Journal of Southeast University (English Edition) Vol. 32, No. 3, pp. 267 - 271 Sept. 2016 ISSN 1003—7985

A secure outsourced Turing-equivalent computation scheme against semi-honest workers using fully homomorphic encryption

Fang Hao Hu Aiqun

(School of Information Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: A scheme that can realize homomorphic Turing-equivalent privacy-preserving computations is proposed, where the encoding of the Turing machine is independent of its inputs and running time. Several extended private information retrieval protocols based on fully homomorphic encryption are designed, so that the reading and writing of the tape of the Turing machine, as well as the evaluation of the transition function of the Turing machine, can be performed by the permitted Boolean circuits of fully homomorphic encryption schemes. This scheme overwhelms the Turing-machine-to-circuit conversion approach, which also implements the Turing-equivalent computation. The encoding of a Turing-machine-to-circuit conversion approach is dependent on both the input data and the worst-case runtime. The proposed scheme efficiently provides the confidentiality of both program and data of the delegator in the delegator-worker model of outsourced computation against semi-honest workers.

Key words: Turing machine; fully homomorphic encryption; outsourced computation

DOI: 10.3969/j.issn.1003-7985.2016.03.002

The delegator-worker model is suitable for cloud computing, where the device of the user (i.e., the delegator) has limited computing resources, and the user outsources his/her program and data to the cloud servers (i.e., the workers), which usually have excellent computing resources. When the privacy of data is very important, fully homomorphic encryption (FHE) schemes[1-3] are the best cryptographic candidates, which enable the workers to operate the ciphertext given by the delegator without the private key and decryption process. One of the drawbacks of FHE schemes is that they only support the computation model of the circuit or the Boolean function, which is less flexible than the common Turing-machine-based computation model. As early as 1979, Pippenger et al.[4] showed how to encode a Turing machine (TM) into a circuit. However, the size of the encoded circuit is $O(N\log N)$, which is dependent on the worst runtime N of the original Turing machine.

Recently, Gentry et al.[5] proposed a new primitive model called the garbled random access machine (GRAM), which takes on the similar idea of the garbled circuit[6] but it is designed for random access machines. However, the GRAM needs to prepare as many garbled transition functions as runtime steps during the initialization by the delegator, which is still a large cost.

In this paper, we propose a reusable FHE-based scheme to describe and execute the Turing machine, where the size of the description is only dependent on the size of the transition function, and the whole work for the delegator is to encrypt some binary matrices bit-by-bit.

1 Preliminary

We use λ to denote the security parameter throughout, and use the negligible notation $f(\lambda)\leqslant \mathrm{negl}(\lambda)$ to show a function $f(\lambda)=O(\lambda^{-c})$ for every constant $c>0$.

We employ the fully homomorphic encryption scheme as the building block for our scheme, which usually consists of four algorithms KeyGen (λ), Enc (pk, m), Dec(sk, $[[m]]$) and Eval (pk, f, $[[m_0]]$, $[[m_1]]$, $\cdots$), where the plaintext message space belongs to $\{0,1\}$ and the abbreviation $[[m]]$ denotes the ciphertext of the corresponding plaintext m. The correctness of such a scheme guarantees that Dec (sk, Eval (pk, f, $[[m_0]]$, $[[m_1]]$, $\cdots$)) $=f(m_0,m_1,\cdots)$, where f can be the mod-2 addition (denoted as $\oplus$), the mod-2 multiplication (denoted as $\otimes$) or any other expressions based on these two operations. Without loss of generality, we assume that the FHE scheme is semantically secure, which, for any polynomial-time algorithm A, satisfies

$$\left|\Pr[A(\mathrm{Enc}(\mathrm{pk},0))=1]-\Pr[A(\mathrm{Enc}(\mathrm{pk},1))=1]\right|\leqslant \mathrm{negl}(\lambda)$$

In this paper, we employ the FHE primitives as black-boxes and do not care about the implementation details. For simplicity, when we use $\oplus$ and $\otimes$ between ciphertexts, we imply the calling to Eval ($\cdot$). The notation $[[x]]\cong[[y]]$ denotes the relationship Dec (sk, $[[x]]$) = Dec (sk, $[[y]]$), as that in almost all FHE schemes, the decryption is perfect (which means that no error will occur in decryption). The notations $[[\boldsymbol{m}]]$ or their $[[\boldsymbol{M}]]$ for a binary vector $\boldsymbol{m}$ or a binary matrix $\boldsymbol{M}$ are their shortcuts for the encrypted vector or matrix of

Received 2016-01-15.

Biographies: Fang Hao (1985—), male, graduate; Hu Aiqun (corresponding author), male, doctor, professor, aqhu@seu.edu.cn.

Foundation item: The National Basic Research Program of China (973 Program) (No. 2013CB338003).

Citation: Fang Hao, Hu Aiqun. A secure outsourced Turing-equivalent computation scheme against semi-honest workers using fully homomorphic encryption[J]. Journal of Southeast University (English Edition), 2016, 32(3): 267 - 271. DOI: 10.3969/j.issn.1003-7985.2016.03.002.

each element.

For a binary string $i \in \{0, 1\}^n$, we have two representations: i itself as an integer satisfies $i < 2^n$ and the vector $\boldsymbol{i} = \{i_0, i_1, \cdots, i_{n-1}\}$ as the binary form for further encryption, where $i_0, i_1, \cdots, i_{n-1} \in \{0,1\}$ satisfy $\sum_{j=0}^{n-1} 2^j i_j = i$.

2 Encryptable Turing Machine

For simplicity, we formalize a one-tape bound Turing machine as a 4-tuple $M = (Q, \Gamma, \delta, q_0)$, where $Q \subseteq \{0, 1\}^k$ is a set of states; $\Gamma \subseteq \{0,1\}^w$ is a set of symbols; $\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{\text{Left, Right}\}$ is the transition function, which takes the current state q_{current} and current symbol of the tape head h_{current} as input, outputs the next state q_{next}, the symbol to write by the head h_{output} and the moving direction of the head (L represents one step left and R represents one step right), and it is usually represented as a table, of which each row has the form:

$$q_{\text{current}} \| h_{\text{current}} \| q_{\text{next}} \| h_{\text{output}} \| \text{L/R}$$

where $q_0 \in Q$ is the initial state.

There may be some other configurations of a Turing machine in standard textbooks such as the blank symbol $b \in \Gamma$, input symbol set $\Sigma \subseteq \Gamma \setminus \{b\}$ and the final state set $F \subseteq Q$. We omit them for the following reasons: b and Σ can be directly taken into accounts when δ is designed, and F cannot be tested in a private outsourced scenario since it will leak the runtime of different inputs.

The input (and output) of such a Turing machine is a bound tape which consists of n cells. Each cell contains a w-bit binary string representing one symbol, and it is associated with an address of the length $l = \lfloor \log_2 n \rfloor$, by which we can locate where the head is. Note that we do not offer an unbound tape since algorithms usually have boundaries of space.

A honest-but-curious (or semi-honest) worker always follows the algorithms that the delegator gives to him, but he also tries to learn something from the given data. Obviously, if we encrypt every part of a Turing machine as well as its tape using a fully homomorphic encryption scheme, the worker will learn nothing due to the semantic security of FHE. It is not trivial to build an executable Turing machine based on only mod-2 additions and mod-2 multiplications, so we give the details about how to encrypt such a machine, including the transition function, the states, the symbols, the directions as well as the tape with addresses.

2.1 Simulation of the tape

The tape itself can be viewed as an $n \times w$ matrix $\boldsymbol{T}$, which can be encrypted directly as $[[\boldsymbol{T}]]$. However, we also need to hide the moves of the head, which is simulated as the changes of a head position variable $a \in \{0,1\}^l$ (Recall that $l = \lfloor \log_2 n \rfloor$). Obviously, a should be encrypted as $[[a]]$ in the worker's server, so we need an algorithm to read a cell of $[[\boldsymbol{T}]]$, given $[[\boldsymbol{a}]]$. This is the task of a private information retrieval protocol. For the context of FHE, we refer readers to the details in Ref. [7]. Algorithm HeadRead is inspired by the algorithms in Ref. [7] with some modifications for our scenario, as shown in Algorithm 1.

Algorithm 1 HeadRead

Input: $[[\boldsymbol{T}]] = \{[[t_{i,j}]]\} \in \{[[0]], [[1]]\}^{n \times w}$, $[[\boldsymbol{a}]] = \{[[a_j]]\} \in \{[[0]], [[1]]\}^l$.

Output: $[[\boldsymbol{h}]] = \{[[h_j]]\} \in \{[[0]], [[1]]\}^w$.

1 for $b = 0$ to $n - 1$ do

2 Encrypt b to yield $[[b]] = ([[b_0]], \cdots, [[b_{l-1}]]) \in \{[[0]], [[1]]\}^l$

3 $[[d_i]] \leftarrow \otimes_{j=0}^{l-1}([[a_j]] \oplus [[b_j]] \oplus [[1]])$

4 end

5 for $j = 0$ to $w - 1$ do

6 $[[h_j]] \leftarrow \oplus_{i=0}^{n-1}([[d_i]] \otimes [[t_{i,j}]])$

7 end

8 return $[[h]]$

In short, lines 1-4 calculate $[[d_j]]$ for all the n cells, which satisfy $[[d_a]] \cong [[1]]$ and $[[d_{j \neq a}]] \cong [[0]]$. These $[[d_j]]$ can be viewed as selectors of cells. Lines 5-7 make each row of $[[\boldsymbol{T}]]$ be checked by $[[d_j]]$ so that only the value of $[[t_{a,j}]]$ leaves in $[[h_j]]$. Similarly, there is a HeadWrite algorithm to write the symbol $[[\boldsymbol{h}]]$ in $[[\boldsymbol{T}]]$ at the location $[[\boldsymbol{a}]]$ as shown in algorithm 2.

Algorithm 2 HeadWrite

Input: $[[\boldsymbol{T}]]$, $[[\boldsymbol{a}]]$, $[[\boldsymbol{h}]] = ([[h_0]], \cdots, [[h_{w-1}]]) \in \{[[0]], [[1]]\}^w$.

Output: $[[\boldsymbol{T}]]$.

1 for $b = 0$ to $n - 1$ do

2 Encrypt b to yield $[[b]] = ([[b_0]], \cdots, [[b_{l-1}]]) \in \{[[0]], [[1]]\}^l$

3 $[[d_i]] \leftarrow \otimes_{j=0}^{l-1}([[a_j]] \oplus [[b_j]] \oplus [[1]])$

4 end

5 for $i = 0$ to $n - 1$ do

6 $[[\bar{d}_i]] \leftarrow [[d_i]] \oplus [[1]]$

7 for $j = 0$ to $w - 1$ do

8 $[[t_{i,j}]] \leftarrow ([[\bar{d}_i]] \otimes [[t_{i,j}]]) \oplus ([[d_i]] \otimes [[h_j]])$

9 end

10 end

11 return $[[\boldsymbol{T}]]$

As we can see, lines 1-4 are identical to those of HeadRead and share the same functionality. Lines 5-10 make use of $[[d_i]]$ and its negation $[[\bar{d}_i]]$ to implement the following functionality: When the cell is selected, the current data is replaced with the input $[[h_j]]$, and otherwise the original data keep unchanged.

Finally, we consider the move of the tape head. We assume that the leftmost cell of the tape has the address 0

and the rightmost cell of the tape has the address $n-1$. Then we encode the action "one step left" as $[[1]]$ and the action "one step right" as $[[0]]$, and then use the HeadMove algorithm as follows.

Algorithm 3 HeadMove

Input: The current address $[[\boldsymbol{a}]]$ and the direction $[[\mathrm{dir}]]$.

Output: The next address $[[\boldsymbol{a}']]$.

1 $[[d_{l-1}]] \leftarrow [[1]]$
2 for $i=0$ to $l-2$ do
3 $[[d_i]] \leftarrow [[\mathrm{dir}]]$
4 end
5 $[[\boldsymbol{d}]] = ([[d_0]], \cdots, [[d_{l-1}]])$
6 $[[\boldsymbol{a}']] \leftarrow \mathrm{FullAdder}([[\boldsymbol{a}]], [[\boldsymbol{d}]])$
7 return $[[\boldsymbol{a}']]$

Lines 1-4 make $d=2^l-1$ when dir = 1 (one step left) and $d=1$ when dir = 0 (one step right), which are the two's complements of ∓ 1 of l-bit integers exactly. Then we call the standard Boolean function of l-bit full adder to implement the decrement and increment of the address of the head.

2.2 Simulation of the transition function

The transition function is the kernel of the Turing machine. Applying the encoding of directions, each row of the table of the transition function δ becomes a pure $(2k+2w+1)$-bit binary string. Assuming that there are m rows in the table, we use an $m\times(2k+2w+1)$ matrix $\boldsymbol{F}$ to store it, which can be encrypted directly as $[[\boldsymbol{F}]]$. Given the current state $[[\boldsymbol{q}]]$ and current symbol read by the head $[[\boldsymbol{h}]]$, we can use the following MoveStep algorithm to obtain the next state $[[\boldsymbol{q}']]$, the output of the head $[[\boldsymbol{h}']]$ and the direction of the move of the head $[[\mathrm{dir}]]$.

Algorithm 4 MoveStep

Input: $[[\boldsymbol{F}]] = \{[[f_{i,j}]]\}$, $[[\boldsymbol{q}]] \in \{[[0]], [[1]]\}^k$, $[[\boldsymbol{h}]] \in \{[[0]], [[1]]\}^w$.

Output: $[[\boldsymbol{q}']] \in \{[[0]], [[1]]\}^k$, $[[\boldsymbol{h}']] \in \{[[0]], [[1]]\}^w$, $[[\mathrm{dir}]] \in \{[[0]], [[1]]\}$.

$[[\boldsymbol{b}]] \leftarrow ([[q_0]], \cdots, [[q_{k-1}]], [[h_0]], \cdots, [[h_{w-1}]])$
for $i=0$ to $m-1$ do
$[[d_i]] \leftarrow \otimes_{j=0}^{k+w-1}([[f_j]] \oplus [[b_j]] \oplus [[1]])$
end
for $j=0$ to $k+w$ do
$[[c_j]] \leftarrow \oplus_{i=0}^{m-1}([[d_i]] \otimes [[f_{i,k+w+j}]])$
end
$[[\boldsymbol{q}']] \leftarrow ([[c_0]], \cdots, [[c_{k-1}]])$; $[[\boldsymbol{h}']] \leftarrow ([[c_k]], \cdots, [[c_{k+w-1}]])$; $[[\mathrm{dir}]] \leftarrow [[c_{k+w}]]$
return $[[\boldsymbol{q}']]$, $[[\boldsymbol{h}']]$, $[[\mathrm{dir}]]$

The whole process is similar to the HeadRead algorithm, except for that we use the first $k+w$ columns (the states and symbols; i. e. the inputs of δ) of $[[\boldsymbol{F}]]$ as addresses for matching instead of the natural row number, and use the last $k+w+1$ columns of $[[\boldsymbol{F}]]$ as contents.

2.3 Process of evaluation

Now a Turing machine and its tape can be described as $[[\boldsymbol{T}]]$, $[[\boldsymbol{F}]]$ as well as the encrypted initial state $[[\boldsymbol{q}_0]]$. As we mentioned before, while executing an encrypted Turing machine, we cannot let the worker check whether the machine halts. Instead, we specify a worst-case runtime N. Putting all the things together, we obtain the TMEval algorithm.

Algorithm 5 TMEval

Input: $[[\boldsymbol{T}]]$, $[[\boldsymbol{F}]]$, $[[\boldsymbol{q}_0]]$, N.

Output: $[[\boldsymbol{T}]]$ and the final state $[[\boldsymbol{q}]]$.

1 $[[\boldsymbol{q}]] \leftarrow [[\boldsymbol{q}_0]]$ /* the state register */
2 $[[\boldsymbol{a}]] \leftarrow [[\boldsymbol{b}_0]]$ /* the head position register */
3 for $i=1$ to N do
4 $[[\boldsymbol{h}]] \leftarrow \mathrm{HeadRead}([[\boldsymbol{T}]], [[\boldsymbol{a}]])$ /* the current input symbol */
5 $[[\boldsymbol{q}]], [[\boldsymbol{h}]], [[\mathrm{dir}]] \leftarrow \mathrm{MoveStep}([[\boldsymbol{F}]], [[\boldsymbol{q}]], [[\boldsymbol{h}]])$
6 $\mathrm{HeadWrite}([[\boldsymbol{T}]], [[\boldsymbol{a}]], [[\boldsymbol{h}]])$
7 $[[\boldsymbol{a}]] \leftarrow \mathrm{HeadMove}([[\boldsymbol{a}]], [[\mathrm{dir}]])$
8 end
9 return $[[\boldsymbol{T}]], [[\boldsymbol{q}]]$

Lines 4-7 represent a complete cycle of a single MoveStep of the evaluation of the Turing machine. If the delegator only wants a few bits of the tape, he can use HeadRead as a private information retrieval algorithm to achieve this.

3 Outsourced Computation Based on the Encryptable Turing Machine

3.1 Definition

We follow the definition of the formal description of outsourced computation schemes in Ref. [8]. An outsourced computation consists of the following three algorithms:

1) $\mathrm{FuncKeyGen}(F,\lambda)\rightarrow(\mathrm{PK},\mathrm{SK})$ encodes the program F as the public key PK and outputs a secret key SK kept in the hands of the delegator. λ is the security parameter for verifiability.

2) $\mathrm{ProbGen}_{\mathrm{SK}}(x)\rightarrow(\sigma_x,\tau_x)$ encrypts the function input x as a public value σ_x and a private value τ_x.

3) $\mathrm{Compute}_{\mathrm{PK}}(\sigma_x)\rightarrow\sigma_y$ computes the encrypted result of $y=F(x)$, using PK and σ_x.

The privacy of outsourced computation schemes is defined by the experiment $\mathrm{Exp}_{\mathrm{A}}^{\mathrm{Priv}}(F,\lambda)$ as shown in algorithm 6, where the oracle $\mathrm{PubProbGen}_{\mathrm{SK}}(\cdot)$ calls $\mathrm{ProbGen}_{\mathrm{SK}}(\cdot)$ to generate (σ_x,τ_x) and returns the public part σ_x[8].

Algorithm 6 $\mathrm{Exp}_{\mathrm{A}}^{\mathrm{Priv}}(F,\lambda)$

Input: Adversary A, function F, security parameter λ.

Output: The result of experiement: "succ" or "fail".

$F \leftarrow A(\cdot)$; $(\mathrm{PK}, \mathrm{SK}) \leftarrow \mathrm{FuncKeyGen}(F, \lambda)$
$(x_0, x_1) \leftarrow A^{\mathrm{PubProbGen}(\cdot)}(\mathrm{PK})$
$(\sigma_0, \tau_0) \leftarrow \mathrm{ProbGen}_{\mathrm{SK}}(x_0)$; $(\sigma_1, \tau_1) \leftarrow \mathrm{ProbGen}_{\mathrm{SK}}(x_1)$
$b \leftarrow_{\mathrm{R}} \{0,1\}$; $\hat{b} \leftarrow A^{\mathrm{PubProbGen}(\cdot)}(\mathrm{PK}, x_0, x_1, \sigma_b)$
if $\hat{b} = b$ then output "succ"
else output "fail"
end

Definition 1 A outsourced computation scheme is said to be private for a function F, if for any probabilistic polynomial time adversary A,

$$\left| \Pr[\exp_A^{\mathrm{Priv}}(F, \lambda) \text{ outputs succ}] - \frac{1}{2} \right| \leqslant \mathrm{negl}(\lambda)$$

3.2 Our scheme

Now we present our outsourced computation scheme based on the encryptable Turing machine (ETM):

1) $\mathrm{FuncKeyGen}((\boldsymbol{F}, \boldsymbol{q}_0, N), \lambda) \rightarrow (\mathrm{PK}, \mathrm{SK})$: $(\mathrm{pk}, \mathrm{sk}) \leftarrow \mathrm{KeyGen}(\lambda)$, $\mathrm{PK} \leftarrow (\mathrm{pk}, [[\boldsymbol{F}]], [[\boldsymbol{q}_0]], N)$, $\mathrm{SK} \leftarrow (\mathrm{pk}, \mathrm{sk})$.

2) $\mathrm{ProbGen}_{\mathrm{SK}}(\boldsymbol{X}) \rightarrow (\sigma_X, \tau_X)$: $\sigma_X \leftarrow [[\boldsymbol{X}]]$, $\tau_X \leftarrow \boldsymbol{X}$.

3) $\mathrm{Compute}_{\mathrm{PK}}(\sigma_X) \rightarrow \sigma_Y$: $\sigma_Y \leftarrow \mathrm{TMEval}([[\boldsymbol{X}]], [[\boldsymbol{F}]], [[\boldsymbol{q}_0]], N)$.

Theorem 1 If there exists a semantically-secure fully homomorphic encryption (KeyGen, Enc, Dec, Eval), then our outsourced computation scheme is private for all F.

Proof Assume that there is an adversary A satisfying

$$\mathrm{Adv}_A^{\mathrm{Priv}} = \left| \Pr[\exp_A^{\mathrm{Priv}}(F, \lambda) \text{ outputs succ}] - \frac{1}{2} \right| = \varepsilon$$

where ε is non-negligible. Then, we construct an adversary A' for breaking the semantic security experiment of the FHE scheme with the challenger C' as follows:

A' first receives PK' from C', and then sends $m_0 = 0$ and $m_1 = 1$ to C'. C' picks up a b' and sends $[[m'_b]]$ back. Now A' interacts with A as the challenger. He/She answers the $\mathrm{PubProbGen}_{\mathrm{SK}}(\cdot)$ oracle and FuncKeyGen using $\mathrm{pk} = \mathrm{pk}'$ and a random string as sk. Once A sends two plaintext tapes $\boldsymbol{X}_0$, $\boldsymbol{X}_1$ to A', he/she first calculates σ_{X0} and σ_{X1} as normal, then replaces all those in σ_{X1} with $[[m'_b]]$. A' always sends σ_{X1} to A.

With the probability $1/2$, $\mathrm{Dec}(\sigma_{X1}) = \boldsymbol{X}_1$, and in this case, A will succeed in ε. With the rest probability $1/2$, $\mathrm{Dec}(\sigma_{X1})$ is 0, and in this case, A may not have the advantages leading to success, and in the worst case, A will succeed in $\mathrm{negl}(\lambda)$.

Let A' output what A outputs, then in the worst case, we have

$$\begin{aligned}\Pr[A' \text{ wins}] = \Pr[\hat{b} = b] = {} & \\ & \Pr[\exp_A^{\mathrm{Priv}}(F, \lambda) \text{ outputs succ} \mid b = 1] + \\ & \Pr[\mathrm{Guess} \mid b = 0] = \\ & \frac{1}{2} \cdot \left(\frac{1}{2} + \varepsilon \right) + \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{2} + \mathrm{negl}(\lambda)\end{aligned}$$

which leads to a contradiction of the semantic security of the FHE scheme. Therefore, no such A exists.

4 Comparisons

There are several general secure multi-party computation schemes that can efficiently compute arbitrary functions without leaking private data to other parties, such as Yao's classical garbled circuit scheme[6], the recent distributed oblivious RAM scheme[9] and other similar schemes. However, all these schemes require the client (in our context, the delegator) to interactively communicate with the server(s) (in our context, the worker(s)) during the computation process, which is unacceptable in the outsourced computation scenario, in which the delegator merely sends the data to the worker and fetches the result from the worker later. To the best of our knowledge, this non-interactive general secure computation can be achieved by only the fully homomorphic encryption schemes.

So, we mainly compare our implementation of encryptable Turing machines to the 1970s' oblivious Turing machine (OTM), as the latter is still widely used in recent FHE-based studies[10-11]. As we claimed above, in our proposal, the description of Turing machine F is not dependent on the worst-case runtime N. We summarize the comparisons in Tab. 1.

Tab. 1 Comparisons of different Turing-equivalent computation schemes

Scheme	Worst-case runtime	Encoded size
Original TM	N	$O(1)$
TM-to-circuit schemes[4]	$O(N)$	$O(N\log N)$
Oblivious TM[4,10-11]	$O(N\log N)$	$O(1)$
GRAM[5]	$O(N)$	$O(N)$
Our scheme	$O(N)$	$O(1)$

5 Conclusion

We propose a scheme that enables semi-honest servers to execute Turing-equivalent computations on encrypted data, where the encoding of the Turing machine is independent of its inputs and running time. Future work will be to model the encryptable Turing machine against malicious adversaries, who may return incorrect result to the user, and the user must have some efficient method to discover this malicious behavior. It is always possible, since there are standard compilers[12], to compile any secure multi-party computation protocol against a semi-honest adversary to a more secure protocol against malicious adversaries. However, all these compilers lower the efficiency of the secure multi-party computation protocol enormously. We plan in the future to implement a scheme against malicious adversaries by utilizing the special structure of the encryptable Turing machine and thus, thereby, achieve a more accurate result.

References

[1] Gentry C. Fully homomorphic encryption using ideal lattices[C]//*Proceedings of the* 41*st Annual ACM Symposium on Theory of Computing*. Bethesda, USA, 2009: 169.

[2] Brakerski Z, Gentry C, Vaikuntanathan V. (Leveled) fully homomorphic encryption without bootstrapping [C]//*Proceedings of the* 3*rd Innovations in Theoretical Computer Science Conference*. Cambridge, USA, 2012: 309 - 325.

[3] Gentry C, Sahai A, Waters B. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based [C]//*Advances in Cryptology—CRYPTO* 2013. Berlin, Germany: Springer, 2013: 75 - 92.

[4] Pippenger N, Fischer M J. Relations among complexity measures[J]. *Journal of the ACM*, 1979, **26**(2): 361 - 381. DOI:10.1145/322123.322138.

[5] Gentry C, Halevi S, Lu S, et al. Garbled RAM revisited [C]//*Advances in Cryptology—EUROCRYPT* 2014. Berlin, Germany: Springer, 2014: 405 - 422.

[6] Yao A C C. Protocols for secure computations [C]// 23*rd Annual Symposium on Foundations of Computer Science*. Chicago, USA, 1982: 160 - 164.

[7] Yi X, Kaosar M G, Paulet R, et al. Single-database private information retrieval from fully homomorphic encryption[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2013, **25**(5): 1125 - 1134. DOI: 10.1109/tkde.2012.90.

[8] Gennaro R, Gentry C, Parno B. Non-interactive verifiable computing: Outsourcing computation to untrusted workers[C]//*Advances in Cryptology—CRYPTO* 2010. Berlin, Germany: Springer, 2010: 465 - 482. DOI:10.1007/978-3-642-14623-7_25.

[9] Lu S, Ostrovsky R. Distributed oblivious RAM for secure two-party computation[C]//10*th Theory of Cryptography Conference*. Tokyo, Japan, 2013: 377 - 396. DOI:10.1007/978-3-642-36594-2_22.

[10] Goldwasser S, Kalai Y T, Popa R A, et al. How to run turing machines on encrypted data [C]//*Advances in Cryptology—CRYPTO* 2013. Berlin, Germany: Springer, 2013: 536 - 553.

[11] Boyle E, Chung K M, Pass R. On extractability obfuscation[C]//11*th Theory of Cryptography Conference*. San Diego, USA, 2014: 52-73. DOI: 10.1007/978-3-642-54242-8_3.

[12] Chung K M, Kalai Y, Vadhan S. Improved delegation of computation using fully homomorphic encryption[C]// *Advances in Cryptology—CRYPTO* 2010. Berlin, Germany: Springer, 2010: 483 - 501. DOI: 10.1007/978-3-642-14623-7_26.

基于全同态加密的抵抗半诚实工作者的安全外包图灵等效计算方案

方 昊 胡爱群

(东南大学信息科学与工程学院, 南京 210096)

摘要:提出了一种可进行私密同态图灵等效计算的方案,该方案中图灵机的编码与图灵机的输入及图灵机的运行时间无关.设计了若干种基于全同态加密的隐私数据检索协议扩展,从而用全同态加密所允许的布尔电路运算实现了对图灵机纸带的读写和对图灵机转移函数的执行.此方案明显优于同样实现了图灵等效计算的图灵机-电路转换方案.在图灵机-电路转换方案中,图灵机的编码既依赖于输入数据,也依赖于最坏运行时间.本方案在安全外包计算的委托者-工作者模型中可高效地为委托者提供程序机密性和数据机密性以对抗半诚实工作者.

关键词:图灵机;全同态加密;外包计算

中图分类号:TP309.7