Hiding Input Size in Secure Computation - Presentation Notes

Alex Irpan

June 7, 2016

1 Introduction

Many secure computation protocols convert the desired function $f(x, y)$ to a circuit, then do operations using that circuit. Doing so implicitly forces each party to reveal their input sizes. In some domains, this leaks nothing, because input size is fixed or unimportant. For example, in voting protocols, a vote is either 0 or 1, so everyone's input is known to be one bit long. In other cases, we may have a prior upper bound on input length. For Yao's millionaire problem, we can safely assume no one has more than 2^{64} dollars, so the party that constructs the circuit can assume each input is 64 bits.

However, in problems where there is no public upper bound on input size, sharing input size may leak information. For example, learning the length of a query to a database could leak what that party wants to learn. This motivates protocols designed to hide the input length.

These notes primarily details the results by Lindell, Nissim, Orlandi 2013, which both gives a definition of security and proves when input size can be hidden and when it cannot [1]. Their results are all specific to semi-honest adversaries. Further work by Chase, Ostrovsky, Visconti 2015 extends this work to malicious adversaries, and also introduces the idea of "executable proofs" [2]. We leave this for another time.

2 Definitions

Previous definitions of security did not explicitly consider input size. However, when input size is either hidden or revealed, we need to modify several security definitions. For simplicity, we only consider the two party case.

Again, we base security on indistinguishability from the ideal model. The general form of the ideal model is

- Parties P_1, P_2 send inputs x, y to the third party
- The third party computes $f(x, y)$.
- The third party sends length metadata to the appropriate parties.
- The third party sends $f(x, y)$ to the appropriate parties.

The length metadata is given as $1^{|x|}$ or $1^{|y|}$. This allows simulators to run in polynomial time on the length of the other party's input in the semi-honest setting.

Occasionally, learning the size of $f(x, y)$ leaks something about the input lengths. For example, let $f(x, y) = g(x||y)$ where g is a one-way permutation. If a party learns $|f(x, y)|$ without learning $f(x, y)$, that party can compute both parties input length. Thus, we also need to consider whether the third party lets a party receives $1^{|f(x,y)|}$

Note this security definition can be seen as an extension of the standard security definition where input lengths are revealed. When both input lengths are revealed, the third party broadcasts all length metadata.

Finally, although we will only construct protocols for semi-honest adversaries, we discuss a defintion for malicious adversaries. A malicious party may choose their input length based on the honest party's input length. For example, if P_1 has $|x| = k$, P_2 may pretend to have input length 2^k. If P_2's input size is hidden, the runtime is exponential in the size of revealed inputs. This motivates the following definition.

Definition 1 *Security is defined as follows.*

- *In honest execution, the runtime is polynomial in input length, output length, and κ.*
- *when a party is malicious, the runtime is polynomial in input length, κ,* ***and the run time of the corrupted party****.*

This lets us define polytime simulators that use only public information for both the semi-honest and malicious case.

3 Class 1 Results

In class 1 computation, exactly one party reveals input size. We assume P_1 reveals input size $|x|$ and P_2 hides it. The variations are as follows.

- Class 1.a: both parties get $f(x, y)$.
- Class 1.b: only P_1 gets $f(x, y)$.
- Class 1.c: P_1 gets $f(x, y)$, P_2 gets $1^{|f(x,y)|}$
- Class 1.d: only P_2 gets $f(x, y)$
- Class 1.e: P_1 gets $1^{|f(x,y)|}$, P_2 gets $f(x, y)$.

We show that Class 1.a,1.c,1.e are always achievable, and Class 1.b and 1.d are not always achievable.

The construction will require Fully-Homomorphic Encryption.

Definition 2 *Consider a tuple* $(\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval})$ *of PPT algorithms defined as follows.*

- $(pk, sk) \leftarrow \text{Gen}(1^k)$ *is a key generator*
- $c \leftarrow \text{Enc}_{pk}(m)$ *encrypts m with pk.*
- $m' \leftarrow \text{Dec}_{sk}(c)$ *outputs a message or $\perp$.*

- $\mathcal{C}' \leftarrow \text{Eval}_{pk}(\mathcal{C})$ *gives a circuit that can be evaluated on ciphertexts.*

This tuple is a FHE with circuit privacy if

- $(\text{Gen}, \text{Enc}, \text{Dec})$ *is a CPA-secure public-key encryption scheme*
- *Correctness: for any polysize circuit $\mathcal{C}$ and input m,*

$$\Pr[\mathcal{C}(m) \neq \text{Dec}_{sk}(\mathcal{C}'(\text{Enc}_{pk}(m)))]$$

is negligible, where the probability is over all random coins used in $\text{Gen}, \text{Enc}, \text{Dec}, \text{Eval}$.

- *Circuit Privacy: The ciphertext outputted by* Eval *is statistically indistinguishable from the encrypted plaintext circuit output*

$$\{\mathcal{C}'(c)\} \overset{\text{S}}{\approx} \{\text{Enc}_{pk}(\mathcal{C}(m))\}$$

We assume that ciphertexts from FHE reveal plaintext input lengths.

3.1 Class 1.a/1.c/1.e Construction

First, we will construct a protocol for class 1.c. Both Class 1.a and Class 1.e can be reduced to this case.

The intuition is simple - we will have P_2 do all the computation, using FHE to make sure P_2 never sends their input to P_1, not even in an encrypted or encoded form.

- P_1 samples $(pk, sk) \leftarrow \text{Gen}(1^k)$, compute $c_x \leftarrow \text{Enc}(pk, x)$, and sends pk, c_x to P_2. Note that $\text{Enc}(pk, x)$ leaks $|x|$, but in this class P_2 learns $|x|$ anyways.
- P_2 uses pk to encrypt $c_y \leftarrow \text{Enc}(pk, y)$, gets $f' \leftarrow \text{Eval}_{pk}(f)$, runs $f'(c_x, c_y)$ to get m', and sends m' to P_1. This leaks $|f(x, y)|$ to P_2, but P_2 is allowed to learn this.
- P_1 decrypts the received ciphertext to get $f(x, y)$.

There is one subtlety here. P_2 needs to construct the circuit for f to pass to FHE. To construct the correct number of output wires, P_1 needs to know $|f(x, y)|$ before evaluation.

To explain why this is necessary for security, consider the set union problem. By this point in the protocol, P_2 knows $|x|$ and y. With this, P_2 can compute an upper bound on output length. In this case, the upper bound is $|x| + |y|$. P_2 could construct a circuit with $|x| + |y|$ outputs, padding the true output to that length, but by sending this circuit P_2 reveals the upper bound $|x| + |y|$ to P_1. P_1 can then compute $|y|$.

Thus, even computing an upper bound may reveal extra information. To be fully secure, we need to run computation in two stages. First, we run a size preamble. This is a computation just for computing $|f(x, y)|$. Using that length, we then compute $f(x, y)$.

- P_1 samples $(pk, sk) \leftarrow \text{Gen}(1^k)$, compute $c_x \leftarrow \text{Enc}(pk, x)$, and sends pk, c_x to P_2.
- P_2 constructs $\mathcal{C}_{size,y}$, defined as $\mathcal{C}_{size,y}(x) = |f(x, y)|$. P_2 gets $\mathcal{C}'_{size} \leftarrow \text{Eval}\,pk(\mathcal{C}_{size,y})$, computes $\mathcal{C}'_{size}(c_x)$, and sends the encrypted length c_ℓ.

- P_1 decrypts c_ℓ and sends back ℓ.
- P_2 constructs $\mathcal{C}_{f,y}$ such that $\mathcal{C}_{f,y}(x) = f(x,y)$ and $\mathcal{C}_{f,y}$ has ℓ output wires. P_2 can now compute as before on c_x to get c_f, sending c_f back.
- P_1 decrypts c_f to get the output.

The simulator constructions are fairly straightforward, and will be skipped. The security comes from the CPA encryption scheme and the security we assume FHE supplies.

Briefly, we explain how this can be modified for other cases. If both parties should learn output (class 1.a), P_1 can send $f(x,y)$ at the end of a Class 1.c protocol. If P_2 should learn $f(x,y)$ and P_1 should learn the length (class 1.e), have P_2 mask the output by securely computing $f(x,y) \oplus G(r)$ for some PRG G and some r sampled by P_2, and compute $f(x,y) \oplus G(r)$ in a Class 1.a setting.

4 Impossibility for Class 1.b

Recall in Class 1.b that P_2 learns the other party's input size, but P_1 learns the output $f(x,y)$, and P_2 does not learn output length.

If there is a public upper bound on output length, we could pad $f(x,y)$ to that upper bound, then compute the padded f with a Class 1.c protocol. The impossibility construction will therefore require a function with unbounded output length. We use oblivious transfer. Here, P_1 supplies bit b and P_2 supplies y_b, where both y_0, y_1 are of arbitrary length.

Proof Ideas: First upper bound transcript length, then show a secure protocol would let us compress random strings, a contradiction.

Define $T(\kappa, x, Y)$ as the random variable for bits transmitted when running OT on $x, (y_0, y_1)$, where randomness is over the random tapes.

Since the protocol is polytime, $T(\kappa, 0, (0,0)) < \kappa^c$ for some c, where b is some bit. Let y_1 be a random string of length $m = \Omega(\kappa^{2c})$.

We know $T(\kappa, 0, (0,0)) < \kappa^c$. If

$$\Pr[T(\kappa, 0, (0, y_1')) \geq \kappa^c]$$

is non-negligible, P_1 could distinguish P_2's input by checking transcript length.

This gives $T(\kappa, 0, (0, y_1')) \leq \kappa^c$ almost always. By a similar argument, $T(\kappa, 1, (0, y_1')) \leq \kappa^c$, or else P_2 could check transcript length to attack P_1.

Thus we can compress a random string from κ^{2c} bits to $\Omega(\kappa^c)$ bits, a contradiction.

5 Class 2 Results

In Class 2, all input sizes are hidden.

- Class 2.a: both parties learn $f(x,y)$
- Class 2.b: only P_1 learns $f(x,y)$
- Class 2.c: only P_1 learns $f(x,y)$, P_2 learns $|f(x,y)|$

Note Class 2.b $\subset$ Class 2.c $\subset$ Class 2.a

5.1 General Impossibility

Every one of these cases cannot be computed in general. The proof is based on the communication complexity of the function. In the typical case, the communication complexity is defined assuming $|X| = |y| = n$. For our use case, we use a more general definition.

Definition 3 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be some function. Function f has randomized communication complexity $g(n)$ if for any x, y, Alice and Bob must exchange $g(n)$ bits to compute $f(x, y)$ with negligible error, where $n = \min(|x|, |y|)$. (Only one party needs to learn $f(x, y)$, and the protocol can be insecure.)*

Theorem 1 *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ be a function. If f has randomized communication complexity $\Omega(n^\epsilon)$ for some constant $\epsilon > 0$, f cannot be securely computed with Class 2 hiding.*

Intuition: The communication complexity places a lower bound on the length of the transcript. Since the comm. complexity is $\Omega(n^\epsilon)$, the lower bound is dependent on input length. However, the input lengths are never revealed, and the output is a single bit. We show there is not enough information to have secure protocols with sufficiently long transcripts.

Proof: Assume for contradiction that π is a protocol that securely computes f in class 2.a.

First, we show the communication complexity is upper bounded by $p(\kappa)$ for some polynomial $p(\cdot)$.

Let $b \in \{0,1\}$ be an output bit. Define

$$I_b = \{(x, y) : f(x, y) = b\}$$

By security definitions, there exist simulators S_1, S_2 such that S_1 generates P_1's view from $x, f(x, y)$ and S_2 generates P_2's view from $y, f(x, y)$. For every pair $(x, y) \in I_b$, the simulators must simulate from only (x, b) and (y, b) respectively.

Let x be the shortest string such that for some y, $(x, y) \in I_b$. The runtime of S_1 is bounded by some polynomial $p'(\cdot)$.

Let $p_b(\kappa) = p'(|(x, b)| + \kappa)$. This upper bounds the transcript for a specific (x, y); we now show this upper bounds transcript length for *every* $(x, y) \in I_b$. Suppose $(x, y) \in I_b$ produces a longer transcript with non-negligible probability. Then we have a distinguisher between different (x, y) pairs that produce the same bit b, contradicting security.

Thus, some polynomial $p(\cdot)$ upper bounds communication. Let c be a constant such that $p(\kappa) < \kappa^c$ for sufficiently large κ. Define protocol π' as follows: given input (x, y), let $n = \min(|x|, |y|)$. Run π with security parameter $\kappa = n^{\epsilon/(2c)}$. The output is correct, but the transcript length of π' is at most $p(\kappa) < \kappa^c = n^{\epsilon/2}$, contradicting the $\Omega(n^\epsilon)$ lower bound. ■

Examples of Failing Functions:

Previous work on communication complexity gives several examples of functions that require revealing at least one party's input length [3].

- The inner product function defined as

$$IP(x, y) = \sum_{i=1}^{\min(|x|,|y|)} x_i y_i \mod 2$$

  has communication complexity $\Omega(n)$

- The set disjointness function defined as

$$DISJ(X,Y) = \begin{cases} 1 & \text{if } X \cap Y = \emptyset \\ 0 & \text{otherwise} \end{cases}$$

  has communication complexity $\Omega(n)$.

Both of these functionalities cannot be securely computed while hiding input size.

5.2 Computable Class 2 Functions

The proof earlier shows that when communication complexity grows with input length, we must reveal input length to compute it. This motivates the study of *size-independent* protocols.

Definition 4 *Let* $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$. *A protocol* π *is size-independent if it satisfies the following.*

- *(Correctness)* $\Pr[\pi(x,y) \neq f(x,y)]$ *is negligible.*
- *(Computational Efficiency)* π *can be implemented by two PPT interactive Turing Machines* π_1, π_2.
- *(Communication Efficiency) There exists a polynomial* $p(\cdot)$ *such that when* $|x| = \text{poly}(\kappa)$ *and* $|y| = \text{poly}(\kappa)$, *for sufficiently large* κ *there are at most* $p(\kappa)$ *rounds and the message per round has length at most* $p(\kappa)$.

Intuitively, the communication can be bounded by just the security parameter. This is securely computable, assuuming an even stronger variant of FHE called thresholded FHE.

Definition 5 *Thresholded fully homomorphic encryption is a FHE scheme that replaces* Gen, Dec *with* $\text{ThrGen}, \text{ThrDec}$.

- ThrGen *outputs* $(pk, (sk_1, sk_2))$, *such that* $sk_1 \oplus sk_2 = sk$. *These shares are then divided between the two parties. The first receives* (pk, sk_1), *the second receives* (pk, sk_2).
- ThrDec *takes* $(c_1, sk_1), (c_2, sk_2)$, *and outputs* $\text{Dec}(c_1, sk_1 \oplus sk_2)$ *if* $c_1 = c_2$

Definition Intuition: Thresholded FHE lets us avoid any one party learning the secret key. Both parties must cooperate to decrypt a ciphertext, so the honest party can choose what cipher texts a corrupted party is allowed to decrypt.

Theorem 2 *Assuming thresholded FHE, every* f *with a size independent protocol* π *can be computed in Class 2.c (where* P_1 *learns* $f(x,y)$ *and* P_2 *learns* $1^{|f(x,y)|}$.*)*

Proof Intuition: From π, construct a collection of circuits. Let

$$\mathcal{C}^i_{P_1,\kappa,x,r}, \mathcal{C}^i_{P_2,\kappa,y,s}$$

be circuits that take the previous $i-1$ messages in the transcript so far and outputs the ith message for P_1, P_2 respectively. Parameters r, s are hard-coded random tapes for P_1 and P_2.

Initially, P_1, P_2 run a secure protocol for ThrGen to receive shares of the secret key. P_1 and P_2 then run an encrypted version of π. For message i, the parties construct $\mathcal{C}^i_{P_1,\kappa,x,r}, \mathcal{C}^i_{P_2,\kappa,y,s}$. They transform them with Eval from FHE, evaluating them on the encrypted transcript so far, and sending the next encrypted message to each other. By Communication Efficiency, we have an upper bound of $(i-1)p(\kappa)$ inputs for circuit i, which lets us do the construction without revealing input length.

Finally, construct circuits for the output. Again, this is a two-step process. The first step computes the output length given the transcript. The second step computes the output. Both circuits are then run through Eval, evaluated on the encrypted transcript, and run a secure protocol computing ThrDec for the final outputs.

6 Conclusion

We give a definition of security that explicitly handles input size, and prove various feasibility results in the semi-honest setting. We show that assuming FHE, all functions are securly computable when one input size is hidden and the output size is revealed. Without revealed output size, there are functions that require revealing information on both parties' input sizes, and it is impossible to securely compute all functions while hiding both input sizes.

7 References

[1] Lindell, Yehuda, Kobbi Nissim, and Claudio Orlandi. "Hiding the input-size in secure two-party computation." Advances in Cryptology-ASIACRYPT 2013. Springer Berlin Heidelberg, 2013. 421-440.

[2] Chase, Melissa, Rafail Ostrovsky, and Ivan Visconti. "Executable Proofs, Input-Size Hiding Secure Computation and a New Ideal World." Advances in Cryptology-EUROCRYPT 2015. Springer Berlin Heidelberg, 2015. 532-560.

[3] Kushilevitz, Eyal. "Communication complexity." Advances in Computers 44 (1997): 331-360.