КІБЕРНЕТИКА

А.В. АНИСИМОВ, А.К. НОВОКШОНОВ

УДК 519.72, 003.26

ДОВЕРИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ СКЛАДЫВАЮЩЕЙ МАШИНЫ. I

Аннотация. Предложено решение проблемы целостности арифметических программ, выполняемых на удаленном вычислительном ресурсе. Подход к решению состоит в замене арифметических операций, таких как умножение и деление, процедурами складывающей машины (addition machine), введенной Р. Флойдом и Д. Кнутом. Вычисления и порядок следования команд подписываются динамической цифровой подписью, гомоморфной по сложению/вычитанию. Верификация цифровых подписей гарантирует обнаружение любых несанкционированных изменений в исходном тексте программы и результатах вычислений.

Ключевые слова: складывающая машина, цифровая подпись, гомоморфная криптография.

ВВЕДЕНИЕ

Появление новых экспоненциально развивающихся информационно-коммуникационных технологий, таких как облачные вычисления, децентрализованный сетевой банкинг, «Интернет вещей», обусловили необходимость решения новых проблем безопасности. Пользователи таких технологий должны быть уверены в надежности и конфиденциальности обрабатываемой информации, переданной на удаленные вычислительные ресурсы. В связи с этим значительно возрастает актуальность методов, гарантирующих доказуемую безопасность и доверительность удаленных вычислений.

В настоящее время один из основных подходов к решению проблем безопасности вычислений ориентирован на использование гомоморфной криптографии. Гомоморфное шифрование представляет собой отдельный вид шифрования, позволяющий выполнить определенные операции над шифротекстом и получить зашифрованный результат, который при декодировании совпадает с результатом соответствующей операции, проведенной над исходными данными. Несмотря на многообещающие стартовые продвижения в этой области [1–10], известные методы гомоморфной криптографии не имеют существенного практического применения. Возможно, это связано с тем фактом, что свойство полной гомоморфности вычислений относительно основных арифметических операций находится в противоречии с требованиями доказуемой криптографической стойкости и временны́ми ограничениями вычислительных процессов.

В то же время известны и хорошо изучены полугомоморфные криптографические схемы, в которых гомоморфность выполняется только по одной из операций: как правило, по сложению или умножению. К таким схемам относятся многие системы с публичными ключами: RSA [11] (без недетерминированных добавок типа OAEP [12]), схема Эль-Гамаля [13], схема Пэйе [14], схема Бенало [15]

и ряд других. Однако реальные вычисления требуют значительно более широкого набора арифметических операций, что не позволяет прямого использования указанных полугомоморфных схем.

В настоящей работе исследуется проблема целостности удаленных вычислений, т.е. подтверждаемого соответствия процесса удаленного выполнения программы пользователя ее исходному заданию. Сами данные не шифруются. В рассматриваемой модели вычислений все переменные и константы — целые числа, разрешаются условные конструкции типа if-then-else и циклы типа while-do. Разрешенные операции — только сложения и вычитания на ограниченном количестве регистров. Такая модель вычислений была введена в работе Р. Флойда и Д. Кнута [16] и получила название «складывающая машина» (addition machine). Несмотря на ограниченный набор операций складывающей машины, в [16] показано, что через ее операции с приемлемой вычислительной эффективностью можно выразить более сложные арифметические операции, такие как умножение, целочисленное деление, нахождение остатка по модулю, наибольшего общего делителя, возведение в степень по модулю. Кроме того, не представляет трудности расширение области действия складывающей машины на рациональные и вещественные числа. (Подробное описание складывающей машины будет дано в Приложении 1 части II данной статьи.) Таким образом, промоделировав с помощью складывающей машины операции реальной программы, можно решить определенные проблемы безопасности удаленных вычислений с использованием криптографических схем, гомоморфных только по сложению/вычитанию. Однако семантика неравенств не моделируется сложениями и вычитаниями, поэтому складывающая машина не может рассматриваться как универсальный подход к решению глобальной проблемы полигомоморфной криптографии.

В настоящей работе предлагается решение проблемы целостности вычислений для складывающей машины. Разработан вариант динамической цифровой подписи вычислений, которая изменяется в соответствии с вычислительным процессом. Любые несанкционированные вмешательства в результаты или текст исходной программы (изменение значений переменных или констант, нарушение порядка выполнения команд, внесение новых переменных или команд) приведут к нарушению верификационных условий цифровой подписи. Для получения зашифрованных данных цифровой подписи выбрана схема Бенало [15]. Данные исходной базовой программы не шифруются.

ВЗАИМОДЕЙСТВИЕ ПОЛЬЗОВАТЕЛЯ С УДАЛЕННЫМ ВЫЧИСЛИТЕЛЬНЫМ РЕСУРСОМ

Передача вычислительных полномочий, включая компиляцию программ, удаленному ресурсу предполагает определенную высокую степень доверия к поставщику услуги. Однако во многих случаях, требующих особой защищенности, этого может быть недостаточно для гарантированной защиты исходной программы от несанкционированных изменений.

Пользователь предполагает следующие типы угроз.

1. Злоумышленник может получить доступ к тексту программы до ее выполнения вычислительным ресурсом. Также он имеет доступ к результатам вычислений. Полагаем, что во время выполнения программы вычислительный процесс не искажается.

2. Злоумышленник может попытаться:

- изменить исходные значения переменных или констант;
- ввести новые или уничтожить исходные переменные;
- ввести новые или уничтожить исходные команды;
- изменить порядок выполнения команд программ;
- изменить результаты вычислений, передаваемые после выполнения программы.

Для предотвращения подобных действий злоумышленника пользователь перед отправкой исходной программы вычислительному ресурсу осуществляет преобразования программы путем вставок в специальные места зашифрованных

значений новых контрольных переменных. Эти контрольные переменные отражают логику управляющих команд программы и используются в формировании цифровых подписей. На результаты исходной программы преобразования контрольных переменных не влияют. Удаленному ресурсу на обработку посылается такая видоизмененная программа. Согласно результатам вычислений пользователь проверяет соответствие полученных открытых значений и расшифрованных значений контрольных переменных. На основании сравнения делается вывод о корректности выполненных вычислений.

ОБЩЕЕ ОПИСАНИЕ МЕТОДА

Идея предлагаемого подхода состоит в замене основных арифметических операций над целыми числами (умножения, целочисленного деления, вычисления остатка от деления и других) соответствующими процедурами складывающей машины. Дополнительно при этом используется динамическая цифровая подпись, осуществляемая гомоморфной по сложению криптографической функцией. Действия над цифровыми подписями выполняются в обычной арифметике вычислительной машины. Цифровая подпись переменной должна учитывать три идентифицирующих показателя: текущее значение переменной, имя переменной и место в программе, где эта переменная меняла свое значение.

Пусть f — недетерминированное отображение множества целых чисел Z в мультипликативную группу вычетов по модулю n, $f : Z \to Z_n^*$. Отображение f строится на основе односторонней функции с потайным входом (trapdoor function). Это означает, что величина $y = f(x)$ легко определяется, но эффективное вычисление x по y требует знания секретных параметров (лазейка), известных только пользователю.

Предполагаем, что f задает отображение аддитивной группы целых чисел Z в мультипликативную группу вычетов Z_n^*, гомоморфное по сложению, т.е.

$$f(x+y) = f(x)f(y),\ f(x-y) = f(x)f^{-1}(y).$$

Здесь и далее под умножением значений функции f понимается операция умножения в группе Z_n^*. На Z_n^* определена верификационная функция $V : Z_n^* \to Z_n^*$. Считаем, что пара (x, y), $x \in Z$, $y \in Z_n^*$, корректна, если $y = V(f(x))$.

В настоящей работе выбрана следующая схема отображения f. Для $x \in Z$ полагаем $f(x) = g^{ax} s \bmod n$, где s — произвольный элемент собственной подгруппы S группы Z_n^*, g — фиксированный элемент, $g \notin S$, a — фиксированный секретный параметр. Верификационная функция V задается формулой $V(y) = y^c \bmod n$, где c — порядок подгруппы S. Параметры g, a, c, S секретные, известны только пользователю. Верификация значения переменной x осуществляется проверкой соотношения $E^c(x) = k_x f^c(x) \bmod n$, где $E(x)$ — цифровая подпись x, вычисляемая в процессе выполнения программы, k_x — константа, идентифицирующая имя переменной x. (Более точное описание для отображения f будет дано в разделе описания схемы Бенало части II данной статьи.)

Стартовой цифровой подписью значения x считаем значение $f(x)$. В дальнейшем к этой подписи добавим идентификацию имени переменной.

Предполагаем, что, зная значение переменной x, принимающей значения из Z, и значение $y = f(x)$ из Z_n^*, злоумышленник не может подменить пару этих значений, поскольку он не имеет доступа к закрытым параметрам f. Отображение f может рассматриваться как динамический вариант хеш-функции.

Для защиты от искажения наименований переменных дополнительно вводится идентифицирующая цифровая подпись $Id(x)$ имени переменной x, $Id(x) = f(r_x)$, где $r_x \in Z$ — случайно выбираемый секретный параметр, r_x — константа, фиксированная для каждой переменной x.

Цифровая подпись переменной x задается выражением

$$E(x) = Id(x)f(x) = f(r_x)f(x),\ E(-x) = E^{-1}(x) = f^{-1}(r_x)f^{-1}(x).$$

Изменение цифровой подписи $E(z)$ после выполнения команды присваивания складывающей машины задается следующим образом:

- команда $z \leftarrow x + y$;

$$\text{подпись } E(z) = Id(z)Id^{-1}(x)Id^{-1}(y)E(x)E(y); \tag{1}$$

- команда $z \leftarrow x - y$;

$$\text{подпись } E(z) = Id(z)Id^{-1}(x)Id(y)E(x)E^{-1}(y); \tag{2}$$

- команда $z \leftarrow x$;

$$\text{подпись } E(z) = Id(z)Id^{-1}(x)E(x). \tag{3}$$

Выполнение каждой команды присваивания складывающей машины сопровождается предварительным вычислением соответствующей цифровой подписи — (1), (2) или (3), ассоциированной с данной командой. При этом результирующая переменная z фиксируется своим идентификатором и цифровой подписью полученного числового значения.

Начальное значение x подписывается $E(x)$.

Константы, применяемые в программе складывающей машины, рассматриваются как переменные, принимающие фиксированное значение. Эти значения также подписываются как соответствующие переменные. Введенная цифровая подпись не позволяет злоумышленнику изменять имена переменных в командах присваивания или вводить новые переменные.

Для контроля целостности команд и их порядка следования при выполнении программы осуществляется предварительная числовая разметка всех команд программы. Каждой команде сопоставляется уникальное целое положительное число. Выбор нумерующих чисел предполагается случайным. Команде с выбранным номером i сопоставляется константный идентификатор $Id(i) = f(i)$.

Результирующей переменной z, используемой в команде присваивания с номером i, соответствует значение $Id^{-1}(i)E(z)$. Таким образом, нумерованной команде $i: z \leftarrow x + y$ соответствует значение

$$Id^{-1}(i)Id(z)Id^{-1}(x)Id^{-1}(y)E(x)E(y).$$

Для проверки правильности переходов от команды к следующей команде вводится новая переменная H, описывающая историю выполнения команд. Начальное значение H равно $f(i_0)f(i_1)$, где i_0 и i_1 — случайно выбранные секретные числа, i_1 — идентификационный номер первой выполненной команды, i_0 — стартовое число, приписываемое началу программы.

Если после выполнения j-й команды осуществляется переход к i-й команде, то выполняется присваивание $H \leftarrow Id^{-1}(j)Id(i)H$. Такая команда присваивания вставляется непосредственно после j-й команды исходной программы. Отсюда следует, что текущее значение H в момент выполнения i-й команды равно $f(i_0)Id(i) = f(i_0)f(i)$. Таким образом, если в исходной программе i-я команда имеет вид $z \leftarrow x + y$, то в модифицированной программе такому фрагменту соответствует блок

$$\begin{aligned} &H \leftarrow Id^{-1}(j)Id(i)H;\\ &begin\ E(z) \leftarrow Hf^{-1}(i_0)Id^{-1}(i)Id(z)Id^{-1}(x)Id^{-1}(y)E(x)E(y);\\ &z \leftarrow x + y\\ &end. \end{aligned} \tag{4}$$

Здесь j — номер команды, предшествующей команде с номером i.

Нетрудно проверить, что в (4) выражение в правой части оператора присваивания $E(z)$ равно $Id(z)f(x)f(y) = E(z)$.

Значения констант

$$c_i = f^{-1}(i_0)Id^{-1}(i)Id(z)Id^{-1}(x)Id^{-1}(y),\ c_{ji} = Id^{-1}(j)Id(i)$$

подсчитываются заранее после выполнения разметки. Заметим, что коэффициент c_i не зависит от предыдущего состояния истории. Из состояния j в зависимости от условий можно перейти в разные состояния. Эти состояния однозначно определяются номером j и соответствующими условиями.

Для составных операторов история меняется по уровням. Если фрагмент размеченной программы имеет вид $i : A$, где A — составной оператор и последняя команда A имеет номер k_A, то после выполнения A история возвращается в состояние, соответствующее номеру i. Для этого в конец A добавляется команда $H \leftarrow Id^{-1}(k_A)Id(i)H$. Так, например, если условный оператор ветвления с размеченными составными операторами A и B имеет вид

$$i : if < \text{условие} > then\ \ A\ \ else\ \ B;$$

и последняя команда A имеет номер j_A, а в B соответствующая команда имеет номер j_B, то в A и B дополнительно вставляются команды изменения истории, возвращающие значение идентификатора входа:

$$i : if <\text{условие}>\ then$$

$$begin\ \ A;\ H \leftarrow Id^{-1}(j_A)Id(i)H\ \ end$$

$$else$$

$$begin\ \ B;\ H \leftarrow Id^{-1}(j_B)Id(i)H\ end.$$

Размечаются только команды исходной программы; операторы изменения истории и контролирующих цифровых подписей не размечаются.

После выполнения всех необходимых вставок контролирующих переменных и соответствующих команд присваивания введенная разметка уничтожается.

Для контроля правильности выполнения условных переходов вводятся дополнительные переменные, в которых запоминаются значения переменных после выполнения входов/выходов в условие. Запоминаются значения только последних входов/выходов в условие. Предполагается, что если было вставлено новое несоответствующее условие, то исходное условие хотя бы при одной из последних итераций будет искажено. Это обнаружится на этапе проверки после получения результатов выполнения программы.

ВЫПОЛНЕНИЕ РАЗМЕТКИ

Разметка выполняется рекурсивно. Пусть $\mathcal{L}$ обозначает процесс разметки.

Предположим, что A — оператор присваивания, например, $A = z \leftarrow x + y$. Тогда $\mathcal{L}(A) = i : z \leftarrow x + y$, где i — случайно выбранное целое положительное число из заранее обусловленного интервала I.

Разметка блочного оператора *begin A end*:

$$\mathcal{L}\ (begin\ \ A\ \ end) = i{:}\ begin\ \ \mathcal{L}(A)\ \ end.$$

Разметка последовательности операторов:

$$\mathcal{L}(A; B) = \mathcal{L}(A);\ \ \mathcal{L}(B).$$

Разметка условного оператора:

$$\mathcal{L}\ (if\ x \geq y\ \ then\ \ A\ \ else\ \ B) = i : if\ x \geq y\ then\ \mathcal{L}(A)\ else\ \mathcal{L}(B).$$

Разметка цикла:

$$\mathcal{L}\ (while\ \ x \geq y\ do\ A) = i{:}\ while\ \ x \geq y\ \ do\ \ \mathcal{L}\,(A).$$

Во всех случаях метка i — случайно выбранное целое положительное число из интервала I. Предполагаем, что все метки — разные числа, хотя это условие не является обязательным.

ВЫЧИСЛЕНИЕ ИСТОРИИ ПЕРЕХОДОВ

Выполнение программы является детерминированным процессом. Это означает, что после выполнения команды с номером j в зависимости от условий перехода однозначно вычисляется место передачи управления — команда с номером i. Отсюда следует, что по тексту программы можно вычислить коэффициенты $c_{ji} = Id^{-1}(j)Id(i)$. Таким образом, после команды с номером j необходимо вставить условие согласованности перехода $H \leftarrow c_{ji}H$.

Для составных операторов, имеющих вложенную структуру, выполнение последней программы нижнего уровня приводит к возвращению управления на соответствующую команду верхнего уровня, из которой был осуществлен переход на нижний уровень. Поэтому для вложенных команд после их выполнения необходимо добавлять изменения истории, возвращающие номер на вызывающий оператор верхнего уровня.

Рассмотрим некоторые примеры.

Предположим, имеем размеченную условную команду

$$i_1\colon \mathit{if}\ \ x \geq y\ \ \mathit{then}\ \ \ i_2 : A\ \ \ \mathit{else}\ \ \ i_3 : B;\ i_4 : C;$$

Полагаем, что A и B — составные операторы.

После выполнения команды с номером i_1 начинается выполнение команды с номером i_2 или i_3. Поэтому перед командой A вставляется оператор изменения истории, соответствующий переходу к команде i_2, $H \leftarrow c_{i_1 i_2}H$. После тела оператора A вводится оператор перехода истории в состояние i_1. Аналогичные вставки выполняются перед и после оператора B. Перед оператором C вставляется соответствующий переход к номеру i_4, $H \leftarrow c_{i_1 i_4}H$. Операторы A и B являются подчиненными команде с номером i_1. Поэтому необходимо вставить условия согласования возвращения к номерам i_2 и i_3 соответственно и последующего возвращения к номеру i_1. Данный фрагмент видоизменяется следующим образом:

$$H \leftarrow c_{ji_1}H;$$
$$i_1\colon \mathit{if}\ \ x \geq y\ \ \mathit{then}$$
$$\mathit{begin}\ \ H \leftarrow c_{i_1 i_2}H;\ i_2 : A;\ H \leftarrow c_{j_A i_2}H;\ H \leftarrow c_{i_2 i_1}H\ \ \mathit{end}$$
$$\mathit{else}$$
$$\mathit{begin}\ \ H \leftarrow c_{i_1 i_3}H;\ i_3 : B;\ H \leftarrow c_{j_B i_3}H;\ \ H \leftarrow c_{i_3 i_1}H\ \ \mathit{end}$$
$$H \leftarrow c_{i_1 i_4}H;\ i_4 : C;$$

Здесь j_A и j_B — номера последних операторов блоков A и B соответственно. Предполагается, что A и B — составные операторы.

В случае, если A или B составляют базовый оператор присваивания, нет необходимости вставлять переходы истории в начало A или B.

Предположим, фрагмент размеченной программы имеет такой вид:

$$i_1\colon \mathit{while}\ \ x \geq y\ \ \mathit{do}\ \ i_2\colon \mathit{begin}\ \ i_3 : z \leftarrow x + y;\ \ i_4 : y \leftarrow z\ \ \mathit{end};\ \ i_5 : A;$$

Вставки H и E выполняются следующим образом:

$$H \leftarrow c_{ji_1}H;\ \ i_1 : \mathit{while}\ x \geq y\ \mathit{do}$$
$$\mathit{begin}$$
$$H \leftarrow c_{i_1 i_2}H$$
$$\mathit{begin}$$
$$H \leftarrow c_{i_2 i_3}H;\ E(z);\ i_3 : z \leftarrow x + y;\ \ H \leftarrow c_{i_3 i_4}H;$$
$$E(y);\ \ i_4 : y \leftarrow z\,;\ H \leftarrow c_{i_4 i_2}H$$
$$\mathit{end}$$
$$H \leftarrow c_{i_2 i_1}H$$
$$\mathit{end}$$
$$H \leftarrow c_{i_1 i_5}H;\ i_5 : A.$$

Для упрощения в приведенном фрагменте запись $E(z)$ обозначает изменение цифровой подписи $E(z)$ для оператора $z \leftarrow x + y$ согласно формуле (4); аналогичное обозначение вводится и для $E(y)$. Подпись $E(z)$ должна вычисляться непосредственно перед выполнением команды $z \leftarrow x + y$, так как используются текущие значения x и y, которые в случае операторов вида $x \leftarrow x + y$ могут изменяться после выполнения присваивания.

КОНТРОЛЬ УСЛОВНЫХ ВЕТВЛЕНИЙ

Для складывающей машины все условия имеют вид неравенств или равенств типа $x \le y$, $x < y$, $x = y$.

Обозначим $[\alpha]$ некоторую последовательность операторов, которые предназначены для проверки выполнения текущих значений условия α. Выполнение $[\alpha]$ не влияет на результаты выполнения исходной программы. Возможны разные варианты реализации блока $[\alpha]$. Один конкретный вариант описан ниже.

Условный оператор *if* α *then A else B* преобразуется следующим образом:

$$\begin{aligned} &\textit{if } \alpha \quad \textit{then begin } [\alpha];\ A \ \textit{ end} \\ &\qquad \textit{else begin } [\neg\alpha];\ B \ \textit{end}. \end{aligned}$$

В блоке *begin* $[\alpha]$; *A end* фиксируется информация о входе в часть *A*. Аналогично интерпретируется соответствующая часть ветвления, ведущая в *B*.

Односторонний условный оператор, *if* α *then A*, предварительно переписывается в условный оператор с двумя возможными выходами:

$$\textit{if } \alpha \ \textit{ then } \ A \ \textit{ else begin } x \leftarrow x;\ y \leftarrow y \ \textit{end}.$$

Здесь x и y — переменные, используемые в формировании условия α складывающей машины. Соответствующее преобразование имеет следующий вид:

$$\begin{aligned} &\textit{if } \alpha \quad \textit{then begin } [\alpha];\ A \ \textit{ end} \\ &\qquad \textit{else begin } \ [\neg\alpha];\ x \leftarrow x;\ y \leftarrow y \ \textit{end}. \end{aligned}$$

Таким образом, если, например, условие α имеет вид $x \ge y$, а злоумышленник изменит его на какое-то другое, например $u < v$, где u и v — некоторые переменные, и при входе в условный оператор выполняется переход, противоречащий необходимому входу в *A*, то этот факт будет зафиксирован при проверке $[\neg\alpha]$ во второй части условного оператора.

Рассмотрим оператор цикла *while* α *do A*.

Данный оператор преобразуется следующим образом:

$$\begin{aligned} &\textit{if } \alpha \ \textit{then} \\ &\qquad \textit{begin } [\alpha];\ \textit{while } \alpha \ \textit{do} \\ &\qquad\qquad \textit{begin } [\alpha];\ A;\ [\neg\alpha] \ \textit{end} \\ &\qquad \textit{end} \\ &\textit{else} \\ &\qquad \textit{begin } [\neg\alpha];\ x \leftarrow x;\ y \leftarrow y \ \textit{end}. \end{aligned}$$

Здесь x и y — переменные, используемые в формировании условия α.

Для конкретной реализации проверки правильности выполнения условных ветвлений, задающихся неравенствами типа $x \ge y$, вводятся дополнительные наборы переменных, в которых запоминаются последние k текущих значений управляющих переменных при вхождении в соответствующий блок. Выбор параметра k зависит от требований к степени доверия к удаленным вычислениям. Чем больше k, тем больше вероятность получения достоверной информации о правильности выполнения условных переходов. В предельном случае для гарантирования 100% достоверной информации возможна запись в файл всех значений переменных, определяющих условные переходы, вместе с их цифровыми подписями.

Предположим, переменная x входит в блок *A*. Последние k значений, которые принимала переменная x, запоминаются с помощью k циклических

присваиваний

$$x_k \leftarrow x_{k-1};\ x_{k-1} \leftarrow x_{k-2}; \ldots;\ x_1 \leftarrow x.$$

Здесь $x_1, x_2 \ldots x_k$ — новые переменные, уникальные для каждого блока и переменных, определяющих условия ветвления. Эти переменные предназначены только для запоминания последних k значений соответствующей переменной x. Они никак не влияют на процесс вычислений. Поэтому для них нет необходимости вводить нумерацию и вносить в историю H. При переходе от x_{j-1} к x_j, $x_j \leftarrow x_{j-1}$, $2 < j \le k$, цифровая подпись меняется по общему правилу

$$E(x_j) = Id(x_j) Id^{-1}(x_{j-1}) E(x_{j-1}).$$

Для x_1 цифровая подпись вычисляется по формуле

$$E(x_1) \leftarrow Id(x_1) Id^{-1}(x) E(x).$$

Рассмотрим примеры. Для упрощения полагаем $k = 2$. Предполагаем, что в программе уже выполнены разметка и вставки соответствующих цифровых подписей.

Условный оператор имеет вид

$$i_1:\ if\ x \ge y\ then\ i_1 : A\ else\ i_2 : B.$$

Для блока A резервируем четыре новые переменные: $x^+_{i1}, x^+_{i2}, y^+_{i1}, y^+_{i2}$. Соответственно для блока B вводим четыре переменные: $x^-_{i1},\ x^-_{i2},\ y^-_{i1},\ y^-_{i2}$.

Условный оператор преобразуется следующим образом:

$i_1 : if\ x \ge y\ then$
begin
$E(x^+_{i2}); x^+_{i2} \leftarrow x^+_{i1}; E(x^+_{i1}); x^+_{i1} \leftarrow x;$
$E(y^+_{i2}); y^+_{i2} \leftarrow y^+_{i1};\ E(y^+_{i1}); y^+_{i1} \leftarrow y;$
$i_1 : A$
end
else
begin
$E(x^-_{i2}); x^-_{i2} \leftarrow x^-_{i1};\ E(x^-_{i1}); x^-_{i1} \leftarrow x;$
$E(y^-_{i2}); y^-_{i2} \leftarrow y^-_{i1};\ E(y^-_{i1}); y^-_{i1} \leftarrow y;$
$i_2 : B$
end.

Изменения цифровых подписей происходит по общим правилам. Например, запись $E(x^+_{i2})$ на самом деле обозначает

$$E(x^+_{i2}) \leftarrow Id(x^+_{i2}) Id^{-1}(x^+_{i1}) E(x^+_{i1}).$$

После получения результатов программы выполняется проверка соответствий

$$x^+_{i1},\ x^+_{i2},\ y^+_{i1},\ y^+_{i2},\ \ x^-_{i1},\ x^-_{i2},\ y^-_{i1},\ y^-_{i2}$$

своим цифровым подписям. Затем вычисляются и проверяются разности

$$x^+_{i1} - y^+_{i1} \ge 0,\ x^+_{i2} - y^+_{i2} \ge 0,\ y^-_{i1} - x^-_{i1} > 0,\ y^-_{i2} - x^-_{i2} > 0.$$

Заметим, что количество вхождений в условный блок может быть меньше выбранного параметра запоминания k последних вхождений. Поэтому первые $x_k, x_{k-1}, \ldots, x_i$ значений из набора $x_k, x_{k-1}, \ldots, x_1, x$, $i < k$, могут задавать только начальные значения этих контролирующих переменных и по ним нельзя судить о правильности/неправильности условия. Данный случай легко проверяется сравнением сохраненных начальных значений $x^0_k, x^0_{k-1}, \ldots, x^0_1$ запоминающих переменных.

Более экономным вариантом будет фиксирование для начальных значений уникального параметра, равного разности соседних значений

$$s = x_k^0 - x_{k-1}^0 = x_{k-1}^0 - x_{k-2}^0 = \ldots = x_2^0 - x_1^0.$$

Константа s может быть, например, 20-м знаком десятичного представления числа π или e. Однако маловероятно совпадение с таким числом всех соответствующих разностей значений использованных переменных условий.

Конструкция $[\alpha]$; A; $[\neg\alpha]$ для циклического оператора

if α *then* *begin* $[\alpha]$; *while* α *do*
begin $[\alpha]$; A; $[\neg\alpha]$ *end*
else *begin* $[\neg\alpha]$; $x \leftarrow x$; $y \leftarrow y$ *end*

означает, что в $[\alpha]$ запоминаются k последних значений условий входа в блок A, а в $[\neg\alpha]$ — последние k значений выхода из A.

Если последние k значений входа в A переменной x были $x_k, x_{k-1}, \ldots, x_1$, то в $[\neg\alpha]$ соответствующими значениями будут $x'_k = x_{k-1}$, $x'_{k-1} = x_{k-2}, \ldots, x'_2 = x_1$, $x'_1 = x$. Поэтому для реализации проверок достаточно запомнить только k значений входов в A $x_k, x_{k-1}, \ldots, x_1$ и значение последнего выхода из A.

Отметим, что предлагаемое решение проблемы целостности условных операторов является эвристическим. По последним k текущим значениям переменной, входящей в условный оператор, делаем вывод о правильности/неправильности всех предыдущих значений. Как показывают эксперименты, в большинстве случаев даже при малых $k = 1, 2, 3$ неправильные вычисления обнаруживаются на стадии заключительной проверки.

КОНТРОЛЬ ОТБРАСЫВАНИЯ ОПЕРАТОРОВ

Введение истории переходов обеспечивает выполнение команд программы в линейном порядке, индуцированном порядком изменения истории. Взаимодействие истории переходов с функцией цифровой подписи защищает от изменения содержания команд присваивания. Но при этом возможен вариант отбрасывания команды вместе с цифровой подписью и даже блоков команд при сохранении порядка следования истории и правильности постановки цифровой подписи в оставшемся теле программы. Например, возможен такой сценарий. В модифицированной программе уничтожается фрагмент $E(z)$; $z \leftarrow x + y$; все остальное, включая историю, не изменяется. Поэтому при выполнении такой урезанной программы все изменения результирующих переменных в операторах присваивания будут находиться в согласованном отношении с их цифровыми подписями и историей выполнения. Для того чтобы сделать невозможными подобные действия, необходимо получить проверяющую информацию о существовании соответствующего оператора. Предполагая, что каждый составной блок содержит команды присваивания и выполняется хотя бы один раз, достаточно получить подтверждение выполнения в нужном месте только для всех операторов присваивания.

С этой целью с каждым оператором присваивания с идентифицирующим номером i связываем две контрольные переменные — ζ_{i1} и ζ_{i2}, в которых запоминается текущая соответствующая информация о структуре оператора, истории в момент i и его реальной и необходимой цифровых подписях. Согласование этой информации проверяется на заключительном этапе проверки целостности программы.

Рассмотрим процесс введения контрольных переменных ζ_{i1} и ζ_{i2} более детально на примере команды присваивания i: $z \leftarrow x + y$. После ее выполнения цифровая подпись для z должна быть $Id(z)\, Id^{-1}(x)\, Id^{-1}(y)\, E(x)\, E(y)$, где x и y — текущие значения, принимаемые непосредственно перед выполнением команды.

История H в момент выполнения этой команды равна $f(i_0) f(i)$. Поэтому, как нетрудно проверить, при выполнении команды i: $z \leftarrow x + y$ всегда должно выполняться равенство

$$V(H\ Id(x)\, Id(y)\, E(z)) = V(f(i_0) f(i) Id(z)\, E(x)\, E(y)). \qquad (5)$$

Обозначим $c_{xy} = Id(x)\, Id(y) \bmod n$ и $c_{iz} = f(i_0) f(i) Id(z) \bmod n$.

Равенство (5) перепишем в виде

$$V(c_{xy} HE(z)) = V(c_{iz} E(x)\, E(y)). \qquad (6)$$

В формуле (6) $E(z)$ — текущее реальное значение цифровой подписи для z в момент, в который должна выполняться команда i: $z \leftarrow x + y$, $E(x)$ и $E(y)$ — соответствующие текущие значения цифровых подписей для x и y в этот момент.

Информацию, равную левой части равенства (6), запоминаем в ζ_{i1}, $\zeta_{i1} \leftarrow c_{xy} HE(z)$, а правой части — в ζ_{i2}, $\zeta_{i2} \leftarrow c_{iz} E(x)\, E(y)$. Команда присваивания $\zeta_{i2} \leftarrow c_{iz} E(x)\, E(y)$ вставляется непосредственно перед оператором i: $z \leftarrow x + y$, а команда $\zeta_{i1} \leftarrow c_{xy} HE(z)$ занимает место после этого оператора присваивания. Для ζ_{i1} и ζ_{i2} также вычисляем и запоминаем в соответствующих переменных цифровые подписи $E(\zeta_{i1})$ и $E(\zeta_{i2})$. Полагаем, что ζ_{i1} и ζ_{i2} имеют одинаковые идентификаторы имен $Id(\zeta_{i1}) = Id(\zeta_{i2})$.

Введение подписей $E(\zeta_{i1})$ и $E(\zeta_{i2})$ делает невозможным несанкционированные изменения значений ζ_{i1} и ζ_{i2}.

При проверке переменные ζ_{i1} и ζ_{i2} принимают последние значения соответствующих параметров, связанных с выполнением оператора $i: z \leftarrow x + y$. Если этот оператор был изъят, то нарушится и последнее необходимое соотношение (6).

После выполнения программы проверяется соответствие ζ_{i1} и ζ_{i2} их цифровым подписям, а также равенство их значений. Напоминаем, что все вычисления, касающиеся ζ_{i1} и ζ_{i2}, выполняются по $\bmod n$.

ЗАКЛЮЧЕНИЕ

Предлагается решение проверки целостности удаленных критически важных вычислений с помощью замены основных арифметических операций процедурами складывающей машины и использованием динамической цифровой подписи, выполняемой гомоморфной по сложению криптографической функцией. Процедуры вставки цифровых подписей могут выполняться автоматически препроцессором, обрабатывающим исходную программу. Проверка корректности вычислений осуществляется простой процедурой после получения результатов выполнения программы.

Область применения предлагаемого решения проблемы целостности удаленных вычислений включает финансовые вычисления, цифровой контроль объектов критической инфраструктуры, электронное правительство, контроль криптографических преобразований и многие другие процедуры, выполняемые в сетях ЭВМ, требующие самоаутентификации и повышенной надежности.

СПИСОК ЛИТЕРАТУРЫ

1. Damgard I., Geisler M., Kroigard M. Homomorphic encryption and secure comparison. *International Journal of Applied Cryptography*. 2008. Vol. 1, Iss. 1. P. 22–31.
2. Gentry C. Fully homomorphic encryption using ideal lattices. *Symposium on Theory of Computing*. 2009. Vol. 9. P. 169–178.
3. Gentry C. A fully homomorphic encryption scheme. Doctoral dissertation. Stanford University. 2009. 199 p.
4. Van Dijk M., Gentry C., Halevi S., Vaikuntanathan V. Fully homomorphic encryption over the integers. *Annual Intern. Conf. on the Theory and Applications of Cryptographic Techniques*. Berlin; Heidelberg: Springer, 2010. P. 24–43.
5. Brakerski Z. Fully homomorphic encryption without modulus switching from classical GapSVP. Advances in Cryptology–CRYPTO 2012. Berlin; Heidelberg: Springer, 2012. P. 868–886.
6. Lopez-Alt A., Tromer E., Vaikuntanathan V. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. *Proc. of the forty-fourth annual ACM symp. Theory of computing*. ACM, 2012. P. 1219–1234.
7. Gentry C., Sahai A., Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. *Advances in Cryptology–CRYPTO* 2013. Berlin; Heidelberg: Springer, 2013. P. 75–92.

8. Ryan M.D. Cloud computing security: The scientific challenge, and a survey of solutions. J*ournal of Systems and Software*. 2013. Vol. 86, Iss. 9. C. 2263–2268.
9. Brakerski Z., Vaikuntanathan V. Efficient fully homomorphic encryption from (standard) LWE. *SIAM Journal on Computing*. 2014. Vol. 43, Iss. 2. P. 831–871.
10. Brakerski Z., Gentry C., Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. on Computation Theory*. 2014. Vol. 6, Iss. 3. Article N 13. 36 p.
11. Rivest R.L., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. 1978. Vol. 21, Iss. 2. P. 120–126.
12. Bellare M., Rogaway P. Optimal asymmetric encryption. *Advances in Cryptology–EUROCRYPT'94*. Berlin; Heidelberg: Springer, 1995. P. 92–111.
13. ElGamal T. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. on Information Theory*. 1985. Vol. 31, Iss. 4. P. 469–472.
14. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. *Intern. Conf. on the Theory and Applications of Cryptographic Techniques*. Berlin; Heidelberg: Springer, 1999. P. 223–238.
15. Benaloh J. Dense probabilistic encryption. *Proc. of the workshop on selected areas of cryptography*. Kingston, 1994. P. 120–128.
16. Floyd R.W., Knuth D.E. Addition machines. *SIAM Journal on Computing*. 1990. Vol. 19, Iss. 2. P. 329–340.

Надійшла до редакції 03.04.2017

А.В. Анісімов, А.К. Новокшонов

ДОВІРЧІ ОБЧИСЛЕННЯ З ВИКОРИСТАННЯМ ДОДАВАЛЬНОЇ МАШИНИ. I

Анотація. Запропоновано розв'язання проблеми цілісності арифметичних програм, які виконуються на віддаленому обчислювальному ресурсі. Підхід до розв'язання полягає у заміні арифметичних операцій, таких як множення і ділення, процедурами додавальної машини (addition machine), введеної Р. Флойдом і Д. Кнутом. Обчислення і послідовність команд підписуються динамічним цифровим підписом, що є гомоморфним за додаванням/відніманням. Верифікація цифрових підписів гарантує виявлення будь-яких несанкціонованих змін у вихідному тексті програми і результатах обчислень.

Ключові слова: додавальна машина, цифровий підпис, гомоморфна криптографія.

A.V. Anisimov, A.K. Novokshonov

TRUSTED COMPUTING WITH ADDITION MACHINES. I

Abstract. A solution of the integrity problem for arithmetic programs running on a remote computing resource is proposed. The solution is to replace the arithmetic operations such as multiplication and division by procedures of the addition machine introduced by R. Floyd and D. Knuth. The order of instructions as well as current meanings of variables are signed by dynamic digital signatures, which are homomorphic with respect to addition and subtraction. Verification of digital signatures ensures detection of any unauthorized changes to the source code of the program and to the results of calculations.

Keywords: addition machine, digital signature, homomorphic cryptography.

Анисимов Анатолий Васильевич,
чл.-кор. НАН Украины, доктор физ.-мат. наук, профессор, декан Киевского национального университета имени Тараса Шевченко, e-mail: ava@unicyb.kiev.ua.

Новокшонов Андрей Константинович,
аспирант Киевского национального университета имени Тараса Шевченко,
e-mail: andrey.novokshonov@ukr.net.