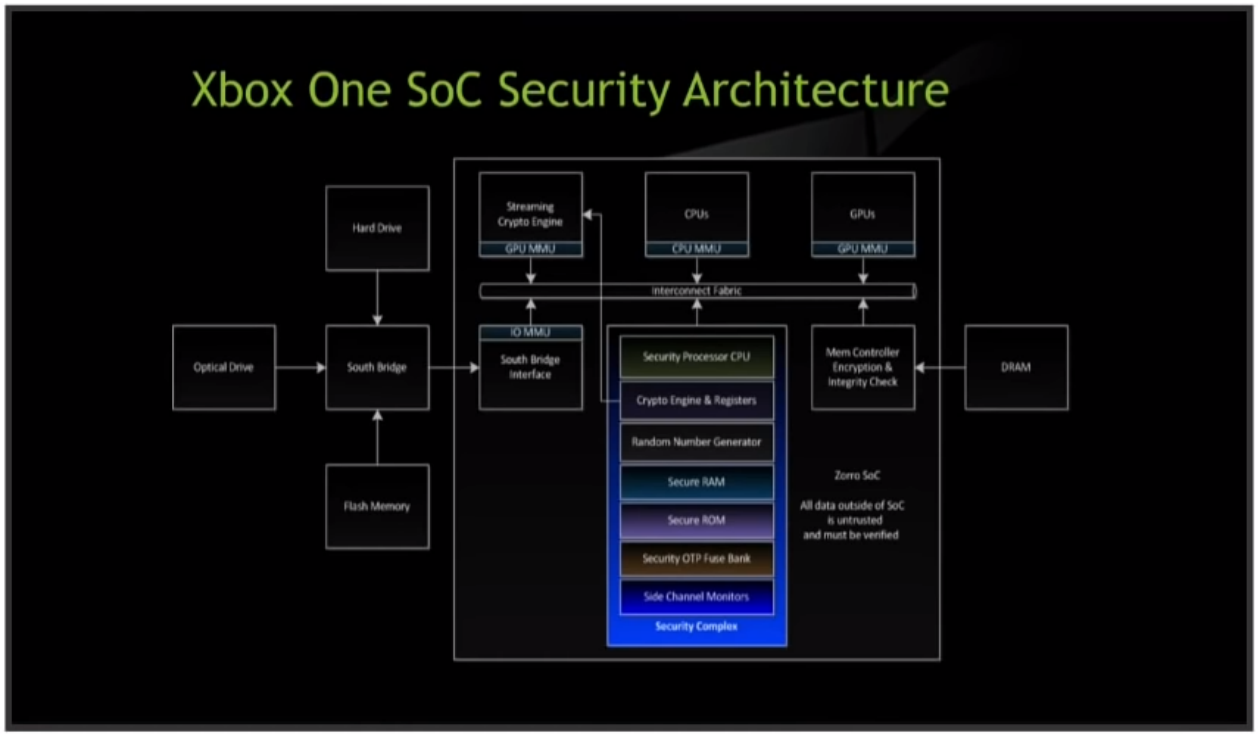
Fig. 2. A hierarchy of computational levels.

Root-Of-Trust

- “Root-Of-Trust” чипы - основа аппаратной безопасности: TPM, DRM, etc . . .
- Что мешает применять это прямо сейчас? Централизация, фокус только на DRM, у каждого производителя своя архитектура/протоколы

Tony Chen
Microsoft

Platform Security Summit
Oct 1–3, 2019 • Redmond, WA



PHCM

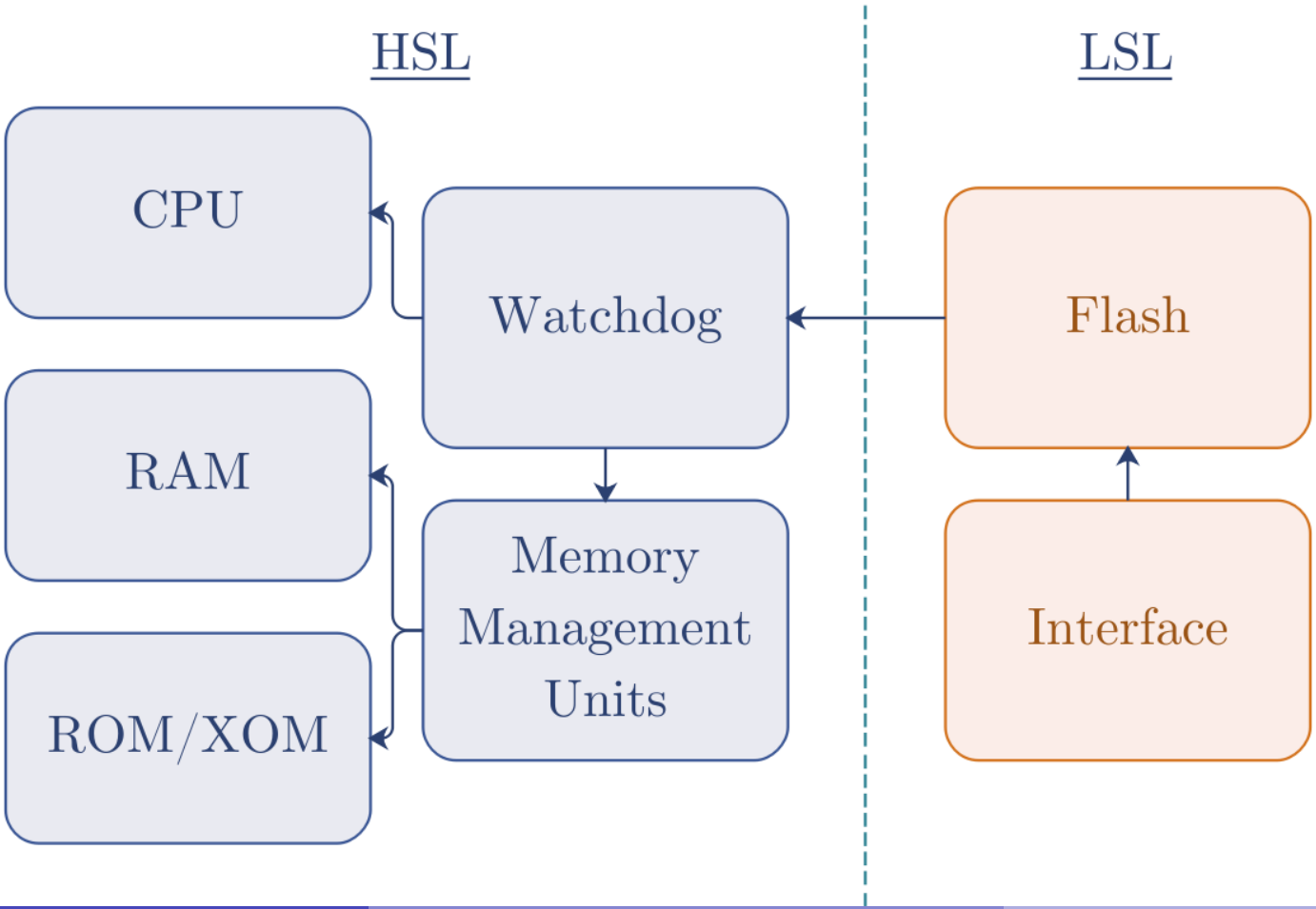
“Pseudo-Homomorphic Computational Module”

– это “Root-Of-Trust” чип с фокусом на децентрализацию, унифицированным протоколом и возможностью гомоморфных вычислений

- Производитель PHCM не имеет полного контроля над ним. Делаем вещи только с согласия пользователя. Производитель софта не зависит от производителя PHCM.
- Изоляция это хорошо
- Рыночек и институт репутации порешает нечестных производителей

Топология PHCM

*Это более абстрактная топология чипа, а не инженерная спецификация



Топология PHCM

- LSL: хранит зашфированные контейнеры для дальнейшего исполнения HSL
- HSL: джентельменский набор “Root-Of-Trust” чипа: watchdog, MMU, hidden ROM/XOM, RAM, CPU
- Обладает приватными ключами: индивидуальным(γ) и общим(α)
- Обладает публичным ключом сертификатора/сервера (β)
- LSL может быть симулирован средствами софта

В чем профит?

- Унификация протокола, прозрачность и конкуренция на ROT рынке
- Гомоморфные вычисления на PHCM намного быстрее стандартных, т.к. временная сложность не меняется
- Производитель PHCM не имеет доступа к контейнерам, безопасность для пользователя и разработчиков софта ставится во главу угла
- Сертификация на уровне вычислений для защиты приватности в интернете

Несколько определений

Программа

Вектор $\mathfrak{B}^T \in \{0,1\}^n$, реализующий машину Тьюринга T называется программой.

Лемма 1. Код на любом Тьюринг-полном языке программирования можно привести к виду $\mathfrak{B}^T$.

Док-во. Перепишем код на другом Тьюринг-полном языке – Brainfuck. Его синтаксис состоит из символов $\{+, -, <, >, [,]\}$, переведем полученный код в 5-ричное число, которое затем переведем в 2-ичную СС.

Несколько определений

Лемма 2. Можно свести пользовательский ввод к программе, т.е. если $I \in \{0,1\}^p$ – ввод, а $\mathfrak{B}^T$ - программа ожидающая ввода размера $P = inp\left(\mathfrak{B}^T\right)$, утверждается, что существует такая функция $i\colon \{0,1\}^n \times \{0,1\}^p \mapsto \{0,1\}^k$, которая сопоставляет паре $\left(\mathfrak{B}^T, I\right)$ программу $\mathfrak{B}^{T\prime}$, ожидающую ввода размера $max(0, P - p)$.

Док-во. После i-го $(i \leq p)$ пользовательского ввода, мы будем присваивать переменной получившей ввод предопределенную константу I_i. Таким образом, значение первых $min(dim(I), inp(\mathfrak{B}^{T\prime}))$ вводов пользователя не важно.

Таким образом все нижеизложенные ситуации можно свести к виду программ.

- Необходимо вычислить контейнер написанный на Тьюринг-полном языке программирования с предопределенным вводом, не предпологая пользовательского вмешательства
- Необходимо вычислить контейнер написанный на Тьюринг-полном языке программирования с частичным вводом пользователя(в том числе и полным)

Несколько определений

Протокол

Пусть П - протокол взаимодействия PHCM, тогда для каждого субъекта i обозначено $\Pi[i] \in \mathcal{P}\left(\{\alpha_d, \alpha_e, \beta_d, \beta_e, \gamma_d, \gamma_e\}\right)$ - ключи которыми он располагает

Защищенные туннели

$\mathfrak{C}_{I,J}(\phi, \eta)$ - защищенный туннель между субъектами I и J на основе RSA+AES с RSA-ключами ϕ, η: $\{\phi_d, \eta_e\} \subseteq \Pi[I], \{\phi_e, \eta_d\} \subseteq \Pi[J]$

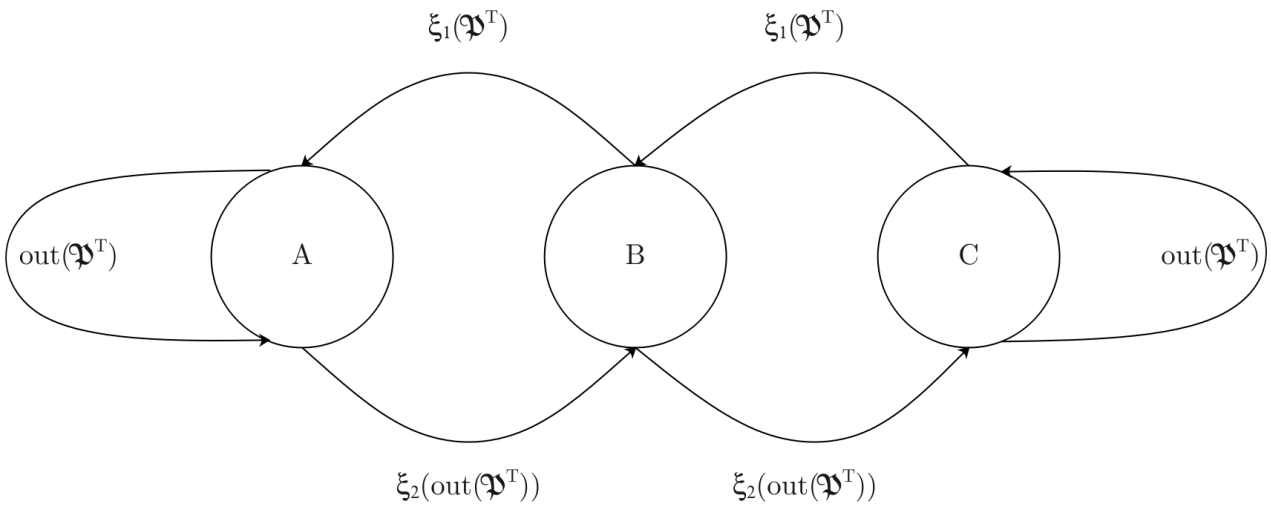
А так-же определим $\overline{\mathfrak{C}_{I,J}}$, как защищенный туннель со случайно сгенерированными ключами, которые не представляют значения вне этого туннеля.

Протокол

Что нам нужно от протокола?

- Безопасность и стабильность: handshake система, RSA+AES шифрование
- Унификация и децентрализация: Blockchain, BitTorrent, цифровые подписи

Как примерно он должен работать?



Протокол: Security Handshake

Security Handshake

Handshake - опреация в ходе которой две стороны убеждаются в легитмности друг друга.

- Сервер (C) убеждается в том, что HSL распологает ключами α_d, γ_d
- Зона высокой безопасности (HSL) убеждается в том, что C распологает ключом β_d

Определим *rand* () как случайное число, а $\mathfrak{s}(k, i_d)$ как электронную подпись значения k RSA-ключом i_d.

$$\mathfrak{s}'(k, i_{d1}, i_{d2}, \ldots, i_{dn}) = (k, \mathfrak{s}(k, i_{d1}), \mathfrak{s}(k, i_{d2}), \ldots, \mathfrak{s}(k, i_{dn}))$$

Протокол: Security Handshake

Ниже представлен протокол выполнения процедуры Handshake (в канале $\overline{\mathfrak{C}_{HSL,C}}$):

1. $HSL \to C$: $RSA_e\left(\mathfrak{s}'\left(\{\beta_e, \gamma_e, r\}, \alpha_d, \gamma_d\right), \beta_e\right)$
2. C: проверяет корректность подписей $\mathfrak{s}'\left(\{\beta_e, \gamma_e, r\}, \alpha_d, \gamma_d\right)$, присутствие γ_e в базе данных пользователей и соответствие β_e. Затем генерирует сервер случайное значение $r_1 = rand\,()$ и создает сессию для ключа γ, с ключом сессии r_1. Если все прошло успешно, C должен быть уверен в легитимности HSL.
3. $C \to HSL$: $RSA_e\left(\mathfrak{s}'\left(\{\gamma_e, r_1, r\}, \beta_d\right), \gamma_e\right)$
4. HSL: проверяет корректность подписей $\mathfrak{s}'\left(\{\gamma_e, r_1, r\}, \beta_d\right)$. Теперь и HSL убежден в C.

Проткол: Security Handshake

Сценарии атаки:

- HSL - нелегитимен, тогда на 2 этапе ему будет необходимо либо перехватить подпись настоящего HSL, либо получить её при имперсонации сервера. Т.к. аутентификация сервера происходит используя секретный ключ сессии r_1, который HSL получает в зашифрованном виде, атакующий не сможет подтвердить свой статус
- Если сервер нелегитимен, возможно только запоминание подписей и попытка переиспользования, рассмотренная ранее

Протокол: Request

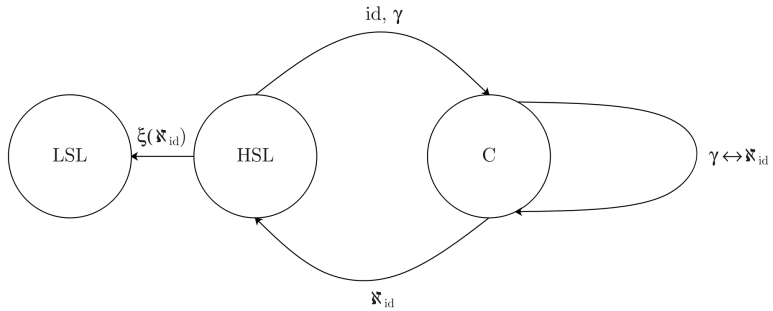
Request

Метод получения контейнера напрямую от сервера. Необходим для гарантии того, что у пользователя есть хотя-бы один способ получить контейнер.

Algorithm 1 Request $\aleph^{TU}$

Require: r ключ сессии HSL, id уникальный ID запрашиваемого контейнера

$s \leftarrow session(r)$
$\gamma \leftarrow s_{\gamma}$
$\aleph^{T} \leftarrow container(id)$
$licenses \leftarrow \aleph^{T}.licenses$
$userlicense \leftarrow \emptyset$
for $i = 1 \ldots |licenses|$ **do**
  if $licenses_{i\gamma} = \gamma$ **then**
    $userlicense \leftarrow licenses_{i}$
  end if
end for
if $userlicense.ts\_expiration > time()$ **then**
  $\aleph^{TU} \leftarrow \langle firmware, maxlen, \mathfrak{P}^{T}, userlicense.ts\_issued, userlicense.ts\_expiration \rangle$
  return $\mathfrak{s}'\left(\xi\left(\aleph^{TU}, \gamma, \alpha\right)\right), \beta, \alpha)$
else
  return 0
end if



Протокол: Execute

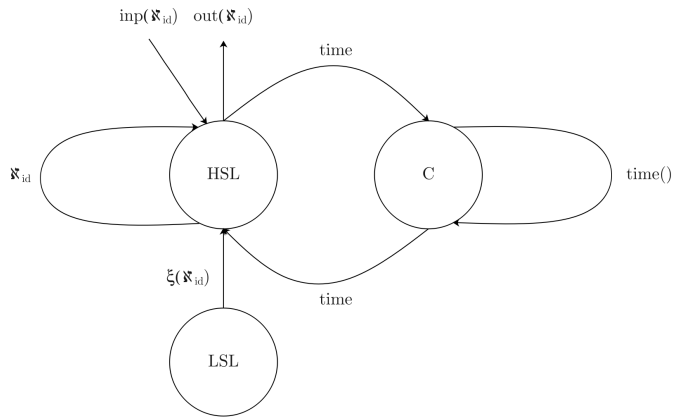
Execute

Метод исполнения зашифрованного контейнера, который каким-то образом уже попал к пользователю (Request/Blockchain).

Algorithm 2 Execute $\aleph^{TU}$

Require: *id* ID копии контейнера в *LSL*.

$\aleph^{TU}_{enc} \leftarrow LCL.request\,(id)$
if $not\,\left(check\_signatures\,\left(\aleph^{TU}_{enc}, \alpha, \gamma, \beta, \right)\right)$ **then**
  return 0
end if
$\aleph^{TU} \leftarrow \xi_d\,\left(\aleph^{TU}_{enc}, \gamma, \alpha\right)$
$expiration \leftarrow \aleph^{TU}.ts\_exp$
$time \leftarrow C.request\_time\,()$
if $expiration > time$ **then**
  $input \leftarrow inp\,()$
  if $|input| <= \aleph^{TU}.max\_len$ **then**
    return $exec\,\left(\aleph^{TU}, input\right)$
  else
    return 0
  end if
end if



Протокол: Децентрализация

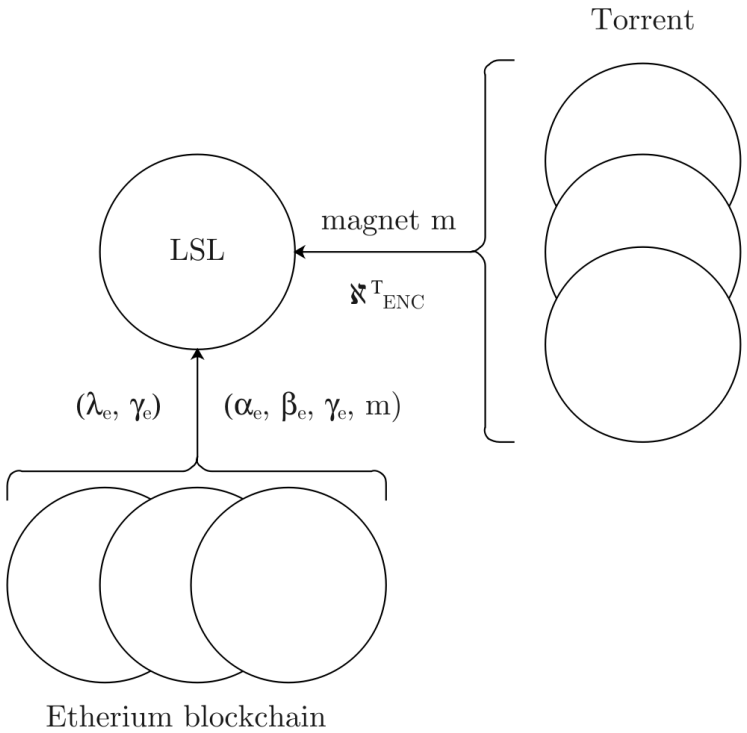
Blockchain/BitTorrent сеть

Используется для большей независимости создателей ПО/пользователей от сертификаторов “Root-Of-Trust” чипов.

Введем новый ключ RSA λ – индивидуальный для каждого пользователя, он будет использовать его (публичную часть) как свой логин, оформляя на него лицензию. Так-же зададим создателя ПО Ф, обладающего ключом ϕ. Итак пусть Ф сотрудничает с сертификаторами $P = \{\beta_1, \beta_2, \ldots, \beta_n\}$, причем каждый сертификатор однозначно характеризуется его ключом β.
Предположим пользователь обладающий ключом λ заключил сделку с Ф и стал обладателем его лицензии в k-экземплярах сообщив ему k пар вида (β, γ), причем $\beta \in P$. В таком случае далее представлен сценарий действий Ф:

Протокол: Децентрализация

1. Ф регистрирует лицензии своим ключом ϕ для всех k пар и запрашивает с каждого сертификатора данные $(\alpha_e, \beta_e, \gamma_e, \aleph_{enc}^{TU}) \subseteq L$, где L множество лицензий
2. Ф размещает torrent-раздачу с магнет-ссылкой m
3. Ф посылает в блокчейн запрос, подписанный своим ключом ϕ: `register {"user":` λ_e `, "licenses": {{` $\alpha_e, \beta_e, \gamma_e,$ `m}, ...}}`



Оракулы

Оракулы

Оракул - это особый контейнер код которого публичен. Он отличается от всех остальных тем, что C хранит лист доверенных контейнеров в виде хешей и соответствующих им ключей.

Когда пользователь получает PHCM, он делает запрос сертификатору C вернуть список оракулов и получает:
$\mathfrak{s}'$({ "version": 1, "oracles": {"oracle": "1c5863cd55b5a4413fd59f054af57ba3c75c0698b3851 d70f99b8de2d5c7338f'', "key": $RSA(ok, \alpha)$}, ... } $, \beta_d$)

Оракулы

В таком виде список оракулов может быть отправлен в PHCM, причем любой другой предустановленный список оракулов будет заменен, в случае если “version” последнего превосходит предыдущий. Для запуска оракула приминяется следующий алгоритм:

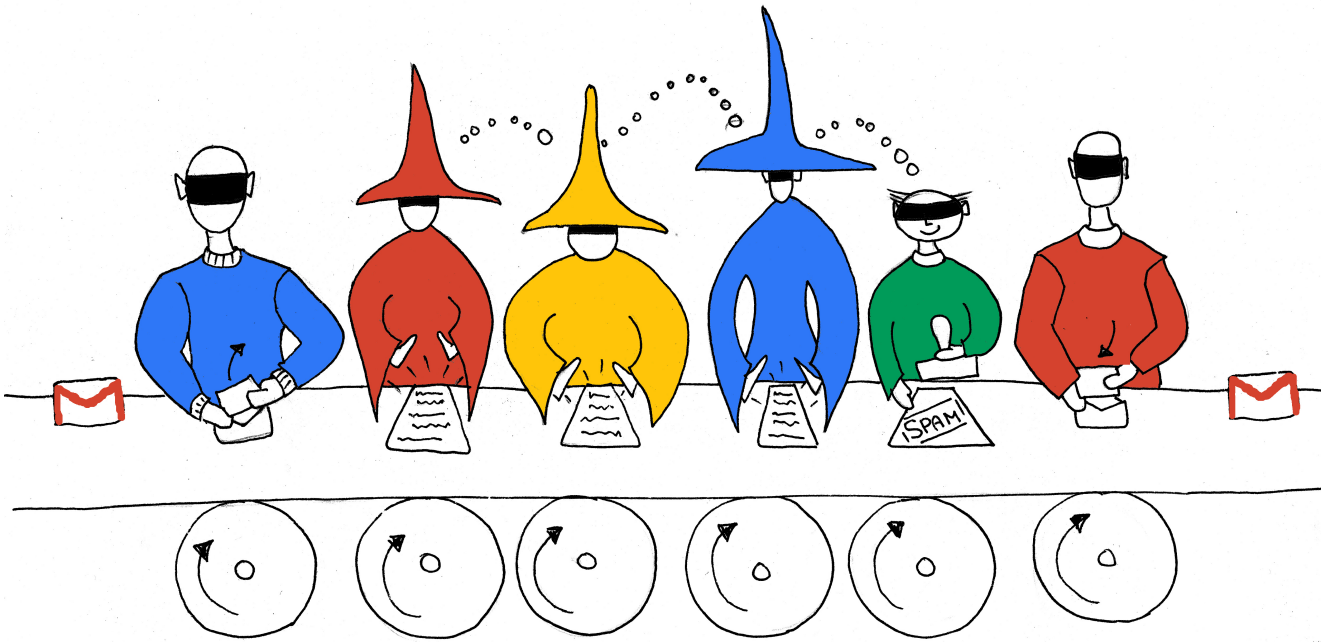
Algorithm 3 Execute oracle O with hash h

Require: h, O (O - код оракла, полученный от пользователя)

```
O_h ← sha256(O)
if O_h = h then
    ok ← ∅
    for i = 1 ... |oracle_list| do
        if oracle_list.oracle = h then
            ok ← oracle_list.key
        end if
    end for
    if ok = ∅ then
        return 0
    end if
    ok ← RSA_d(ok, α_d)
    exec_with_input(O, ok)
else
    return 0
end if
```

Оракулы: Свойства

- Вы всегда можете подтвердить, что результат получен определенным оракулом, т.к. у них есть свои ключи
- Вы всегда можете узнать исходный код оракула и что он делает с вашими данными
- При этом только вы решаете доверять оракулу или нет
- Могут работать с зашифрованными данными (!), если вы им доверяетсе



Оракулы: Приминения

- По определению решают все задачи для которых нужно гомоморфное шифрование: SMPC, ZKP, etc ...
- Платформа для любых крипто-задач: подписи, хранилища данных, Anti-Evil Maid Attack, etc ...
- Доверенно-прозрачные вычисления: изолированные open-source сервера без утечек данных с Machine Level Security

Оракулы: Пример

Secure Multi-Party Computation

В SMPC участвуют N участников $p_1, p_2, \ldots, p_N$. У каждого участника есть тайные входные данные $d_1, d_2, \ldots, d_N$ соответственно. Участники хотят найти значение $F(d_1, d_2, \ldots, d_N)$, где F — известная всем участникам вычислимая функция от N аргументов. Допускается, что среди участников будут получестные нарушители, то есть те, которые верно следуют протоколу, но пытаются получить дополнительную информацию из любых промежуточных данных.

Algorithm 4 Oracle $SMPC$: $O(n) + O(f)$

$func \leftarrow inp()$
$parties\_am \leftarrow inp()$
$parties \leftarrow \emptyset$
$ok \leftarrow request\_oracle\_key()$
for $i = 1 \ldots parties\_am$ **do**
  $user\_pubkey \leftarrow inp()$
  $parties = parties \cup \{user\_pubkey\}$
end for
$out(ok_e, alpha_e, gamma_e, beta_e)$
$data, proofkeys \leftarrow \emptyset$
for $i = 1 \ldots |parties|$ **do**
  $T \leftarrow establish(\mathfrak{C}_{O,i}(ok, parties[i]))$
  $magic\_val \leftarrow T_{i \to O}(inp())$
  $salt \leftarrow rand()$
  $out(T_{O \to i}(func, magic\_val, parites[i], salt))$
  $salt', proofkey, value \leftarrow T_{i \to O}(inp())$
  if $salt' \neq salt$ **then**
    return 0
  end if
  $proofkeys = proofkeys \cup \{proofkey\}$
  $data = data \cup \{val\}$
end for
$result \leftarrow exec(f, data[1], data[2], \ldots, data[n])$
$output \leftarrow \mathfrak{s}'((result, func), proofkeys[0], \ldots, proofkeys[n])$
$out(output)$

Конец

- github.com/rusyaew/PHCM_Salieri
- gleb@gleb.tk
- pip3 install PHCMlib
- Спасибо за внимание <3