КІБЕРНЕТИКА

А.В. АНИСИМОВ, А.К. НОВОКШОНОВ

УДК 519.72, 003.26

ДОВЕРИТЕЛЬНЫЕ ВЫЧИСЛЕНИЯ С ИСПОЛЬЗОВАНИЕМ СКЛАДЫВАЮЩЕЙ МАШИНЫ. II [1]

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

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

ВВЕДЕНИЕ

Повсеместное распространение и использование облачных технологий привело к появлению большого количества сервисов, которые позволяют делегировать данные и вычисления на удаленные компьютерные ресурсы. Однако поскольку такие ресурсы не являются доверенными с точки зрения пользователя (например, по причине возможных сбоев программного и аппаратного обеспечения, заражения вирусами или преднамеренных искажений), то возникает необходимость в методах, которые бы обеспечили пользователю возможность проверки корректности делегированных вычислений. При этом если проверка корректности делегированных вычислений менее трудоемка, чем сами вычисления, то подобные методы проверки корректности можно использовать в сценариях с маломощными устройствами (такими, как мобильные устройства и интернет вещей), которые не в состоянии провести сложных вычислений локально, но могут эффективно выполнить проверку корректности результатов таких вычислений.

Формальное определение проверяемых вычислений (verifiable computing) впервые дали Р. Дженнаро, К. Джентри и Б. Парно в работе [1], что послужило отправной точкой для интенсивных исследований в этой области, одновременно стимулируемых развитием и массовым внедрением облачных технологий такими компаниями, как Google, Amazon, Microsoft и другими.

Учитывая недостаточную для использования на практике эффективность существующих методов гомоморфной криптографии, на современном этапе проверяемые вычисления представляются более перспективным и востребованным направлением для исследований.

Основные подходы в области проверяемых вычислений [2]:

- вероятностно-проверяемые доказательства [3–7];
- неинтерактивные проверяемые вычисления [1, 8–11];
- публично проверяемые вычисления [12–16];
- гомоморфные аутентификаторы [17–24].

[1] Начало см. в № 5, 2017.

© А.В. Анисимов, А.К. Новокшонов, 2018

Несмотря на активные исследования в области проверяемых вычислений в последние годы, большинство методов проверяют корректность только одной функции и не учитывают полноценных программ с операторами ветвления и циклов. Из известных нам подходов только два [25, 26] ориентированы на проверку целостности программ, написанных на подмножестве языка C. В то же время подход, предлагаемый в настоящей статье, ориентирован на проверку целостности полноценных программ, представленных в виде процедур складывающей машины с использованием условных ветвлений и циклов.

Настоящая статья является продолжением работы [27] и использует принятые там обозначения.

ОБЩЕЕ ОПИСАНИЕ МЕТОДА

Рассмотрим проблему целостности удаленных вычислений, т.е. когда процесс удаленного выполнения программы пользователя соответствует ее исходному заданию. Сами данные не шифруются. В рассматриваемой модели вычислений все переменные и константы — целые числа. Разрешены условные конструкции типа if-then-else и циклы типа while-do. Разрешенные операции — только сложения и вычитания на ограниченном числе регистров, которые принимают целочисленные значения. При этом не представляет трудности перенос вычислений на рациональные числа, используя арифметику чисел с плавающей запятой. Такая модель вычислений была введена Р. Флойдом и Д. Кнутом [28] и получила название «складывающая машина» (addition machine). Детальное описание складывающей машины, а также анализ производительности ее процедур даны в Приложении 1.

Для складывающей машины мы разработали вариант динамической цифровой подписи вычислений, которая изменяется в соответствии с вычислительным процессом. Любые несанкционированные модификации результатов или текста исходной программы (изменение значений переменных или констант, нарушение условий и порядка выполнения команд, внесение новых переменных или команд) приведут к нарушению верификационных условий цифровой подписи. Для получения зашифрованных данных цифровой подписи выбрана схема Бенало [29]. Данные исходной базовой программы не шифруются.

Основная 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 к этому мы используем динамическую цифровую подпись, осуществляемую гомоморфной по сложению криптографической функцией. Действия над цифровыми подписями выполняются в обычной модулярной арифметике вычислительной машины. Цифровая подпись переменной должна учитывать три идентифицирующих показателя: текущее значение переменной, имя переменной и место в программе, где эта переменная изменяла свое значение.

Более детальное описание предлагаемого метода приведено в [27].

ВЫБОР ФУНКЦИИ ЦИФРОВОЙ ПОДПИСИ

Существует достаточно большое количество криптографически стойких односторонних функций f, обладающих свойством гомоморфности по сложению типа $f(x+y)=f(x)f(y)$. К таковым относятся схема Эль-Гамаля [30], схема Пэйе [31], схема Бенало [29] и ряд других.

Схема Эль-Гамаля требует дополнительных затрат, связанных с реализацией выборов случайных вычислительных параметров. В схеме Пэйе $f(x)=$ $=(1+Rn)^x b^n \bmod n^2$, где R — секретный параметр, b — произвольное число, $b \in Z_n^*$, возможно выполнение несанкционированной команды типа $x \leftarrow x+n$, которая изменяет значения переменной x без изменения цифровой подписи, т.е. $f(x+n)=f(x)$. В схеме Бенало такие действия отсутствуют, поэтому мы ее выбираем. Схема Бенало была предложена в [29] как схема шифрования с публичными ключами.

Оригинальная схема Бенало (Benaloh). Генерация ключей:

1) выбрать два больших простых числа: p и q; r — делитель числа $p-1$ такой, что

$$\text{НОД}(r,(p-1)/r)=1,\ \text{НОД}(r,q-1)=1,$$

$$n=pq,\ \varphi(n)=(p-1)(q-1),\ c=\varphi(n)/r;$$

2) выбрать $g\in Z_n^*$, $g^{\varphi(n)/r}\neq 1 \bmod n$.

Кодирование: $E_r(m)=g^m b^r \bmod n$, где $m\in Z_r$, b — произвольный вычет из Z_n^*; очевидно, что $E_r(m_1+m_2)=E_r(m_1)E_r(m_2)$.

Здесь Z_n^* обозначает мультипликативную группу вычетов по модулю n. Отметим, что (n, g, r) — открытые ключи; p и q — секретные ключи.

Для цифровой подписи переменных используем модификацию схемы Бенало.

Модифицированная схема Бенало для цифровой подписи складывающей машины. Ключи p, q, g, r, c, a — секретные, известные только пользователю, при этом $\text{НОД}(a,\varphi(n))=1$, $c=\varphi(n)/r$. Открытый ключ $n=pq$.

Для целого положительного числа x полагаем $f(x)=g^{ax}b^r \bmod n$, $f(-x)=g^{-ax}b^r \bmod n$, где b — случайно выбранное число из Z_n^*.

Каждой переменной x пользователь сопоставляет случайно выбранное фиксированное число r_x, $0<r_x<\varphi(n)$. Все числа r_x — секретные.

Полагаем $Id(x)=g^{r_x}u^r \bmod n$, $u\in Z_n^*$. Выбор u случаен.

Цифровая подпись переменной x задается формулой

$$E(x)=Id(x)f(x)=g^{ax+r_x}(ub)^r \bmod n=g^{(ax+r_x)\bmod r}(ubg^k)^r \bmod n, \qquad (1)$$

где $k=\lfloor (r_x+ax)/r \rfloor$.

Верификационное условие для переменной x, которое проверяется пользователем после получения результатов вычисления программы, следующее:

$$V(E(x)):\ E^c(x) \bmod n=(g^{acx+cr_x}) \bmod n=(g^{ac} \bmod n)^x \cdot g^{cr_x} \bmod n.$$

Заметим, что декодирование в схеме Бенало с публичными ключами достаточно сложное. В режиме цифровой подписи эта операция заменяется проверкой простых условий, связывающих значения и имена переменных.

Отметим, что если начинать генерацию схемы Бенало с выбора простых чисел p и q, то возникает проблема поиска делителя r числа $p-1$. Поэтому целесообразно начинать построение схемы Бенало с фиксации простого числа r. Затем p находим как первое простое число в арифметической прогрессии $lr+1=p$, где $l=k, k+1,\ldots$

Возможны модификации схемы Бенало в части выбора параметров r и соответственно c.

ПРИМЕР ПРОВЕРКИ ЦЕЛОСТНОСТИ ВЫЧИСЛЕНИЙ

На основании предложенного в настоящей статье метода был разработан программный прототип системы проверки целостности вычислений с использованием языка программирования Python. Суть его состоит в том, что пользователь подает на вход системы исходный текст программы на языке складывающей машины. Далее система автоматически выполняет вставку контрольных переменных, вычисляет начальные значения цифровых подписей для переменных, определяет значения констант изменения истории вычислений, добавляет их в программу и выводит пользователю новый исходный текст модифицированной программы на языке Python, пригодный для выполнения.

Затем пользователь запускает модифицированную программу на удаленном вычислительном ресурсе и по результатам выполнения для каждой переменной

получает кортеж вида <имя переменной, значение переменной, значение цифровой подписи переменной>.

Полученные кортежи пользователь подает на вход системы, после чего система выполняет проверку верификационных условий и выдает результат об обнаружении (или не обнаружении) нарушения целостности вычислений. В случае обнаружения факта нарушения целостности выдается информация о том, какие именно верификационные условия не прошли проверку.

Генерация криптографических параметров для цифровых подписей проводится системой автоматически с использованием библиотеки PyCrypto, исходя из нужного размера секретных ключей, который задается пользователем. Параметр k, определяющий количество запоминаемых последних значений управляющих переменных цикла, также задается пользователем.

Рассмотрим пример проверки целостности вычислений простой программы, определяющей первое число Фибоначчи, которое строго больше заданного числа x. Псевдокод такой программы на языке складывающей машины имеет следующий вид:

```
Pr0: read x;
t ← 0; y ← 1; z ← 1;
while x >= z do
    begin
    t ← z + y; y ← z; z ← t;
    end;
write z.
```

В целях тестирования системы рассмотрим следующие основные сценарии преднамеренной модификации программы и экспериментальным путем проверим возможность обнаружения системой нарушения целостности вычислений.

Сценарий 1. Искажение условия (замена условия $x >= z$ противоположным):

```
Pr1: read x;
t ← 0; y ← 1; z ← 1;
while x < z do { изменено условие }
    begin
    t ← z + y; y ← z; z ← t;
    end;
write z.
```

Сценарий 2. Дублирование оператора $(t \leftarrow z + y)$:

```
Pr2: read x;
t ← 0; y ← 1; z ← 1;
while x >= z do
    begin
    t ← z + y; { продублирован оператор }
    t ← z + y; y ← z; z ← t;
    end;
write z.
```

Сценарий 3. Отбрасывание оператора $(z \leftarrow t)$:

```
Pr3: read x;
t ← 0; y ← 1; z ← 1;
while x >= z do
    begin
    t ← z + y; y ← z; { отсутствует оператор z ← t }
    end;
write z.
```

Сценарий 4. Искажение значения существующей переменной ($z \leftarrow z + 1$):

```
Pr4: read x;
t ← 0; y ← 1; z ← 1;
while x >= z do
    begin
    t ← z + y; y ← z;  z ← t;
    end;
z ← z + 1; {добавлено искажение значения переменной z}
write z.
```

Сценарий 5. Введение новой переменной ($w \leftarrow 1$):

```
Pr5: read x;
t ← 0; y ← 1; z ← 1; w ← 1; {введена новая переменная w}
while x >= z do
    begin
    t ← z + y; y ← z;  z ← t;
    end;
write z.
```

Сценарий 6. Изменение порядка команд ($y \leftarrow z$ и $z \leftarrow t$):

```
Pr6: read x;
t ← 0; y ← 1; z ← 1;
while x >= z do
    begin
    t ← z + y; z ← t; y ← z; {изменен порядок команд y ← z и z ← t}
    end;
write z.
```

При многократных запусках системы в процессе случайно сгенерированных значений x и параметре $k = 2$ были получены результаты, представленные в табл. 1, которые свидетельствуют о корректном 100%-обнаружении несанкционированных вмешательств для всех рассмотренных сценариев. Символ + означает, что проверка целостности успешно пройдена для конкретного типа; символ – означает обнаружение несанкционированных вмешательств; столбцы Pr1–Pr6 означают описанные выше программы для сценариев преднамеренной модификации. Названия конкретных переменных, для которых проверка не прошла, специально опущены, чтобы не загромождать таблицу. Итоговый вывод о целостности программы фиксирует прохождение программой проверку целостности по совокупности всех верификационных условий. В приведенном в табл. 1 примере ни одна из программ для описанных сценариев не прошла проверку целостности, что свидетельствует об успешном выявлении несанкционированных изменений.

Таблица 1

Тип проверки	Результаты проверки целостности вычислений для сценариев					
	Pr1	Pr2	Pr3	Pr4	Pr5	Pr6
Соответствие цифровой подписи значению переменной	+	–	+	–	–	–
Разность контрольных переменных основного блока ветвления	+	+	+	+	+	+
Разность контрольных переменных альтернативного блока ветвления	–	+	+	+	+	+
Разность контрольных переменных цикла	+	+	+	+	+	+
Равенство реальной и необходимой цифровых подписей для операторов присваивания	+	+	–	+	+	+
Итоговый вывод о целостности программы	–	–	–	–	–	–

ЗАКЛЮЧЕНИЕ

В настоящей статье рассмотрено решение проблемы проверки целостности удаленных арифметических вычислений над целочисленными данными, использующих условные ветвления и циклы. Подход состоит в замене основных арифметических операций (сложение/вычитание, умножение/деление) соответствующими процедурами складывающей машины Флойда–Кнута и использовании динамической цифровой подписи, выполняемой гомоморфной по сложению криптографической функцией схемы Бенало. Разработаны верификационные условия, которые позволяют контролировать правильность выполнения условных ветвлений и операторов цикла. Также дан анализ производительности операций, моделируемых с помощью складывающей машины.

ПРИЛОЖЕНИЕ 1
ОПИСАНИЕ СКЛАДЫВАЮЩЕЙ МАШИНЫ

В оригинальной статье Р. Флойда и Д. Кнута [28] была введена математическая модель, названная складывающей машиной. Суть ее состоит в том, что с помощью всего лишь операций сложения, вычитания, сравнения, присваивания и ограниченного количества регистров можно с приемлемой вычислительной эффективностью выразить более сложные операции, такие как умножение, нахождение остатка по модулю, нахождение наибольшего общего делителя, возведение в степень по модулю. Несмотря на небольшой набор операций, с их помощью можно выразить значительную часть арифметических алгоритмов, используемых в современных криптографических протоколах.

Формально, складывающая машина — это вычислительное устройство с конечным количеством регистров и следующим ограниченным набором операций:

- read x — ввод данных в регистр;
- $x \leftarrow y$ — копирование регистра y в регистр x;
- $x \leftarrow x + y$ — прибавление регистра y к регистру x;
- $x \leftarrow x - y$ — вычитание регистра y из регистра x;
- if $x \geq y$ — сравнение регистров x и y;
- write x — вывод данных из регистра x.

Регистры могут содержать как целые числа (используется понятие целочисленной складывающей машины), так и действительные числа (используется понятие складывающей машины с действительными числами).

Основной идеей алгоритма нахождения остатка от целочисленного деления, который был предложен Р. Флойдом и Д. Кнутом, является использование представления Фибоначчи вместо традиционного бинарного представления. Поэтому складывающую машину уместно было бы назвать машиной Фибоначчи. Известно, что любое неотрицательное число может быть представлено в виде суммы чисел Фибоначчи. Ключевым моментом является то, что с помощью складывающей машины можно легко переходить от пары чисел Фибоначчи $< F_t, F_{t+1} >$ к следующей паре $< F_{t+1}, F_{t+2} >$ с использованием всего одной операции сложения или к предыдущей паре $< F_{t-1}, F_t >$ с помощью только одной операции вычитания. Числа Фибоначчи возрастают экспоненциально и именно поэтому могут использоваться в качестве аналогов степеней двойки.

Для полноты описания далее приведем более сложные алгоритмы, которые могут быть выражены через базовые операции с помощью складывающей машины. Все программы заимствованы из работы [28]. В командах регистровой складывающей машины алгоритм P1 вычисления $x[y/z]$, который играет ключевую роль в предложенном в данной статье методе проверки целостности, имеет следующий вид:

```
P1: read x; read y; read z; { подразумевается, что y ≥ 0, z > 0 }
w ← w − w;
if   y >= z  then
    begin  u ← x;   v ← z;
    repeat  < u, x > ← < x, u + x >;  < v,  z > ←  < z, v + z >;
```

```
    until not y >= z;
    repeat if y >= v then <w, y> ← <w + u, y − v>;
        <u, x> ← <x − u, u>; <v, z> ← <z − v, v>;
    until v >= z;
    end;
  write w.
```

Умножение xy выполняется программой P1 при $z = 1$.

Аналогично алгоритм P2 нахождения остатка от целочисленного деления $x \bmod y$ в командах регистровой складывающей машины имеет вид:

```
P2: read x; read y; { подразумевается, что x ≥ 0, y > 0 }
if x >= y then
    begin z ← y;
    repeat <y, z> ← <z, y + x> until not x >= z;
    repeat if x >= y then x ← x − y, <y, z> ← <z − y, y>;
    until y >= z;
    end;
  write x.
```

Операция $< y, z > \leftarrow < z, y + x >$ означает одновременное присваивание $y \leftarrow z$ и $z \leftarrow y + x$.

Также в [28] приведены программы для эффективного вычисления НОД (x, y) и $x^y \bmod z$.

Используя арифметику чисел с плавающей запятой, нетрудно перенести операции складывающей машины на рациональные числа, тем самым приблизить модель к реальным вычислительным задачам.

ПРОИЗВОДИТЕЛЬНОСТЬ СКЛАДЫВАЮЩЕЙ МАШИНЫ

Моделью вычислений для рассмотренного в данной статье метода проверки целостности служит модель вычислений складывающей машины. В связи с этим представляют интерес вопросы оценки сложности операций складывающей машины и сравнения производительности этих операций со встроенными операциями реальных языков программирования.

Теоретические оценки сложности операций, которые могут быть реализованы с помощью складывающей машины, приведены в табл. 2. (Данные оценки были выведены Р. Флойдом и Д. Кнутом — авторами складывающей машины [28].)

Теоретически время моделирования основных арифметических операций программами складывающей машины не является лимитирующим фактором. Естественно, в процедурах складывающей машины присутствует некоторое замедление по сравнению со встроенными операциями реального языка программирования, но оно не настолько критично, чтобы нивелировать ценность использования складывающей машины и ее математических свойств.

Графики быстродействия встроенных операций C++/GMP и выраженных с помощью складывающей машины для операций умножения и нахождения остатка по модулю приведены на рис. 1. Результаты быстродействия встроенной операции умножения C++/GMP и выраженной с помощью складывающей машины даны в табл. 3, а результаты быстродействия встроенной операции нахождения остатка по модулю C++/GMP и выраженной с помощью складывающей машины приведены в табл. 4. Показателем быстродействия для приведенных результатов является реальное время выполнения программно реализованных алгоритмов на персональном ком-

Таблица 2

Операция	Время выполнения (количество команд складывающей машины)
Умножение xy	$O(\log(\min(\lvert x\rvert, \lvert y\rvert))$
Остаток $x \bmod y$	$O(\log(x / y))$
Наибольший общий делитель НОД (x, y)	$O(\log(\max(x, y) / \text{НОД}(x, y))$
Экспонента $x^y \bmod z$	$O((\log y)(\log z) + \log(x / z))$

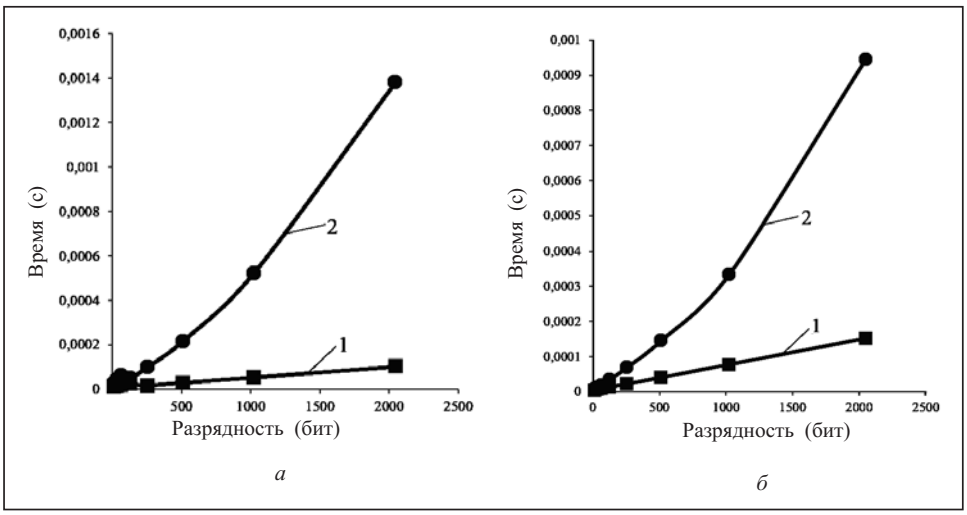

Рис. 1. Графики быстродействия встроенных операций C++/GMP (*1*) и выраженных с использованием складывающей машины (*2*) для операции умножения (*а*) и для операции нахождения остатка по модулю (*б*)

Таблица 3

Разрядность чисел	Время выполнения встроенной операции умножения (с)	Время выполнения операции умножения с помощью складывающей машины (с)
16	0,000012	0,000028
32	0,000014	0,000044
64	0,000019	0,000061
128	0,000023	0,000049
256	0,000016	0,000099
512	0,000028	0,000216
1024	0,000052	0,000522

Таблица 4

Разрядность чисел	Время выполнения встроенной операции нахождения остатка по модулю (с)	Время выполнения операции нахождения остатка по модулю с помощью складывающей машины (с)
16	0,000004	0,000008
32	0,000006	0,000012
64	0,000008	0,000018
128	0,000013	0,000035
256	0,000022	0,000069
512	0,00004	0,000145
1024	0,000077	0,000334
2048	0,000151	0,000944

пьютере с процессором Intel Pentium Duo 2,8 ГГц. Программная реализация выполнена с использованием языка программирования C++ и библиотеки длинной арифметики GMP (The GNU Multiple Precision Arithmetic Library). С более детальными результатами анализа быстродействия процедур складывающей машины можно ознакомиться в работе [32].

Таким образом, исходя из полученных экспериментальных данных можно сделать вывод о том, что быстродействие операций, выраженных с помощью складывающей машины, является от 2 до 10 раз меньшим по сравнению с быст-

родействием встроенных операций C++/GMP на данном конкретном процессоре и на входных данных указанной разрядности.

СПИСОК ЛИТЕРАТУРЫ

1. Gennaro R., Gentry C., Parno B. Non-interactive verifiable computing: outsourcing computation to untrusted workers. *Advances in Cryptology*. 2010. P. 465–482.
2. Yu X., Yan Z., Vasilakos A.V. A survey of verifiable computation. *Mobile Networks and Applications.* 2017. Vol. 22, Iss. 3. P. 438–453.
3. Goldwasser S., Kalai Y.T., Rothblum G.N. Delegating computation: Interactive proofs for muggles. *STOC'08 Proc. of the Fortieth Annual ACM Symposium on Theory of Computing.* Victoria, Canada, 2008. P. 113–122.
4. Cormode G., Mitzenmacher M., Thaler J. Practical verified computation with streaming interactive proofs. *Proc. of the 3rd Innovations in Theoretical Computer Science Conf*. 2012. P. 90–112.
5. Setty S.T., McPherson R., Blumberg A.J., Walfish M. Making argument systems for outsourced computation practical (sometimes). *19th Annual Network and Distributed System Security Symposium.* San Diego, USA, 2012.
6. Setty S.T., Vu V., Panpalia N., Braun B., Blumberg A.J., Walfish M. Taking proof-based verified computation a few steps closer to practicality. *USENIX Security Symposium.* 2012. P. 253–268.
7. Vu V., Setty S., Blumberg A.J., Walfish M. A hybrid architecture for interactive verifiable computation. *SP'13 Proc. of the 2013 IEEE Symposium on Security and Privacy*. Berkeley, USA, 2013. P. 223–237.
8. Kate A., Zaverucha G.M., Goldberg I. Constant-size commitments to polynomials and their applications. *Advances in Cryptology — ASIACRYPT 2010. Lecture Notes in Computer Science*. Berlin; Heidelberg: Springer, 2010. Vol. 6477. P. 177–194.
9. Benabbas S., Gennaro R., Vahlis Y. Verifiable delegation of computation over large datasets. *Proc. of the 31st Annual Conf. on Advances in Cryptology CRYPTO'11.* Berlin; Heidelberg: Springer, 2011. P. 111–131.
10. Fiore D., Gennaro R. Publicly verifiable delegation of large polynomials and matrix computations, with applications. *ACM Conf. on Computer and Communications Security.* 2012. P. 501–512.
11. Setty S., Braun B., Vu V., Blumberg A.J., Parno B., Walfish M. Resolving the conflict between generality and plausibility in verified computation. *Proc. of the ACM European Conf. on Computer Systems (EuroSys).* 2013. P. 71–84.
12. Canetti R., Riva B., Rothblum G.N. Two *l*-round protocols for delegation of computation. *IACR Cryptology ePrint Archive* 2011:518. 2011.
13. Papamanthou C., Tamassia R., Triandopoulos N. Optimal verification of operations on dynamic sets. *Annual Cryptology Conf.* Berlin; Heidelberg: Springer, 2011. P. 91–110.
14. Papamanthou C., Shi E., Tamassia R. Publicly verifiable delegation of computation. *IACR Cryptology ePrint Archive* 2011:587. 2011.
15. Parno B., Raykova M., Vaikuntanathan V. How to delegate and verify in public: Verifiable computation from attribute-based encryption. *Proc. of the 9th International Conf. on Theory of Cryptography (TCC'12).* Berlin; Heidelberg: Springer, 2012. P. 422–439.
16. Parno B., Howell J., Gentry C., Raykova M. Pinocchio: nearly practical verifiable computation. *SP'13 Proc. of the 2013 IEEE Symposium on Security and Privacy*. Berkeley, USA, 2013. P. 238–252.
17. Johnson R., Molnar D., Song D., Wagner D. Homomorphic signature schemes. *CT-RSA.* 2002. Vol. 2271. P. 244–262.
18. Boneh D., Freeman D.M. Homomorphic signatures for polynomial functions. *Advances in Cryptology — EUROCRYPT 2011. Lecture Notes in Computer Science.* Berlin; Heidelberg: Springer, 2011. P. 149–168.
19. Papamanthou C., Shi E., Tamassia R. Signatures of correct computation. *Proc. of the 10th Conf. on Theory of Cryptography (TCC'13).* Berlin; Heidelberg: Springer, 2013. P. 222–242.
20. Catalano D., Marcedone A., Puglisi O. Linearly homomorphic structure preserving signatures: New methodologies and applications. *IACR Cryptology ePrint Archive* 2013:801. 2013.
21. Libert B., Peters T., Joye M., Yung M. Linearly homomorphic structure preserving signatures and their applications. *Designs, Codes and Cryptography*. 2015. Vol. 77, Iss. 2–3. P. 441–477.
22. Gennaro R., Wichs D. Fully homomorphic message authenticators. International *Conf. on the Theory and Application of Cryptology and Information Security.* Berlin; Heidelberg: Springer, 2013. P. 301–320.
23. Catalano D., Fiore D. Practical homomorphic MACs for arithmetic circuits. *Annual International Conf. on the Theory and Applications of Cryptographic Techniques.* Berlin; Heidelberg: Springer, 2013. P. 336–352.

24. Lai J., Deng R.H., Pang H., Weng J. Verifiable computation on outsourced encrypted data. *Computer Security — ESORICS 2014. Lecture Notes in Computer Science.* Cham, Switzerland: Springer, 2014. Vol. 8712. P. 273–291.
25. Ben-Sasson E., Chiesa A., Genkin D., Tromer E., Virza M. SNARKs for C: Verifying program executions succinctly and in zero knowledge. *Advances in Cryptology — CRYPTO 2013.* Berlin; Heidelberg: Springer, 2013. P. 90–108.
26. Wahby R.S., Setty S.T., Ren Z., Blumberg A.J., Walfish M. Efficient RAM and control flow in verifiable outsourced computation. *22nd Annual Network and Distributed System Security Symposium.* San Diego, USA, 2015.
27. Анисимов А.В., Новокшонов А.К. Доверительные вычисления с использованием складывающей машины. I. *Кибернетика и системный анализ.* 2017. Т. 53, № 5. С. 3–13.
28. Floyd R.W., Knuth D.E. Addition machines. *SIAM Journal on Computing.* 1990. Vol. 19, Iss. 2. P. 329–340.
29. Benaloh J. Dense probabilistic encryption. *Proceedings of the Workshop on Selected Areas of Cryptography.* Kingston, Canada, 1994. P. 120–128.
30. 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.
31. Paillier P. Public-key cryptosystems based on composite degree residuosity classes. *International Conf. on the Theory and Applications of Cryptographic Techniques.* Berlin; Heidelberg: Springer, 1999. P. 223–238.
32. Новокшонов А.К. Аналіз ефективності реалізації арифметичних алгоритмів на мовах програмування C++ та Python. *Проблеми програмування.* 2016. № 2–3. С. 26–31.

Надійшла до редакції 28.09.2017

А.В. Анісімов, А.К. Новокшонов
ДОВІРЧІ ОБЧИСЛЕННЯ З ВИКОРИСТАННЯМ ДОДАВАЛЬНОЇ МАШИНИ. II

Анотація. Запропоновано розв'язання проблеми перевірки цілісності арифметичних програм з розгалуженнями і циклами, які виконуються на віддаленому обчислювальному ресурсі. Підхід до розв'язання полягає у заміні арифметичних операцій, таких як множення і ділення, відповідними 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. Для цифрового підпису застосовано модифіковану схему Бенало. Верифікація цифрових підписів результатів виконання програми гарантує виявлення будь-яких несанкціонованих змін у вихідному тексті програми.

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

A.V. Anisimov, A.K. Novokshonov
TRUSTED COMPUTING WITH ADDITION MACHINES. II

Abstract. A solution of the integrity verification problem for arithmetic programs with branching and looping statements running on a remote computing resource is proposed. The solution is to replace the arithmetic operations such as multiplication and division by corresponding 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. A modification of the Benaloh scheme is used for digital signatures implementation. Verification of digital signatures of computation results ensures detection of any unauthoried changes to the source code of the program.

Keywords: addition machine, digital signature, verifiable computing, 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.