ИКТ 004: Псевдо-гомоморфный вычислительный модуль и его применения.

Русяев Глеб

Задача

Создание концепта псевдо-гомоморфного вычислительного модуля (PHCM), т.е. устройства, способного производить безопасные “outsource” вычисления (в том числе на секретных данных и используя секретный алгоритм вычисления результата).

Главная проблема которую решает PHCM: Как нам производить безопасные вычисления в небезопасном окружении (подверженном физическим атакам)? Решение этой проблемы позволяет нам не только создать anti-tampering системы нового поколения (сводя проблемы защиты программного обеспечения (software) к защите аппаратных средств (hardware)), но и новые сервера для облачных вычислений, ставящие конфиденциальность пользовательских данных на первое место, при этом сохраняя всю ту же производительность.

Последние открытия в области гомоморфного шифрования, а именно работа Крейга Гентри от 2009 года: “Полностью гомоморфное шифрование с использованием идеальных решеток” (Gentry, 2009) позволяли теоретически создать такие системы, однако они были либо практически не реализуемы, либо не эффективны. Таким образом, по сегодняшний день проблема инженерного решения, применимого на практике, оставалась открытой (Shan et al., 2018).

Моя работа предлагает подход, при котором полагаясь на новейшие меры противодействия аппаратному реверс-инжинирингу, мы представляем вычислительный модуль в виде некой черной коробки, к которой мы не можем получить внутренний доступ, однако можем отправлять подписанные электронной подписью сервера (зашифрованные) программы для исполнения на произвольном (возможно зашифрованном) вводе.

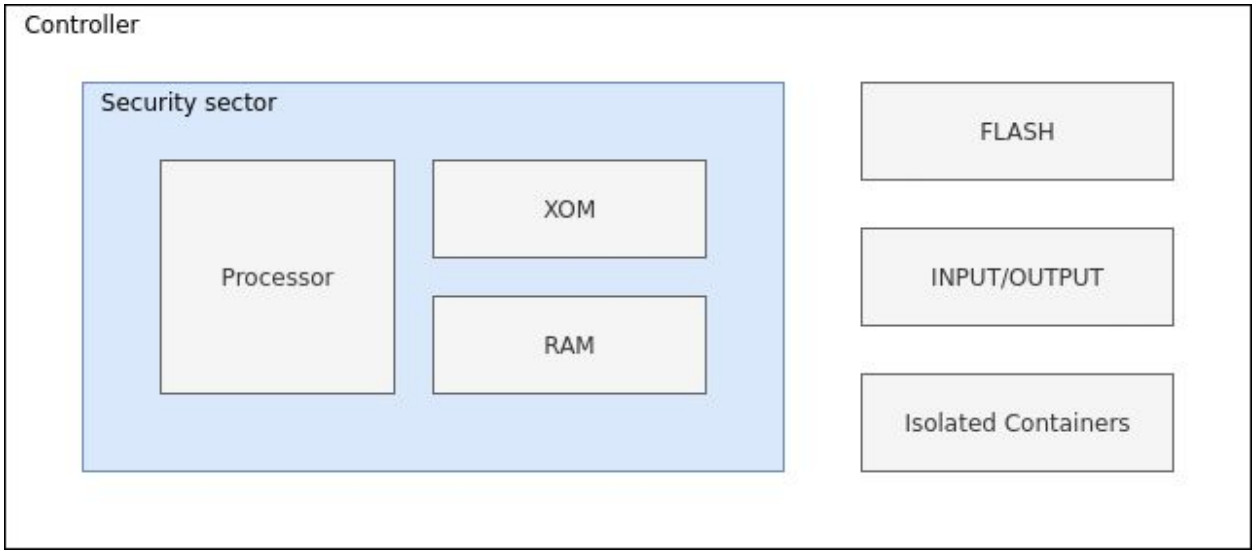
Похожий подход был использован, к примеру, в проприетарных консолях поколения 2013 года для защиты игр от пиратства и показал свою эффективность (Xbox One и PS4 по сей день остаются не взломанными). При этом сами игры хранились в зашифрованном и подписанном виде на

дисках консоли, и единственное место где они были в расшифрованном виде — внутри кремния процессора, делая reverse-engineering сложным и очень ресурсозатратным. Через 9 дней после публикации (отправки на конкурс POCT-ISEF: 8 ноября 2020) черновика этой статьи также был анонсирован Microsoft Pluton (по-сути являющийся закономерным продолжением процессоров Xbox One) (Microsoft, 2020).

В отличии от вышеперечисленных решений, PHCM и эта статья (содержащая его концепт, т.к. у меня недостаточно ресурсов для выпуска процессоров) распространяется под лицензией GNU GPL v3. Как показывает практика именно open-source решения чаще всего оказываются более стойкими к атакам. Также PHCM не акцентирует внимание только на бизнесе, технология позволяет не только защищать компании от недобросовестных пользователей (anti-tampering), но и пользователей от недобросовестных компаний (при облачных вычислениях или при предоставлении каких либо услуг (например анонимный поисковой движок) запрещая серверу сохранять любые данные о клиенте).

Методы и процессы

Первым шагом при разработке концепта стало описание составляющих PHCM. Он подразделяется на две зоны с различными уровнями безопасности — высоким и низким. Разрабатывая зону с высоким уровнем безопасности (HSL), мы предполагаем, что зона с низким уровнем безопасности (LSL) и система пользователя уязвима перед самим пользователем посредством физических атак.



Ядро PHCM - это HSL, она является цельной системой состоящей из кремния, существенно повышая стоимость реверс-инжиниринга (RE) (особенно учитывая то, что RE процессора Intel 8080 завершился только в 2015 году, а процессоры консолей поколения 2013 года до-сих не взломаны). Он состоит из базового процессора (который собственно и занимается исполнением инструкций после расшифровки), который может быть любой архитектуры и иметь любые свойства, в том числе шифрование шины. Также в HSL присутствует XOM (eXecution Only Memory) память (или просто ROM) для перманентной прошивки, Memory Management Units для доступа к памяти (увеличение безопасности), а также оперативная память, содержащая расшифрованные инструкции для процессора. Объединяя все эти 3 компонента в единый кремниевый чип

(для того, чтобы противостоять Man In The Middle атакам между базовым процессором и оперативной памятью), мы получаем HSL.

В задачи HSL входит проверка электронных подписей, расшифровка программ, генерация случайных чисел, watch-dog сервис служащий для защиты от атак подобных fuzzing (по-сути контроль вольтажа) и API для коммуникации с LSL. Прошивка для всего дополнительного функционала может находится внутри LSL в зашифрованном виде, подписанная HSL и сервером, доставляемая на место точно также как и любая программа для исполнения на PHCM.

Самая главная информация, которой обладает HSL — ключи шифрования асимметричного алгоритма (можно использовать RSA или пост-квантовые алгоритмы в долгосрочной перспективе). Остановимся на них поподробнее. При создании некоего класса устройств (при серийном выпуске) PHCM одного производителя, он создает 2 пары ключей — alpha α и beta β, для PHCM и сервера соответственно. Сервер содержит публичный ключ α и приватный ключ β, а PHCM наоборот (приватный α и публичный β). В качестве дополнительной меры защиты (в основном для предотвращения атак, заключающихся в физической замене памяти LSL), на этапе производства, мы вводим еще один случайно генерируемый ключ gamma γ. Он генерируется внутри HSL и используя API, мы получаем публичный ключ γ, отправляя его на сервер.

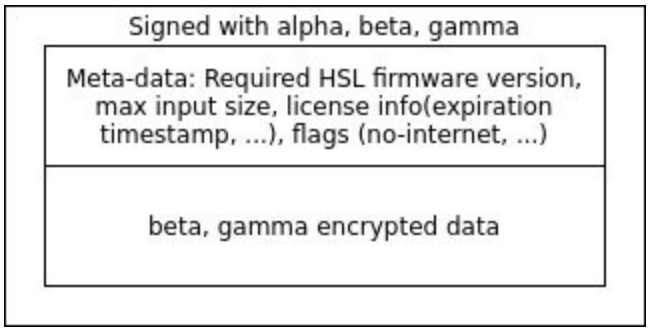
Возвращаясь к предыдущей теме, HSL хранит α-private (приватный ключ alpha) и β-public. Нет строгого ограничения на хранения β ключа в LSL, однако приватный ключ α не должен покинуть HSL и хранится прямо в XOM с запретом на чтение (используя MMU).

Теперь давайте опишем LSL, первое его важное свойство это то, что он управляется другим (незащищенным процессором). Если PHCM не является главным вычислительным устройством, а возможно co-processor или видеокартой (для анти-темперинга игр), то мы можем использовать для этой цели основной процессор компьютера. Если же PHCM сам является основным процессором, то мы воспользуемся дополнительным

низкопроизоводительным процессором (т.к. его спектр задач достаточно узок). LSL состоит из flash-памяти, для его прошивки (возможно содержащая зашифрованные доп. части HSL прошивки) и хранилища для изолированных контейнеров в зашифрованном виде (контейнер это программа + доп. информация), каждый из которых подписан и зашифрован α и γ ключами.

Основные задачи LSL — хранение зашифрованных данных, взаимодействие с HSL и пользователем (по-сути он нужен для самодостаточности PHCM) и его работу теоретически может выполнять сам пользователь (его ПО), но это сужает возможности вычислительного модуля.

Ниже представлена структура контейнера (единицы программного обеспечения).



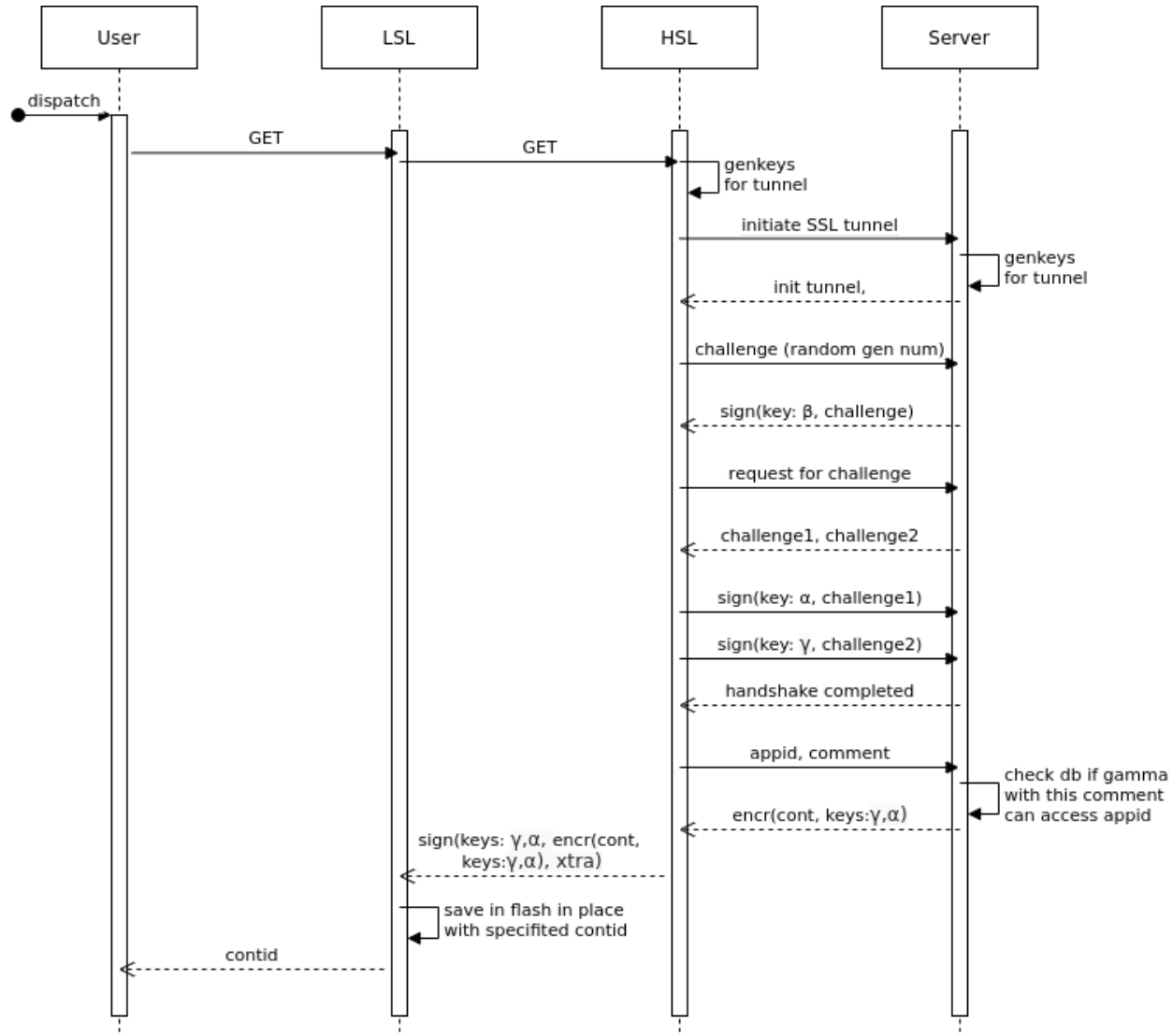
Метаданные являются публичной информацией (для большей прозрачности системы).

Теперь, когда мы разобрались, что из себя представляет контейнер, необходимо описать способ, которым контейнеры попадают с сервера на PHCM.

Функция “GET”, параметры: AppID (уникальный ID контейнера на сервере), адрес сервера (server), ContID (ID слота в хранилище зашифрованных контейнеров), comment (доп. информация, которую необходимо передать серверу, чтобы получить контейнер).

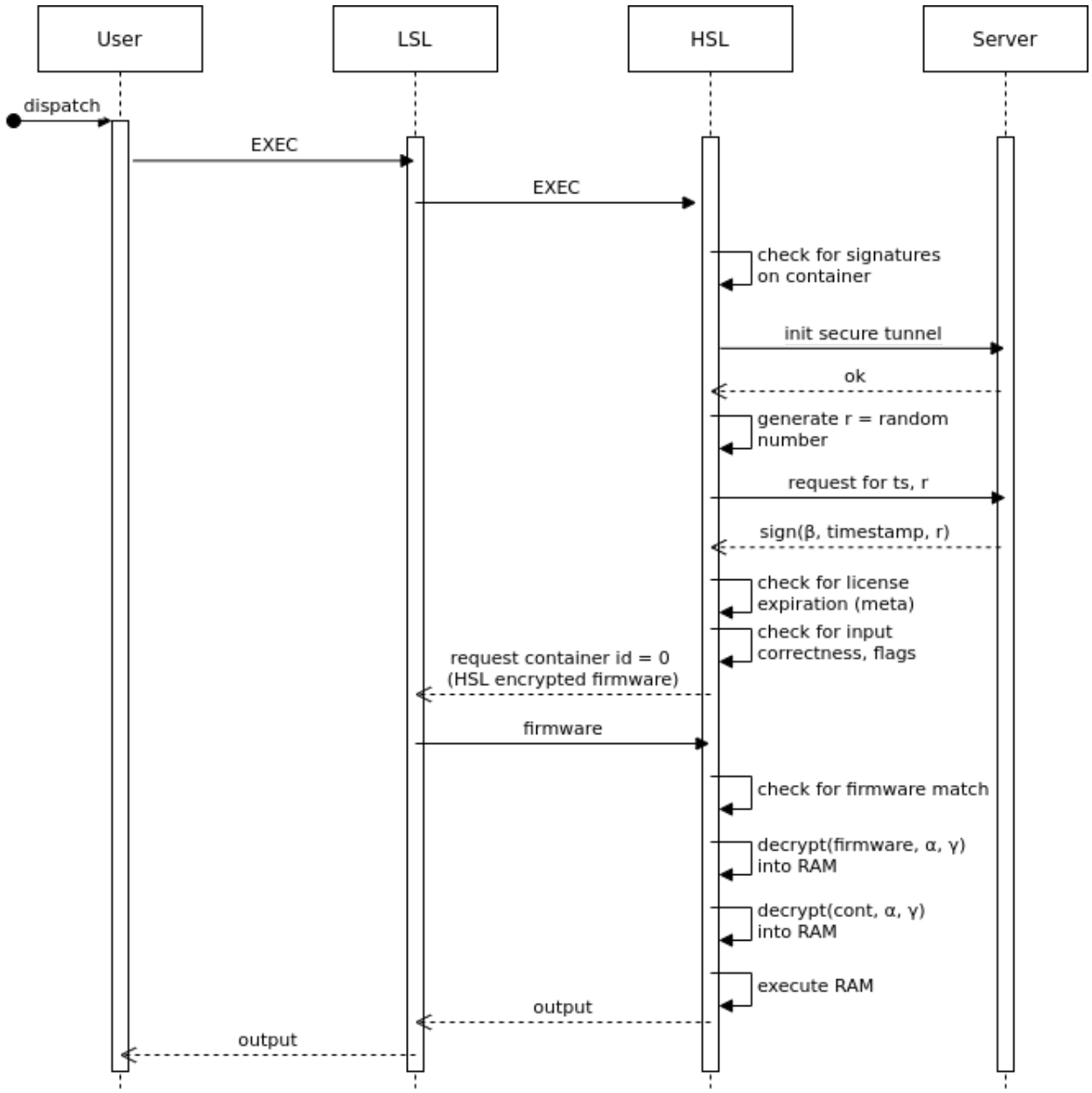
Функция загружает зашифрованный и подписанный контейнер в хранилище (внутри LSL)

На нижеприведенной диаграмме, весь трафик между HSL и сервером проходит через LSL и пользователя, но идет по защищенному туннелю со случайными ключами, а затем используя “challenge” — случайные последовательности байтов, которые клиент либо сервер должны подписать, мы проводим некое рукопожатие между HSL и сервером, в ходе которого две стороны убеждаются в легитимности друг друга.



Функция “EXEC”, параметры: ContID (ID контейнера для воспроизведения), Input (входные данные).

Данная функция описывает воспроизведение контейнера off-line (т.е. без интерактивного ввода). Пользователь при старте указывает ввод и получает вывод.



Такая схема в принципе является общей, однако в случае, если в метаданных контейнера стоит флаг “no-internet”, то мы пропускаем все взаимодействия с сервером (тогда если лицензия не пожизненная, мы не сможем проверить ее истечение).

Также, на всем периоде работы HSL работает watchdog модуль, который занимается мониторингом различных физических показателей внутри HSL, как уже объяснялось ранее, он предназначен для защиты от различных

физических атак (fuzzing, замораживание и т.п.). В случае обнаружения такой атаки, ключи немедленно стираются и могут быть восстановлены производителем. Установка нового beta-ключа, а затем генерация и занесение в базу данных сервера gamma-ключа должны решить эту проблему.

Оракулы. Дополнительные built-in контейнеры, hash (для проверки) и исходный код которых доступен каждому. Они призваны расширить возможности PHCM.

Их главное отличие от обычных контейнеров — повышенные привилегии по сравнению с остальными контейнерами. Прошивка разрешает им расшифровывать и зашифровывать данные используя α и γ ключи (она проверяет контейнер по хэшу и уникальному флагу).

Хэши и приватные ключи таких оракулов записаны в прошивке (той, которая хранится в LSL и предназначается для HSL). Чтобы удостовериться, что именно этот оракул производил вычисления, на сервере хранятся публичные ключи оракулов и их хэши. Приватный ключ будет использован для доказательства вычисления именно на этом оракуле, а публичный для проверки подписей.

Оракул-SMPC (secure multi-party computation).

Есть некоторое кол-во пользователей, им надо посчитать значение функции $f(a_1, a_2, \dots, a_n)$ не раскрывая значений a. При этом пользователи знают функцию f и хотят убедиться, что результат получился именно после ее применения.

В таком случае один из участников (вычислитель) запускает на PHCM данного оракула в режиме интерактивного ввода. Первоначальный вывод составляют бинарное представление функции, кол-во участников и их публичные ключи.

Затем вычислитель отправляет запрос **get_keys**. В ответ на этот запрос, PHCM возвращает два публичных ключа — α и γ. Эти ключи вычислитель

отправляет остальным. Каждый участник делает запрос к серверу, чтобы подтвердить правильность ключей.

Теперь используя тот факт, что у каждой стороны есть публичные ключи друг-друга, установим защищенный туннель между ними (данные передаются как ввод и вывод в PHCM). Затем чтобы проверить функцию, участник запрашивает сообщение с кодом функции и случайным значением, подписанное ключом оракла от вычислителя.

Далее пользователь получает “соль”, которую он прибавляет к предварительно сгенерированному приватному ключу и своему значению a_i (для i-го пользователя). Все это он шифрует используя ключ оракла (получая его с сервера) и отправляет по защищенному туннелю.

PHCM принимает эту информацию, расшифровывает ключом оракла (получая его, доказав прошивке, что это он), сравнивает соль (необходимо для защиты от различных атак, при которых злоумышленник передаст, например, контейнер вместо ввода) и сохраняет значение для дальнейших вычислений. После того как заданное кол-во пользователей отправило свой ввод, мы посчитаем значение функции f при этих входных данных, а результат подпишем ключами α, γ, ключом оракла, а также каждым из приватных ключей участников. Формат сообщения: “OUTPUT” + hash(func) + stdout

Возможности оракулов на этом не заканчиваются, с их помощью можно реализовать решения различных задач из Computer Science (к примеру Zero-Knowledge Proof) и по-сути переконфигурировать PHCM под новые задачи.

Применения

В “стандартном” виде применение ограничивается доверенными облачными вычислениями. Эта проблема стала особенно актуальна в последнее время, когда за 12 лет рынок облачных вычислений вырос с 6 до 236 миллиардов долларов США (Holst, 2020).

Также PHCM может стать частью anti-tampering систем нового поколения, сочетая в себе свободу модификации ПК и безопасность консолей. А тот факт, что для противостояния пиратству, PHCM использует физическую защиту, а не устоявшуюся (но не эффективную) концепцию “Security Through Obscurity” позволяет разрабатывать прозрачные DRM системы.

Более того, используя концепцию оракулов, PHCM может оказаться эффективным и реализуемым на практике решением теоретических проблем криптографии и computer science в целом. В качестве примера решения таких проблем приведен оракул-SMPC, решающий усложненную версию проблемы Эндрю Яо — “Secure Multi-Party Computation”. Ее решение может быть использовано например в P2P-сетях для создания доверенных “supernodes” (такие части сети обладающие особыми правами). Это поможет полностью анонимизировать TOR-сеть (она безопасна внутри, но выходные гейты в интернет могут быть скомпрометированными) или создать “неуязвимые” системы электронного голосования.

Фактически применения ограничены только человеческим воображением, от анти чита для игр до сетей доверенных вычислений и обучения нейросетей на секретных больших данных.

Однако самым интересным применением я считаю децентрализованные файловые сети (подобные интернету), где данные будут распределены между всеми пользователями одновременно, обеспечивая огромные скорости интернета (подобно торренту, только файлы зашифрованы и сжаты).

Так-же мне кажется интересным cloud-streaming системы (по сути являющиеся закономерным продолжением cloud-computing технологий). Сейчас они применяются в основном для игр, но если вычисления будут доверенными, а скорости мобильного интернета достаточно высокими (5G+), вполне возможно, что в будущем ПК, ноутбуки и телефоны будут лишь оболочкой, которая постоянно поддерживает связь с PHCM в каком-то дата центре. Т.е. все они будут запущены на одном и том-же оборудовании, только в разных виртуальных машинах. Это может быть особенно актуально, в связи с трендом на утоньшение устройств. Так самый тонкий ноутбук сможет использовать самое мощное оборудование, независимо от его реального размера.

Bibliography

Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. *Proceedings of the 41st Annual ACM Symposium on Symposium on Theory of Computing*. doi:10.1145/1536414.1536440.

Holst, A. (2020). *Global public cloud computing market 2008-2020*. https://www.statista.com/statistics/510350/worldwide-public-cloud-computing/

Microsoft. (2020, November 17). *Meet the Microsoft Pluton processor – The security chip designed for the future of Windows PCs*. https://www.microsoft.com/security/blog/2020/11/17/meet-the-microsoft-pluton-processor-the-security-chip-designed-for-the-future-of-windows-pcs/

Shan, Z., Ren, K., Blanton, M., & Wang, C. (2018). Practical Secure Computation Outsourcing: A Survey. *ACM Computing Surveys*, *51*, 1-40. doi:10.1145/3158363.