Ю. Ю. Мышко, О. В. Ниссенбаум
Тюменский государственный университет, г. Тюмень
УДК 004.056.5

ГАРАНТИРОВАННАЯ ДОСТАВКА ВЫХОДНЫХ ДАННЫХ С БОЛЕЕ ЧЕМ 1/3 КОРРУМПИРОВАННЫХ УЧАСТНИКОВ В КЛИЕНТ-СЕРВЕРНОМ ПРОТОКОЛЕ MPC

Аннотация. Безопасные многосторонние вычисления (MPC) — это мощный инструмент для защиты данных и процессов в информационных системах. Доказано, что гарантированная доставка выхода достижима при условии, что активный противник коррумпирует не более трети участников. В данной работе мы представляем протокол безопасного ввода для фреймворка безопасных распределенных вычислений, расширяя тем самым порог коррумпированности для клиент-серверных протоколов, где, помимо трети серверов, коррумпированным может быть также клиент.

Ключевые слова: MPC, полная безопасность, общий доступ, безопасный ввод, аутентификация.

Введение. Протоколы безопасных многосторонних вычислений (Secure Multiparty Computation — MPC) позволяют взаимно не доверяющим сторонам совместно вычислять функцию своих частных входных данных. Теоретические основы MPC были заложены в конце 1980-х годов [1]. Первое практическое крупномасштабное применение состоялось только в 2008 году на датском аукционе сахарной свеклы. В последующие годы использование MPC резко возросло и нашло применение во многих важных областях науки и техники.

Проблема исследования. Несмотря на то, что многосерверные приложения являются естественным применением MPC, не так много решений было опубликовано, и очень немногие из них с гарантированной доставкой выхода. В этой работе мы рассмотрим, как можно безопасно использовать фреймворк, когда вычисления выполняются серверами, а данные предоставляются клиентом, который также может быть поврежден. Это может привести к превышению порога коррумпированных участников фреймворка.

Классификация противника. Конструкция и свойства протокола MPC в значительной степени зависят от модели противника. Противник коррумпирует некоторое подмножество сторон и получить доступ к их ресурсам и данным.

Пассивный, или получестный противник, следует протоколу, но может попытаться получить конфиденциальную информацию. *Активный* противник может отклоняться от протокола произвольным образом.

Противник берет под контроль до t из n сторон, где t называют *порогом коррумпированности*. В лучшем случае безопасности с полным порогом мы предполагаем, что протокол защищен, когда даже $t = n - 1$. Но если мы требуем более строгого условия, такого как честное большинство, $t < \frac{n}{2}$, то становится возможным достичь большей эффективности или/и больших гарантий безопасности.

Гарантии безопасности. Первыми двумя требованиями безопасности любого протокола MPC являются полнота и конфиденциальность. Поскольку противник может отправлять некоторые сообщения по своему выбору, то в качестве транзакций коррумпированных сторон, необходимы некоторые гарантии против вредоносного поведения. Самой слабой из них явля-

ется *безопасность с прерыванием*. Более эффективными являются *честные* протоколы. Лучшей гарантией безопасности является *полная безопасность* или *гарантированная доставка выхода* (guaranteed output delivery — g.o.d.).

Каналы связи. Мы рассматриваем два типа каналов связи: защищенный канал «*точка-точка*» и *широковещательный* канал. Используя широковещательную рассылку, отправитель не может отправлять разные сообщения разным сторонам. В общем случае g.o.d. достижим, когда $t < \frac{n}{2}$ с широковещательным каналом [2]. Без использования примитивов с асимметричным ключом, таких как забывчивая передача, g.o.d. возможен, когда $t < \frac{n}{3}$ [3].

Материалы и методы. В этой работе мы рассматриваем общую гарантию безопасности в клиент-серверных протоколах MPC. Мы рассмотрим протоколы, в которых выделенная сторона, клиент, предоставляет свои данные распределенному серверу для выполнения некоторых вычислений и предоставления результата только клиенту. Под распределенным сервером мы понимаем группу серверов, которые выполняют необходимые вычисления в режиме MPC таким образом, что даже если некоторые из них коррумпированы активным противником, и протокол по-прежнему обеспечивает корректность, конфиденциальность и g.o.d.

Одним из наиболее эффективных статистически защищенных протоколов с g.o.d. является DEN22 [4], который позволяет безопасно реализовать любую вычислимую функциональность в любом конечном кольце с порогом повреждения $t < \frac{n}{3}$. Нагрузка на канал связи в пересчете на выполнение одной многосторонней операции умножения в среднем составляет пересылку менее $1\frac{1}{3}$ элемента из алгебраического кольца, в котором выполняются вычисления. Все линейные операции выполняются локально и не создают нагрузки на канал связи.

На первый взгляд, клиент-серверное приложение MPC могло бы работать следующим образом: клиент предоставляет секретные входные данные серверам, которые выполняют вычисления MPC. Хотя при более тщательном анализе мы можем заметить, что это не всегда безопасно. Например, клиент также может быть коррумпирован и предоставить честному серверу некорректные данные, которые не согласуются с данными других честных серверов. В случае g.o.d., где стороны исключают тех, чьи транзакции несогласованны, это может привести к исключению честных серверов и, следовательно, к ситуации, когда коррумпированные серверы могут повлиять на выход протокола. В частности, в DEN22 с 4 серверами, в многостороннем протоколе умножения серверы проверяют, не предоставил ли один из них долю, которая не согласуется с долями других серверов. Если это так, они помечают его как поврежденный и завершают умножение без него. Если клиент намеренно портит входные данные для одного из честных серверов, это может привести к тому, что два других завершат операцию вместе с поврежденным сервером, где последний может испортить выходные данные.

В этой работе мы адаптируем фреймворк DEN22, дополняя его этапом безопасного ввода, который серверы могли бы выполнять для проверки, возможно, неправильно сформированных данных клиента. Этот шаг в клиент-серверных протоколах позволяет сохранить общую гарантию, даже если повреждение клиента привело к превышению порога $t < \frac{n}{3}$, если все еще более двух третей серверов честны. Мы создаем наш защищенный протокол ввода для минимального случая из 4 серверов с порогом повреждения $t = 1$ и с 5-й стороной — клиентом, который также может быть коррумпирован. Этот шаг может быть расширен для случаев, когда

задействовано более четырех серверов. Наш защищенный ввод реализован как с широковещательной передачей и без нее и является вычислительно безопасным.

Сопутствующая работа. В последние годы был достигнут значительный прогресс в разработке практически эффективных протоколов общего пользования. Гарантированной доставки выхода можно достичь при затрате ресурсов почти как в полу-честном протоколе, когда $t < \frac{n}{3}$, в частности, в случае $n = 4$, $t = 1$, в [5]. Фреймворк DEN22 предоставляет общие данные для общего случая $t < \frac{n}{3}$ для любого n и обходится каждой стороне в отправку только $1\frac{1}{3}$ кольцевых элемента за умножение. Для случая $t < \frac{n}{2}$ построение эффективного общего безопасного протокола является более сложным. При идеальной безопасности протокол [6] требует 5,5 кольцевых элементов на умножение. В [7], при снижении безопасности до расчетной, затрата ресурсов умножения составляет 1,5 и 1 кольцевой элемент (но только при $n = 3$, $t = 1$) соответственно. Однако для этих протоколов требуется широковещательный канал, который является ресурсоемким в случае $t < \frac{n}{2}$.

Несмотря на то, что многосерверная аутентификация является естественным применением MPC, не так много решений было опубликовано, и очень немногие из них с g.o.d. Мы можем указать на [8] в качестве примера на трех серверах с полу-честной безопасностью и g.o.d. в случае DOS одного из серверов и распространение этой идеи на многосерверную настройку [9].

Предварительные сведенья. В этом разделе мы объясняем модель безопасности, некоторые определения, необходимые для понимания нашей работы.

Обозначение. В этой статье мы рассматриваем клиент-серверную инфраструктуру: клиент, которого мы обозначаем как C, и четыре сервера, обозначаемыми как P_1, P_2, P_3 и P_4. Как A мы обозначаем противника. Через $[n]$ мы обозначаем набор $\{1, 2, ..., n\}$.

Аддитивное разделение секрета. Элемент алгебраического кольца x является совместно используемым аддитивно разделенным секретом, если каждый участник P_i $(i \in [n])$ обладает случайным равномерно распределенным в данном кольце x_i, называемым аддитивной долей x таким, что $x = x_1 + x_2 + \cdots + x_n$.

Реплицированное разделение секрета. Элемент кольца x является реплицируемым секретом, разделяемым между 4 сторонами с пороговым значением 2, если существуют его аддитивные доли $x_1, ..., x_4$, и каждая сторона P_i обладает набором $\{x_j\}$, где $j \neq i$, $i, j \in [4]$.

В этой схеме у каждой стороны есть все аддитивные доли секрета x, кроме одной, и каждая пара сторон совместно имеет все доли, необходимые для восстановления секрета. Секрет x, разделяемый в этой схеме, мы обозначаем через $[[x]] = \left\{\{x_j\}_{j \neq i}\right\}_{i,j \in [4]}$, и мы будем обращаться к доле x, принадлежащей P_i, как $[[x]]_i$. Например, $[[x]]_1 = \{x_2, x_3, x_4\}$.

Хэш-функция. В нашем протоколе проверки согласованности мы используем хэш-функцию h, которая моделируется как (глобальный) случайный оракул.

Методика доказательства. Доказательство безопасности протоколов MPC выполняется в парадигме реального и идеального мира. В модели безопасности, основанной на симуляции, сравнивается реальный мир, где стороны выполняют протокол, и идеальный мир, где все вычисления выполняются доверенной стороной, называемой функциональностью. Неформально мы говорим, что протокол безопасен, если для любого противника есть симулятор,

который может имитировать протокол на основе взаимодействия с функциональностью таким образом, что эта симуляция неотличима от протокола в реальном мире для противника в идеальной модели [10] или для среды в универсальной композиции (UC) [11].

Безопасный ввод. Функциональность F_{DEN} принимает входы от честных серверов, восстанавливает входные данные и вычисляет протокол. Если коррумпированный клиент предоставляет входные доли честным серверам, он может заставить их согласиться друг с другом, либо поступить иначе, например, отправить разные значения доли x_1. Так же один из серверов может быть коррумпирован и отправлять испорченные доли. Серверы могут сделать вывод, что C поврежден, если у двух или более сторон есть «испорченные» доли, и прервать протокол. Это прерывание не нарушает g.o.d.

В случае, когда только одна сторона P_j «испортила» доли, серверы не могут решить, происходит ли это из-за коррумпированного клиента, который отправил их честному P_j, или потому, что P_j коррумпирован.

Функциональность безопасного ввода. В этой статье мы улучшаем DEN22, добавляя протокол безопасного ввода. Функция показанная на рис. 1 устанавливает входные данные честных серверов на правильные входные данные, что в сочетании с DEN22 обеспечивает g.o.d. Кроме того, это позволяет противнику, повреждающему клиента, выбрать прерывание протокола, что не разрушает g.o.d. как только клиент получает результат.

$F_{SecInput}$

Входные данные: клиент C вводит x

Вычисления: если A решает повредить C, он предоставляет клиенту вывод x' или $\perp$. Если на вход поступает значение x', функциональность разделяет секрет $[[x']]$ и передает доли соответствующим серверам P_i ($i \in [4]$). Если на вход поступает значение $\perp$, функциональность отправляет $\perp$ всем серверам. Если C не поврежден, то функциональность разделяет секрет $[[x]]$ и передает доли соответствующим серверам P_i.

Выход: каждый сервер P_i выводит то, что функциональность отправила ему.

Рис. 1. Функциональность защищенного ввода $F_{SecInput}$

Безопасный ввод с использованием широковещательного канала. При широковещательной передаче для любого секретного значения x каждый сервер P_i, $i \in [4]$, вычисляет и отправляет другим серверам контрольные суммы дополнительных общих ресурсов, которыми он владеет: $h_j^i = h(x_j), i \neq j$. В результате каждая сторона получает таблицу (см. табл. 1).

Затем каждая сторона может проверить, чьи объявленные доли совпадают друг с другом, а чьи — нет. Множество контрольных сумм входных данных участника P_i мы определяем следующим образом:

$$cs_i = \{h_j^i\}_{i,j\in[4],i\neq j} \quad (1)$$

Таблица 1

Таблица широковещательной передачи

P_1		h_2^1	h_3^1	h_4^1
P_2	h_1^2		h_3^2	h_4^2
P_3	h_1^3	h_2^3		h_4^3
P_4	h_1^4	h_2^4	h_3^4	

Каждая сторона может вычислить контрольные суммы от конкатенации всех долей, которые она получила от клиента. Например, если клиент предоставляет серверам доли [[x]], [[y]] и [[z]], то P_i вычисляет $h_j^i = h(x_j, y_j, z_j)$. Тогда, мы можем упростить обозначение, опустив входные данные и обозначив контрольные суммы как h_j^i. Тем самым мы подразумеваем, что контрольная сумма вычисляются от объединения всех j-х аддитивных долей, которые P_i получил от C.

Далее в уравнении (2) мы формализуем функцию проверки согласованности долей двух сторон:

$$agree(cs_i, cs_j) = \begin{cases} true, \text{если } h_k^i = h_k^j \forall k \in [4]\{i,j\} \\ false, \text{иначе} \end{cases} \quad (2)$$

Каждый сервер может проверить функцию попарного согласования (2), используя таблицу 1, а затем принять решение о продолжении протокола или его прерывании.

Мы приводим протокол безопасного ввода на рис. 2.

$\Pi_{SecInput}$

Участники: клиент C с входом x, серверы P_i, где $i \in [4]$.

Вычисления:

1. C вычисляет $[[x]]_i$ и отправляет P_i, где $i \in [4]$.
2. Каждый P_i вычисляет и передаёт $h_j^i = h(x_j)$ для каждого $j \neq i$.
3. Каждый сервер вычисляет $agree\ (cs_i, cs_j)$ для всех $i, j \in [4]$, $i <$ j.
   (a) Если все результаты верны, то выводится «продолжить»;
   (b) Если существует $k \in [4]$ такой, что для любых i, j : $agree\ (cs_i, cs_j) = false$, либо $i = k$, либо $j = k$, то запустить $\Pi_{Reset}(k)$;
   (c) В противном случае прервать протокол.
4. Каждый сервер локально рандомизирует свои доли $[[x]]_i^* = [[x]]_i + [[0]]_i$

Выход: каждый P_i выводит $[[x]]_i^*$ или $\perp$, C выводит $\emptyset$.

Рис. 2. Протокол защищенного ввода с широковещательной передачей $\Pi_{SecInput}$

Когда объявленные доли только одной стороны P_k отличаются от долей других сторон, их можно обнулить, используя «хорошие» доли, принадлежащие другим сторонам. Мы отме-

чаем, что каждая аддитивная доля x_j принадлежит еще двум сторонам, помимо P_k. Чтобы проверить x_j, P_k может использовать ранее переданные значения хэша из табл. 1. Мы приводим протокол сброса Π_{Reset} на рис. 3.

$\Pi_{Reset}(k)$

Участники: серверы P_i, на вход которым подаются данные $[[x]]_i$ и cs_i, где $i \in [4]$.

Вычисления:

1. Каждый сервер P_i, где $i \neq k$, отправляет значение x_j (где $j \neq i$ и $j \neq k$) на сервер P_k.
2. Получив значение x_j от сервера P_i, сервер P_k проверяет, верно ли выражение $h(x_j) = h_j^i$. Если оно верно, то P_k принимает x_j как аддитивную долю секрета x.

Выход: P_k выводит $[[x]]_k = \{x_j\}$, где $j \neq k$.

Рис. 3. Протокол сброса $\Pi_{Reset}(k)$

Теорема. Функциональность $F_{SecInput}$ надежно вычисляет протокол $\Pi_{SecInput}$ с вычислительной UC-безопасностью в присутствии злоумышленника A, повреждающего один из серверов и/или клиента, предполагая широковещательный канал, где h моделируется как глобальный случайный оракул.

Из-за пространственных соображений мы приведем только набросок доказательства в UC-технике.

Доказательство. (Набросок). Не теряя общности, мы предполагаем, что A портит C и P_4. Затем, получая доли от C, симулятор проверяет, соответствуют ли они друг другу или испорчен только один набор из них. Если да, то он восстанавливает входные данные x' и придает их в идеальную функциональность. В противном случае он отправляет $\perp$. Последнее моделирование можно было бы выполнить, учитывая сообщения C, взятые из первого раунда моделирования. А именно, симулятор получает хэши (отправляя запросы случайному оракулу, как могли бы сделать честные стороны), отправляет контрольные суммы cs_j для $i \in [3]$ A по широковещательному каналу, получает cs_4, а затем прерывает (на $\perp$), сбрасывая общие ресурсы P_4 и/или завершает моделирование. Очевидно, что симуляция идеально совпадает с реальным выполнением и, следовательно, неотличима от окружающей среды.

Защищенный ввод, предполагающий использование только каналов «точка-точка». Протокол защищенного ввода требует широковещательной передачи только для одновременного получения табл. 1. Теперь мы покажем, как обеспечить безопасную передачу с использованием каналов «точка-точка» в среде с четырьмя сторонами, где только одна может быть коррумпирована. Дополнительная возможность, которую получает A в этом протоколе: он может выбрать коррумпированную сторону в качестве отправителя либо отправить какое-то другое сообщение остальным сторонам и заставить их всех получать значение *null*. Мы

приводим протокол $П_{Broadcast}(i, m)$ защищенной широковещательной рассылки для сообщения m с отправителем P_i на рис. 4.

$П_{Broadcast}(i, m)$

Участники: отправитель P_i, сообщение m, получатели P_j, $i \neq j$, $i, j \in [4]$

Вычисления:

1. P_i отправляет сообщение m каждому P_j участнику $(i \neq j)$.
2. Каждый P_j $(i \neq j)$ переотправляет полученное сообщение m каждому P_k участнику $(k \neq i, k \neq j)$.
3. Каждый P_j $(i \neq j)$ участник, получив по 3 сообщения от других участников проверяет значения на согласованность и выносит решение.

Выход:

(a) Если все три сообщения разные, то участник P_k выводит значение *null*

(b) Если хотя бы 2 из них совпадают, то выводит значение большинства.

Рис. 4. Протокол защищенного ввода с каналом «точка-точка» $П_{Broadcast}(i, m)$

Единственное изменение, которое мы вносим в $П_{SecInput}$, заключается в том, что стороны вызывают $П_{Broadcast}(i, cs_i)$ на шаге 2 для всех $i \in [4]$ вместо использования широковещательного канала.

Результаты. Мы модифицировали фреймворк DEN22, дополнив его протоколами безопасного входа, что привело к повышению порога коррумпированных участников. Так же мы привели протокол сброса, который позволяет выявить коррумпированных участников. Безопасность разработанных протоколов была подкреплена теоремой, которая доказана с помощью метода универсальной композиции.

Заключение. Мы рассмотрели, как безопасно использовать фреймворк, где вычисления выполняются серверами, а данные предоставляет клиент, который может быть поврежден. Результаты исследования имеют практическую ценность и могут быть применены для повышения безопасности информационных систем.

СПИСОК ЛИТЕРАТУРЫ

1. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In Proc. of the 20th STOC.,pages 1–10, 1988.
2. T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In Proceedings of the twenty-first annual ACM symposium on Theory of computing, pages 73–85, 1989.
3. D. Chaum, C. Cr´epeau, and I. Damgard. Multiparty unconditionally secure pro-tocols. In Proceedings of the twentieth annual ACM symposium on Theory of computing, pages 11–19, 1988.
4. A. Dalskov, D. Escudero, and A. Nof. Fast fully secure multi-party computation over any ring with two-thirds honest majority. In ACM CCS 22, pages 653–666, 2022.
5. A. Dalskov, D. Escudero, and M. Keller. Fantastic four: {Honest-Majority}{Four-Party} secure computation with malicious security. In USENIX Security 21, pages 2183–2200, 2021.

6. V. Goyal, Y. Song, and C. Zhu. Guaranteed output delivery comes free in honest majority mpc. In CRYPTO'20, pages 618-646. Springer, 2020.
7. E. Boyle, N. Gilboa, Y. Ishai, and A. Nof. Efficient fully secure computation via distributed zero-knowledge proofs. In ASIACRYPT'20, pages 244–276. Springer, 2020.
8. D.-E. F˘al˘ama ̧s, K. Marton, and A. Suciu. Assessment of two privacy preserving authentication methods using secure multiparty computation based on secret sharing. Symmetry, 13(5):894, 2021.
9. O. Sefraoui, A. Bouzidi, K. Ghoumid, et al. Ausdide: Towards a new authentication system for distributed and decentralized structure based on shamir's secret sharing. International J. of Advanced Computer Science and Applications, 13(1), 2022.
10. Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. Tu-torials on the Foundations of Cryptography: Dedicated to Oded Goldreich, pages 277-346, 2017.
11. R. Canetti. Universally composable security. J. ACM, 67(5):28:1–94, 2020.