Towards Practical Topology-Hiding Computation

Shuaishuai Li

State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences

School of Cyber Security, University of Chinese Academy of Sciences

ASIACRYPT 2022

What is Topology-Hiding Computation?

Secure multiparty computation (MPC): n parties securely compute some function without leaking additional information about the inputs.

Classical MPC: every two parties can communicate directly (i.e., the communication graph is complete):

What if the communication graph is incomplete and even sensitive?

Defining Topology-Hiding Computation

Moran, Orlov and Richelson ([MOR15]) formalized the concept of topology-hiding computation (THC).

Informal definition 1

A THC protocol is an MPC protocol over incomplete graph which does not leak any information about the communication graph.

Feasibility of THC I

In the setting that the adversary may statically, passively corrupt any number of parties, THC has been shown to be feasible on any graph.

Akavia, LaVigne and Moran ([ALM17]) showed that topology-hiding broadcast (THB) is feasible on general graphs. This in fact implies that THC computing any function exists on general graphs due to the following lemma.

Lemma 1 (THB implies THC)

If there exists THB for some graph class and a PKE scheme, then THC computing any function exists for the same graph class.

Feasibility of THC II

We know that MPC computing any function exists. To construct a THC protocol for computing some function f, we just use THB and PKE to simulate point-to-point channels in an MPC protocol computing f:

- Assume P_i wants to send x to P_j, they do the follows.
  1. P_j uses THB to broadcast its public key.
  2. To send a message x to P_j, P_i encrypts x using the public key of P_j and then uses THB to broadcast the resulting ciphertext.
  3. Upon receiving the ciphertext, P_j can decrypt it to get x. Other parties know nothing about x because they do not know the decryption key.

Basic Tools: PKCR Encryption

Privately Key-Commutative and Rerandomizable (PKCR) Encryption [AM17] is a special PKE scheme.

With PKCR, one can add or delete a public key layer to a ciphertext. Namely, with $[\![x]\!]_k$ and pk,

- Add layer: one can generate $[\![x]\!]_{k \cdot pk}$.
- Delete layer: one can generate $[\![x]\!]_{k \cdot pk^{-1}}$.

PKCR with homomorphism: in our paper, we require PKCR to be linearly homomorphic. Both the BCP ([BCP03]) and CL ([CL15]) schemes are linearly homomorphic PKCR encryption. We refer to the paper for more details about PKCR.

Our Work: Improving the Efficiency of THC

Although THC is feasible, existing THC protocols are not practical. We improve several existing THC protocols on cycles and general graphs.

State-of-the-Art Topology-Hiding Broadcast I

The Akavia-Moran (AM) and Akavia-LaVigne-Moran (ALM) protocols are the state-of-the-art THB protocols for cycles and general graphs, respectively.

- They are designed for broadcasting a single bit (a bitstring can be broadcast bit-by-bit)
- They are built by first presenting a topology-hiding OR protocol and then letting the broadcaster take the broadcast bit as input and each other party take 0 as input.

State-of-the-Art Topology-Hiding Broadcast II

The AM and ALM protocols can be described in the same framework, which consists of two phases: Aggregate phase and Decrypt phase.

Aggregate phase. This phase takes T (AM takes $T = n - 1$ and ALM takes $T = 8n^3\kappa$) rounds.

1. At the first round, for each party P_i and each of its neighbor d, P_i encrypts its input bit to c under a fresh public key pk and sends (pk, c) to its neighbor d.
2. At each following round, each party P_i does the followings.
   1. Choose a permutation σ of the set of its neighbors (AM uses the only non-identity permutation. ALM uses a fresh random permutation)
   2. For each ciphertext received from neighbor d, add a public key layer and homomorphically OR the input bit to this ciphertext, and then send the resulting ciphertext to neighbor $\sigma(d)$.

State-of-the-Art Topology-Hiding Broadcast III

Decrypt phase. At the end of the aggregate phase, each received ciphertext has been the OR of all the inputs. To decrypt these ciphertexts, the parties send each ciphertext back through the same walk it traversed during the aggregate phase, and each party deletes its own public key layer in the reversed walk.

Improving the AM and ALM Protocols I

In the journal version of the paper of Akavia, LaVigne and Moran, the authors observed that computing $[\![a \vee b]\!]$ from a and $[\![b]\!]$ does not require any homomorphic property of PKCR:

If $a = 1$, *output* $[\![1]\!]$. *Otherwise, output* $[\![b]\!]$.

Our observation is that computing OR is overkill for broadcast. If we only consider broadcast, then we let the parties do the following:

If the computing party is the broadcaster, output $[\![a]\!]$. *Otherwise, output* $[\![b]\!]$.

In this way, the broadcast value does not need to be a bit, i.e., any plaintext can be broadcast!

Improving the AM and ALM Protocols II

- Original AM and ALM: each party needs to send a ciphertext (a ciphertext is of length at least $O(\kappa)$ bits) to each neighbor at each round for broadcasting just a single bit.
- Our optimization: if we instantiate PKCR with the ElGamal scheme, we can set plaintext length to κ bits and ciphertext length to 2κ bits; namely, κ bits can be broadcast with each party sending a ciphertext of length $O(\kappa)$ bits to each neighbor at each round.

As a result:

The communication cost of the AM protocol for broadcasting κ bits is reduced from $O(n^2\kappa^2)$ bits to $O(n^2\kappa)$ bits.

The communication cost of the ALM protocol for broadcasting κ bits is reduced from $O(n^5\kappa^3)$ bits to $O(n^5\kappa^2)$ bits.

State-of-the-Art Topology-Hiding Sum on Cycles I

The sum functionality: receive a message x_i of length $O(\kappa)$ bits from each party P_i; send $\sum_{i\in[n]} x_i$ to all parties.

The only topology-hiding sum protocol on cycles is compiled from the AM protocol. Using additively homomorphic encryption (e.g., the Paillier scheme), a secure sum protocol can be constructed as follows.

1. Each party encrypts its input and sends the resulting ciphertext to P_1.
2. P_1 homomorphically adds all the ciphertexts.
3. The parties distributedly decrypt the final ciphertext.

State-of-the-Art Topology-Hiding Sum on Cycles II

The above protocol has linear communication cost $O(n\kappa)$ bits (we refer to the paper for more details).

Recall that topology-hidingly sending a bit using the AM protocol will cost $O(n^2\kappa)$ bits.

By compiling this sum protocol into a THC protocol, we obtain a topology-hiding sum protocol with communication cost $O(n^3\kappa^2)$ bits.

More Efficient Topology-Hiding Sum on Cycles I

Our idea: instead of compiling black-box from THB, we modify the AM protocol to be a topology-hiding sum protocol. To achieve this, we require the parties to know the value of n and the PKCR to be additively homomorphic.

Remark: the AM protocol only assumes that the parties know an upper bound of n, so does the THS protocol compiled from the AM protocol.

More Efficient Topology-Hiding Sum on Cycles II

In the AM protocol, each party does the followings.

Homomorphically OR its input bit to each received ciphertext.

Instead, we let each party do the following in our topology-hiding sum protocol.

Homomorphically add its input value to each received ciphertext.

Efficiency: our THS protocol has the same asymptotic communication cost as the AM protocol, i.e., $O(n^2\kappa)$ bits.

State-of-the-Art General Topology-Hiding Computation on Cycles I

The general computation functionality: receive a message x_i of length $O(\kappa)$ bits from each party P_i and a function f containing c multiplication gates from all the parties; send $f(x_1, \ldots, x_n)$ to P_1.

The only general topology-hiding computation protocol on cycles is compiled from the AM protocol. We consider the protocol based on additive secret sharing, which contains the following three phases.

1. **Input sharing.** In this phase, each party additively shares its input, which requires to send $O(n)$ shares. (Total communication: $O(n^2\kappa)$ bits.)

State-of-the-Art General Topology-Hiding Computation on Cycles II

2. **Circuit evaluation.** In this phase, the parties compute the circuit gate-by-gate. Addition gates can be computed locally. The parties can compute a multiplication gate using linearly homomorphic encryption (e.g., the Paillier scheme), resulting $O(n\kappa)$ bits communication (we refer to the paper for more details). (Total communication: $O(cn\kappa)$ bits with c being the number of multiplication gates.)
3. **Output recovery.** To recover the output, each party sends its share to P_1 and then P_1 computes the sum of all shares. (Total communication: $O(n\kappa)$ bits.)

By compiling the above MPC protocol into a THC protocol, we obtain a general topology-hiding computation protocol with communication cost $O(n^4\kappa^2 + cn^3\kappa^2)$ bits.

Better TH Input Sharing and Output Recovery

With our THS protocol, we can share the inputs and recover the output more efficiently.

- To share an input x, the input owner P_i samples a random value r and each other party P_j samples a random share x_j. Then the parties execute our THS protocol where P_i takes $x + r$ as input and each other party P_j takes $-x_j$ as input. All parties will get $z = x + r - \sum_{j \neq i} x_j$. P_i takes $x_i = z - r$ as its share. (Total communication: $O(n^3\kappa)$ bits.)
- To recover the output, the parties execute our THS protocol where each party takes its share as input. (Total communication: $O(n^2\kappa)$ bits.)

Better TH Secure Multiplication I

The key point of circuit evaluation: secure multiplication.

The multiplication functionality: receive two additive sharings $\langle x \rangle, \langle y \rangle$ from the parties; return $\langle xy \rangle$.

Starting point: $\langle xy \rangle$ can be computed as follows.

1. The parties generate a random sharing $\langle r \rangle$.
2. The parties execute a protocol to let all parties securely get the value $xy - r$.
3. The parties locally compute $\langle xy \rangle = xy - r + \langle r \rangle$.

Note that $\langle r \rangle$ can be locally generated by letting each party sample a random value r_i and setting $r = \sum_{i \in [n]} r_i$. Now the goal is to publish the value $xy - r$.

Better TH Secure Multiplication II

To achieve this, we require the parties to know the value of n and the PKCR to be linearly homomorphic.

To publish the value $xy - r$, we divide the aggregate phase into two subphases where each takes $n - 1$ rounds.

In the first subphase, each party does the same as in our THS protocol.

Homomorphically add its share of x to each received ciphertext.

At the end of the first subphase, the parties have encryptions of x.

Better TH Secure Multiplication III

In the second subphase, each party P_i does the followings.

> Compute a ciphertext $c = [\![xy_i - r_i]\!]$ *from* $[\![x]\!], y_i, r_i$*; then homomorphically add* c *to the received ciphertext.*

At the end of the second subphase, each party has encryptions of $xy - r$. Finally, the parties execute the decrypt phase as in the AM protocol.

Efficiency: our topology-hiding multiplication protocol has the same asymptotic communication cost as the AM protocol, i.e., $O(n^2\kappa)$ bits.

Improving the LZM^{3}T protocol I

Our final result: an optimization for the LZM^{3}T protocol (LaVigne et al. in TCC 2018)

This protocol is an FHE-based general topology-hiding computation protocol on general graphs. In the aggregate phase of this protocol, each party P_i does

Append the ciphertexts of its input x_i and its ID id_i to each received ciphertext.

At the end of the aggregate phase, each party P_i will receive $T = 8n^3\kappa$ pairs of ciphertexts. The corresponding messages can be parsed as encryptions of all the inputs. Then, the parties can homomorphically compute the given function.

Improving the LZM^{3}T protocol II

Efficiency of LZM^{3}T: each party sends a ciphertext vector of length $O(t)$ to each neighbor at the t-th round and the total number of rounds is $T = O(n^3\kappa)$, which yields

$$O((1 + 2 + \cdots + T) \cdot |E|) = O(T^2 n^2) = O(n^8\kappa^2)$$

ciphertexts communication during the aggregate phase.

Our result: we give an optimization for the aggregate phase of the LZM^{3}T protocol to reduce the communication.

Improving the LZM^{3}T protocol III

Our idea: in the aggregate phase, each party sends ciphertext vectors of length n at each round and

> *for the i-th entry of the ciphertext vectors, the parties act exactly as in the optimized ALM protocol with P_i being the broadcaster and the input x_i of P_i being the broadcast value.*

This way, at the end of the aggregate phase, the last party in each walk will get a ciphertext vector $([\![x_1]\!], \ldots, [\![x_n]\!])$!

Efficiency of our optimization: our optimized aggregate phase only requires the parties to send $O(nT \cdot |E|) = O(n^6\kappa)$ ciphertexts.

Thanks for listening!