Efficient Secure Two-Party Exponentiation*

Ching-Hua Yu[1], Sherman S.M. Chow[2], Kai-Min Chung[3], and Feng-Hao Liu[4]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan
[2] Combinatorics and Optimization, University of Waterloo, Ontario, Canada
[3] Department of Computer Science, Cornell University, New York, USA
[4] Department of Computer Science, Brown University, Rhode Island, USA

Abstract. We present a new framework to design secure two-party computation protocols for exponentiation over integers and over Z_Q where Q is a publicly-known prime. Using our framework, we realize efficient protocols in the semi-honest setting. Assuming the base is non-zero, and the exponent is at most $Q/2$ for the Z_Q case, our protocols consist of at most 5 rounds (each party sending 5 messages) and the total communication consists of a small constant number (≤ 18) of encrypted/encoded elements in Z_Q. Without these assumptions, our protocols are still more efficient than a protocol recently proposed by Damgård et al. in TCC 2006 (24 vs. > 114 rounds, $\approx 279\ell + 12t$ for an error rate of 2^{-t} vs. $> 110\ell \log \ell$ secure multiplications, where ℓ is the bit length of the shares).
Our protocols are constructed from different instantiations of our framework with different assumptions (homomorphic encryption or oblivious transfers) to achieve different advantages. Our key idea is to exploit the properties of both additive and multiplicative secret sharing. We also propose efficient transformation protocols between these sharings, which might be of independent interest.

Keywords: two-party exponentiation, additive/multiplicative share

1 Introduction

Secure two-party computation is one of the central topics in cryptography, where two parties Alice and Bob want to jointly compute a function $f(x_A, x_B)$ from their own secret inputs x_A and x_B without revealing any information about their inputs. General feasibility results have been developed in the 1980s [1–4], which the privacy of the honest party holds even if the other party deviates arbitrarily from the prescribed protocol (the malicious setting). However, the communication complexity of the protocols depends on the *boolean circuit* complexity of f, which is considered too inefficient for most practical applications.

In many cases such as privacy-preserving data-mining/statistical learning [5–8] and distributed generation of cryptographic keys [9], the desired functionality

* A major part of the work was done while the second author was at New York University and the third author was at Harvard University. The third author is supported by US-Israel BSF grant 2006060 and NSF grant CNS-0831289.

f involves mostly arithmetic operations such as addition, multiplication, division, and exponentiation over the integers and/or the finite fields. More efficient protocols for these basic operations can result in an more efficient protocol for f. Indeed, a significant effort has focused on designing protocols for these operations. For examples, Ishai, Prabhakaran, and Sahai [10, 11] studied general solutions for secure arithmetic computations over rings, which correspond to addition/subtraction and multiplication. Bunn and Ostrovsky [6] designed a division protocol, which is the essential part of their k-means clustering protocol. Damgård *et al.* [12] studied the exponentiation operation over integers modulo a public prime Q and gave the only constant-round protocol in this setting.

Information theoretical security is impossible for two-party computation in the plain model even if we only consider security against "semi-honest" adversaries [13, 3]. Hence, sending certain encrypted/encoded messages is necessary. For addition and multiplication, we know secure protocols with a small constant number of rounds involving a constant number of encrypted elements. However, for integer division and exponentiation, no efficient protocol that sends only a constant number of encrypted elements is known. (This is possible using a fully homomorphic encryption scheme [14], but the current candidates [14, 15] are very inefficient.) Indeed, there is no constant round protocol for division, and the best known protocol for exponentiation [12] requires a large constant number of rounds (> 114) and more than $110\ell \log \ell$ secure multiplications[5], where ℓ is the bit-length of the inputs, which are expensive for some applications.

Problem Statement. Motivated by building more efficient protocols for basic operations, we study semi-honest secure exponentiation over different domains with the goal of achieving efficiency comparable to multiplication. We remark that our protocols can be made secure against malicious adversaries by standard but rather expensive techniques such as zero-knowledge proofs. The possibility of an efficient security-strengthening transformation is outside our scope.

Our setting of secure two-party integer exponentiation is as follows. Two parties Alice and Bob receive inputs as secret shares of integers x and y, and the goal is to jointly compute a secret share of x^y. As usual, a predefined upper bound M on the result of computation is necessary.[6] For example, we are guaranteed that $x^y \leq M$. Given M, we can choose a (publicly known) prime Q which is sufficiently large (say $Q > M^2$) and embed the integers into $\mathbb{Z}_Q$. Namely, we associate integers $\{0, 1, 2, \ldots, Q-1\}$ with elements in $\mathbb{Z}_Q$ in a natural way, which induces an ordering on $\mathbb{Z}_Q$. The shares are taken to be additive shares over $\mathbb{Z}_Q$ – the input to Alice and Bob are (x_A, y_A) and (x_B, y_B) respectively such that $x = x_A + x_B \mod Q$ and $y = y_A + y_B \mod Q$, and the output of Alice and Bob is share z_A and z_B respectively such that $z_A + z_B = x^y \mod Q$.

We also consider modular exponentiation over $\mathbb{Z}_Q$, to compute additive shares of $x^y \bmod Q$, from additive shares of x and y over $\mathbb{Z}_Q$. Now, the exponent y can

[5] These come from the bit-decomposition which dominates the complexity [12].

[6] An example is **unsigned int** in C++ which sets $M = 2^{32} - 1$. We need to avoid overflow during the computation.

be any integer in $\{0, 1, \ldots, Q-1\}$, while the exponent for integer exponentiation is restricted to $y \leq M \ll Q$.

Our Results and Techniques. We present a new framework to design semi-honest secure two-party exponentiation protocols for the above two settings. The key idea is to exploit the properties of *both* additive and multiplicative secret sharing. We also propose efficient protocols for transformation between these sharings, which might be of independent interest. Using our framework, the resulting protocols for integer exponentiation (of non-zero base) use at most 5 rounds (each party sends at most 5 messages) and exchange a small constant number of encrypted/encoded elements. For modular exponentiation over $\mathbb{Z}_Q$, the resulting protocols improve the efficiency over the protocol of Damgård *et al.* [12], and achieve essentially the same efficiency as the integer exponentiation setting when the exponent is at most $Q/2$. A summary of our results is given in Table 1.

We present two implementations basing on homomorphic encryption schemes and oblivious transfers (OTs) (the latter uses the noisy encoding technique of Ishai, Prabhakaran, and Sahai [11]). All our protocols share the same framework and hence have similar round complexity. In the following, we elaborate the advantages of our different implementations.

- **Homomorphic encryption**-based approach achieves the best efficiency in terms of communication and computation.
- **Oblivious transfers**-based approach inherits the versatility of OTs [10]. Our protocols can be realized by many different number theoretic assumptions or even reach information-theoretic security with physical assumptions (e.g., binary symmetric channel). Furthermore, OTs can be precomputed [10, 11] which makes the online efficiency better than the encryption-based protocol for certain parameter ranges.

Related Works and Comparison. One can consider secure computation for two-party or multi-party, and there are differences between semi-honest and malicious security. We focus on the semi-honest two-party setting, but our framework can be extended to the multi-party setting (without honest-majority).

There are a variety of settings for exponentiation depending on whether the base x, the exponent y, and the modulus Q are shared or public. For the most general setting where x, y, Q are shared among all parties, existing solution [16] considers (information-theoretic) semi-honest secure multi-party computation with *honest majority*, hence it is not trivial to be adapted to the two-party setting. On the other hand, Damgård *et al.* [12] considered the same setting as us, where x and y are shared and Q is public. They mentioned that their results can be extended to the general setting of shared Q by combining their technique with [16] and [17]. However, they did not explicitly analyze the complexity of their solution in this setting. Their construction is based on the existence of secure multiplication on linear shares over $\mathbb{Z}_Q$, so it works for both multi-party with honest majority and two-party. A simpler setting where only y is private, x and Q are both public, has been considered [18, 5].

We summarize our results and the related existing results [12, 16] in Table 1. The round complexity counts the maximum number of messages Alice or Bob sends, and the communication denotes the total communication complexity of the protocol, which is the total umber of "unit messages" (i.e., ciphertexts/noisy encodings/field elements) sent by the two parties. A "ciphertext" refers to that produced by ElGamal or Paillier encryption schemes. On the other hand, one noisy encoding requires sending roughly $O(k+\ell)$ field elements (i.e., $O(\ell(k+\ell))$ bits) and uses $O(k+\ell)$ calls to OT, where k is a security parameter for a security level of 2^k and $\ell = \log Q$ is the length of elements of the field $\mathbb{Z}_Q$. In practice, depending on the size of field $\mathbb{Z}_Q$, the length of ciphertext k_{Enc} can be larger or smaller than the length of noisy encodings $O(\ell(k+\ell))$. Two-party secure multiplication can also be implemented in different ways as secure exponentiation, which require 4 "unit messages". Our results and the result of Damgård *et al.* [12] are the only constant-round protocols for secure exponentiation (when both x, y are shared), and both results work for integer exponentiation and modular exponentiation modulo a prime[7] Q. The result of Algesheimer, Camenisch, and Shoup [16] is for multi-party with honest majority.

We remark that our results for the general case of modular exponentiation over $\mathbb{Z}_Q$ require a zero-test and a comparison sub-protocol. Both of them are quite expensive in comparison with the rest of our protocol, and in fact, dominate the complexity of our protocols. In particular, the zero-test protocol due to Nishide and Ohta [19] requires 4 rounds and $12t$ secure multiplications to achieve an error rate 2^{-t}. Their comparison protocol [19] requires 15 rounds and $279\ell + 5$ secure multiplications which is the only reason that makes the number of communication elements of our protocol depending on the field size.

2 Our Framework

Our framework exploits the properties of *both* additive and multiplicative secret sharing. A key observation is that exponentiation is very simple when the base x is shared in multiplicative form and the exponent y is shared in additive form (over $\mathbb{Z}_{Q-1}$). All we need is to convert the shares of x and y into the desired form. We formalize our framework in terms of the protocols as described in Figure 1:

1. Alice and Bob convert their additive shares of ($x = x_A + x_B \mod Q$) to multiplicative shares ($x = x'_A \cdot x'_B \mod Q$). This corresponds to the protocol $\mathsf{A2M}$ (additive sharing to multiplicative sharing) that converts shares from the additive form to the multiplicative form.
2. Alice and Bob convert their additive shares of ($y = y_A + y_B \mod Q$) to additive shares ($y = y'_A + y'_B \mod (Q-1)$) with $y'_A, y'_B \in \mathbb{Z}_{Q-1}$. This corresponds to the modular reduction protocol ModRed, which converts additive shares of y over $\mathbb{Z}_Q$ to $\mathbb{Z}_{Q-1}$.

[7] Our results extend to a general modulus N if $\varphi(N)$ is available.

Setting	Protocol	Rounds	Communication
Integer Exp.	Sec. 3	5	18 ciphertexts = $18k_{\mathsf{Enc}}$ bits
$x \neq 0$	Sec. 4	3	10 noisy encodings = $O(\ell \cdot (\ell + k))$ bits
Integer Exp.	Extending	Above	Above plus a zero-test
arbitrary x	above	plus 4	$(+12t$ secure multiplications)
Modular Exp. over $\mathbb{Z}_Q$	Sec. 3	5	18 ciphertexts = $18k_{\mathsf{Enc}}$ bits
$x \neq 0$ and $y \leq Q/2$	Sec. 4	3	10 noisy encodings = $O(\ell \cdot (\ell + k))$ bits
Modular Exp.	Extending	Above	Above plus a zero-test and a comparison
over $\mathbb{Z}_Q$ for	above	plus 19	$(+12t + 279\ell + 5$ secure multiplications)
general case	[12]	> 114	$> 110\ell \log \ell$ secure multiplications
Modular Exp. over	[12]	$O(1)$	$O(\ell \log \ell)$ secure mult. (large constants)
a shared secret	[16]	$O(\ell)$	$O(\ell^2)$ bits (with a large constant)

Table 1. A summary of our results and existing protocols [12, 16] for computing additive shares of $x^y \mod Q$ from additive shares of x and $y \mod Q$. ($\ell = \log Q$ is the length of elements of the field $\mathbb{Z}_Q$; k denotes the security parameter to achieve security 2^k; t denotes the correctness parameter to achieve an error rate 2^{-t}; k_{Enc} denotes the ciphertext length of the ElGamal or Paillier encryption schemes. We consider 1 secure multiplication requires 4 ciphertexts/noisy encodings/field elements, depending on the implementation. Modular Exp. over $\mathbb{Z}_Q$ in Sec. 3 is only for a safe prime Q.)

3. Alice and Bob jointly compute multiplicative shares of $z' = ((x'_A)^{y'_B} \cdot (x_B)'^{y'_A}) = z'_A \cdot z'_B \mod Q$. This uses the protocol SP ("scalar product"[8]), which computes multiplicative shares of $(x'_A)^{y'_B} \cdot (x'_B)^{y'_A}$. We have

$$x^y = (x'_A)^{(y'_A+y'_B)}(x'_B)^{(y'_A+y'_B)} = (x_A'^{y'_A}) \cdot ((x'_A)^{y'_B} \cdot (x'_B)^{y'_A}) \cdot (x_B'^{y'_B}) \pmod{Q}$$

   by the identity $a^{Q-1} = 1 \pmod{Q} \ \ \forall a \in \mathbb{Z}_Q^*$. The terms $x_A'^{y'_A}$ and $x_B'^{y'_B}$ can be computed locally by Alice and Bob, respectively. As a result, they have multiplicative shares of $x^y = (z'_A \cdot x_A'^{y'_A} \mod Q) \cdot (z'_B \cdot x_B'^{y'_B} \mod Q)$.
4. Alice and Bob locally compute $(z'_A \cdot x_A'^{y'_A} \mod Q)$ and $(z'_B \cdot x_B'^{y'_B} \mod Q)$, and convert the multiplicative shares back to the additive shares. $z_A + z_B = x^y = (z'_A \cdot x_A'^{y'_A}) \cdot (z'_B \cdot x_B'^{y'_B}) \mod Q$. This step requires the protocol $\mathsf{M2A}$ (multiplicative sharing to additive sharing) that converts shares from the multiplicative form to the additive form.

Our exponentiation protocol is a straightforward composition of the above four protocols. A formal description is given in Figure 2. Note that a secure multiplication over $\mathbb{Z}_Q$ can be composed by two invocations of $\mathsf{M2A}$. Our implementations of $\mathsf{M2A}$ based on homomorphic encryption schemes requires a new idea, which is inspired by the existing integer sharing schemes [16]. In the rest of the paper, we will present implementations of the above steps basing on

[8] It is possible to define an "exponential module" in such a way that the cross term $x_A^{y_B} \cdot x_B^{y_A}$ is the inner product of two module elements. Indeed, this is the reason that we call it the scalar product protocol.

A2M Protocol, denoted as $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{A2M}}$.

- **Inputs:** Alice holds $x_A \in \mathbb{Z}_Q$, and Bob holds $x_B \in \mathbb{Z}_Q$, where $x = x_A + x_B \in \mathbb{Z}_Q^*$.
- **Outputs:** Alice obtains z_A, and Bob obtains z_B such that $z_A \cdot z_B = x$.

Modular Reduction Protocol, denoted as $(z_A, z_B) \leftarrow (A(x_A), B(x_B); Q)_{\mathsf{ModRed}}$.

- **Inputs:** Alice holds $x_A \in \mathbb{Z}_Q$, and Bob holds $x_B \in \mathbb{Z}_Q$, and $x = x_A + x_B \in \mathbb{Z}_Q$.
- **Outputs:** Alice obtains $z_A \in \mathbb{Z}_{Q-1}$, and Bob obtains $z_B \in \mathbb{Z}_{Q-1}$ such that $z_A + z_B = x \in \mathbb{Z}_{Q-1}$.

Scalar Product Protocol, denoted as $(z_A, z_B) \leftarrow (A(x_A, y_A), B(x_B, y_B))_{\mathsf{SP}}$.

- **Inputs:** Alice holds $x_A \in \mathbb{Z}_Q$, $y_A \in \mathbb{Z}_{Q-1}$ and Bob holds $x_B \in \mathbb{Z}_Q$, $y_B \in \mathbb{Z}_{Q-1}$, where $x_A \cdot x_B \in \mathbb{Z}_Q^*$.
- **Outputs:** Alice obtains z_A, and Bob obtains z_B such that $z_A \cdot z_B = x_A^{y_B} \cdot x_B^{y_A}$.

M2A Protocol, denoted as $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{M2A}}$.

- **Inputs:** Alice holds $x_A \in \mathbb{Z}_Q^*$, and Bob holds $x_B \in \mathbb{Z}_Q^*$, where $x = x_A \cdot x_B \in \mathbb{Z}_Q^*$.
- **Outputs:** Alice obtains z_A, and Bob obtains z_B such that $z_A + z_B = x$.

Fig. 1. The interfaces for Protocol A2M, ModRed, SP, and M2A

different assumptions. However, there are two subtleties to be addressed in the next two paragraphs.

Non-Zero Base in Multiplicative Shares. Our first step does not make sense when $x = 0$ since x has no multiplicative shares. This is also an "exception case" of the protocol of Damgård *et al.* [12]. To handle this, they use a trick to make x always non-zero, which can also be applied to our protocol.[9] However, it costs an additional equality-test protocol, which is relatively cheap when compared with their protocol, but is more expensive than the complexity of our protocol. Hence, we suggest avoiding using that when it can be safely assumed x is non-zero.

Modular Reduction. The second issue is to construct an efficient ModRed protocol. In cases where y is already given as shares in $\mathbb{Z}_{Q-1}$ (e.g., when y is randomly generated by two parties), we do not need to do modular reduction at all. When

[9] Define $x' = x + (x \stackrel{?}{=} 0)$, where $(x \stackrel{?}{=} 0)$ is 1 if $x = 0$, and is 0 otherwise. With a secure equality-test protocol that computes additive shares of $(x \stackrel{?}{=} 0)$, we can use the identity $x^y = x'^y - (x \stackrel{?}{=} 0)$ to compute x^y avoiding the base being zero [12].

Exponentiation Protocol:

- **Inputs:** Alice holds $x_A, y_A \in \mathbb{Z}_Q$, Bob holds $x_B, y_B \in \mathbb{Z}_Q$, where $x = x_A + x_B \in \mathbb{Z}_Q^*$, $y = y_A + y_B \in \mathbb{Z}_Q$.
- **Outputs:** Alice obtains $z_A \in \mathbb{Z}_Q$, and Bob obtains $z_B \in \mathbb{Z}_Q$ such that $z_A + z_B = x^y$.

1. Alice and Bob run $(x'_A, x'_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{A2M}}$.
2. Run $(y'_A, y'_B) \leftarrow (A(y_A), B(y_B); Q)_{\mathsf{ModRed}}$.
3. Run $(z'_A, z'_B) \leftarrow (A(x'_A, y'_A), B(x'_B, y'_B))_{\mathsf{SP}}$.
4. Run $(z_A, z_B) \leftarrow (A(x_A'^{y'_A} \cdot z'_A), B(x_B'^{y'_B} \cdot z'_B))_{\mathsf{M2A}}$.

Fig. 2. The interface of the exponentiation protocol and its implementation

the input x is guaranteed to be less than $Q/2$, we present a simple implementation in Figure 3 using an M2A protocol. Note that for the setting of integer exponentiation, the condition $y \leq Q/2$ holds for free since we are guaranteed that $y \leq M \ll Q$. The security and efficiency of the ModRed protocol follow from those of M2A, and its correctness is proved in the following lemma.

Lemma 1. *For a prime $Q \in \mathbb{Z}$, $x_A, x_B \in \mathbb{Z}_Q$, and $x = x_A + x_B \in \mathbb{Z}_Q$. Suppose $x \leq Q/2$, then $(z_A, z_B) \leftarrow (A(x_A), B(x_B); Q)_{\mathsf{ModRed}}$ form shares of x in $\mathbb{Z}_{Q-1}$.*

Proof. From the construction, it is clear that $z_A, z_B \in \mathbb{Z}_{Q-1}$. From the property of M2A protocol, we know that $z'_A + z'_B = b_A \cdot b_B \mod (Q-1)$. So we know that $z'_A + z'_B = 1 \mod (Q-1)$ if and only if $x_A \leq Q/2$ and $x_B \leq Q/2$. This implies $x = x_A + x_B \leq Q - 1$. Thus, we know that $x = z_A + z_B \mod (Q-1)$.

On the other hand, if $z'_A + z'_B = 0 \mod (Q-1)$, at least one of x_A, x_B is greater than $Q/2$, and thus $2Q > x_A + x_B > Q/2 > x$. This means $x_A + x_B = x + Q$, thus we can see $z_A + z_B = (x_A + x_B - Q) \mod (Q-1) = x$ (as $x \leq Q/2$).

Note that in Step 3, Alice and Bob require to run M2A protocol over $\mathbb{Z}_{Q-1}$ (corresponding to the inputs and outputs of the invocation). This is valid in our scheme even though $Q - 1$ is not a prime. □

For the general case, however, we do not know an efficient protocol with complexity independent of the bit-length $\ell = \log Q$. A natural idea to implement ModRed is to test whether $x_A + x_B \geq Q$ or $x_A + x_B < Q$, and then subtract Q before taking mod $Q - 1$ if it is the former case. However, this idea requires a secure comparison protocol (e.g. [19]), which is expensive.

Semi-honest Security. We informally define the semi-honest security in the two party case and refer the readers to the literature (e.g., [5, 8]) for the standard formal definition. For a deterministic functionality $f(\cdot, \cdot)$, a protocol is said to

Modular Reduction Protocol for a public prime number $Q \in \mathbb{Z}$ (for the case where $x \leq Q/2$):

- **Inputs:** Alice holds $x_A \in \mathbb{Z}_Q$, and Bob holds $x_B \in \mathbb{Z}_Q$, and $x = x_A + x_B \mod Q$ such that $x \leq Q/2$.
- **Outputs:** Alice obtains z_A, and Bob obtains z_B such that $z_A + z_B = x \mod (Q-1)$.

1. Alice locally computes a number b_A such that $b_A = 1 \mod (Q-1)$ iff $x_A \leq Q/2$ otherwise 0.
2. Bob locally computes a number b_B such that $b_B = 1 \mod (Q-1)$ iff $x_B \leq Q/2$ otherwise 0.
3. Run $(z'_A, z'_B) \leftarrow (A(b_A), B(b_B))_{\mathsf{M2A}}$.
4. Alice outputs $z_A = (x_A + z'_A \cdot Q) \mod (Q-1)$, and Bob outputs $z_B = (x_B + (z'_B - 1) \cdot Q) \mod (Q-1)$

Fig. 3. Protocol ModRed

be *semi-honest secure* if for any honest-but-curious adversary $\mathcal{A}$ who corrupts the first party, there exists a probabilistic polynomial time simulator $\mathcal{S}$, who gets the inputs and the randomness of $\mathcal{A}$, can produce a view of $\mathcal{A}$ which is (statistically/computationally) indistinguishable against time 2^k distinguisher with advantage 2^{-k} from the real interaction with the second party, where k is a security parameter. Similar case should hold for the corrupted second party.

3 Implementation Using Homomorphic Encryption

Now we present our implementation of the protocols $\mathsf{M2A}, \mathsf{A2M}, \mathsf{SP}$ using homomorphic encryption schemes. Formal description can be found in Figure 4-6.

Homomorphic Encryption. A homomorphic encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ has both the message space and ciphertext space associated with certain algebraic structure and the encryption function Enc is homomorphic with respect to the corresponding operation in both spaces. There are several efficient public-key homomorphic encryption schemes which possess different homomorphic properties. We will use both the ElGamal encryption scheme [21] and the Paillier encryption scheme [22]. The ElGamal encryption scheme is semantically secure over *the subgroup of quadratic residue* $H \stackrel{\text{def}}{=} \{x^2 : x \in \mathbb{Z}_Q^*\} \subset \mathbb{Z}_Q^*$, when Q is a safe prime (i.e., $Q = 2P + 1$ where P is also a prime) from a common belief that the decisional Diffie-Hellman (DDH) assumption holds for H, and possesses *multiplicative homomorphism*. On the other hand, the Paillier encryption scheme

is semantically secure over $\mathbb{Z}_N$ for a composite number $N = PQ$ under the decisional composite residuosity assumption, and possesses *additive homomorphism*.

We assume that the encryption schemes with security parameter k is semantic secure against time 2^k adversary with advantage 2^{-k}, and our protocol achieve semi-honest security against time $O(2^k/T)$ distinguisher with advantage $O(2^{-k})$, where T is the run-time of the protocol. To achieve this, we also require that the modulo N of the Paillier encryption scheme satisfying $N \geq 20 \cdot 2^k Q^2$. This is a mild requirement[10] and can be satisfied by using a larger security parameter.

Our Implementations. We first observe that if we have additive (resp., multiplicative) homomorphic encryption schemes over $\mathbb{Z}_Q$ (resp., $\mathbb{Z}_Q^*$), then secure $\mathsf{A2M}, \mathsf{M2A}$ (resp., SP) protocols are very easy to achieve — we can let Alice send encryption of her input to Bob, who can then perform computation homomorphically, and send back an encrypted share to Alice. Intuitively, Bob can learn nothing since he only receives encrypted messages from Alice; and Alice also learns nothing, since she only receives a share of the computed value from Bob.

Unfortunately, the Paillier encryption scheme is only semantically secure over $\mathbb{Z}_N$ for a composite number N, and the ElGamal encryption scheme is semantically secure over the subgroup of quadratic residue in $\mathbb{Z}_Q^*$ when Q is a safe prime. These make the implementation of $\mathsf{A2M}, \mathsf{M2A}$ and SP protocols non-trivial. At a high level, we overcome these difficulties with the following ideas.

- To implement $\mathsf{A2M}$ and $\mathsf{M2A}$ using the Paillier encryption scheme over $\mathbb{Z}_N$, we exploit an idea inspired by the integer sharing schemes of Algesheimer, Camenisch, and Shoup [16]. Briefly, instead of using secret sharing to hide the secret x (which can only be done for additively homomorphism over $\mathbb{Z}_Q$), we require $N \gg Q$ and use a random noise to *statistically* hide the secret.
- Implementing SP using the ElGamal encryption scheme over $H \subset \mathbb{Z}_Q^*$ is trickier. Very briefly, note that $\mathbb{Z}_Q^* = B \times H$, where B is the binary subgroup of Legendre symbols, our idea is to handle the H and B parts of $\mathbb{Z}_Q^*$ separately, where the H part is handled by ElGamal, and the B part is handled by two calls to $\mathsf{A2M}$ and $\mathsf{M2A}$ protocols.

Our implementation of the three protocols can be found in Figure 4–6. We proceed to explain the details of our implementation as follows.

- $\mathsf{M2A}$ protocol (Figure 4): Alice sends her encrypted input $\hat{x}_A = \mathsf{Enc}(x_A)$ to Bob, who can homomorphically compute encrypted secret $\hat{x} = \mathsf{Enc}(x_A \cdot x_B) = \mathsf{Enc}(x_A)^{x_B}$ of x using the additively homomorphic property of Enc. Bob then wants to split the secret x into additive shares, so he selects a random $u \in \mathbb{Z}_Q$ and computes encrypted share $\mathsf{Enc}(x+u) = \mathsf{Enc}(x) \cdot \mathsf{Enc}(u)$ and his share $-u$. However, Paillier encryption is additively homomorphic over $\mathbb{Z}_N$ with $N \gg Q$, the resulting $x+u$ is a number between 0 and Q^2+Q and Bob cannot send $\mathsf{Enc}(x+u)$ back to Alice directly (since it leaks partial information). Hence, Bob uses a large random noise w (say, in $[-N/10, N/10]$) to hide the

[10] It is satisfied automatically unless $\log Q \gg k$.

$\mathsf{M2A}$ Protocol $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{M2A}}$

1. Alice generates a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, and sends pk and $\hat{x}_A = \mathsf{Enc}_{\mathsf{pk}}(x_A)$ to Bob. (Recall that we require $N \geq 20 \cdot 2^k Q^2$.)
2. Bob samples uniformly random $u \leftarrow \mathbb{Z}_Q$ and $w \leftarrow [-N/10, N/10]$, computes $\hat{v} = \mathsf{Enc}_{\mathsf{pk}}(w + u + x_A \cdot x_B)$ and sends $\hat{v}$ and $v' = (w \mod Q)$ to Alice.
3. Alice outputs $z_A = (\mathsf{Dec}_{\mathsf{sk}}(\hat{v}) - v' \mod Q) = (u + (x_A \cdot x_B) \mod Q)$.
4. Bob outputs $z_B = -u$.

Fig. 4. Implementation of $\mathsf{M2A}$ using (additively) homomorphic encryption

secret x and sends $\mathsf{Enc}(w+u+x)$ to Alice. On the other hand, to help Alice to find out $x+u \mod Q$, Bob also sends $w \mod Q$ to Alice, who can then recover $x+u \mod Q$. Note that the noise hides $x+u$ statistically.

- $\mathsf{A2M}$ protocol (Figure 5): Its idea and structure are similar to those of $\mathsf{M2A}$.
- SP protocol (Figure 6): The SP protocol involves three parts: 1) converting the messages from $\mathbb{Z}_Q^*$ into H $(x_A, x_B \to x'_A, x'_B)$ accompanied parity bits $(z_{A,1}, z_{B,1}, z_{A,2}, z_{B,2})$ in Step 1-2, 2) computing (an encrypted share of) $x'^{y_B}_A x'^{y_A}_B$ in Step 3-8, and 3) recovering the messages from H to $\mathbb{Z}_Q^*$.
  1) Alice and Bob convert the messages into H, where the DDH assumption is believed to hold, and so ElGamal encryption is semantically secure. Note that in $\mathbb{Z}_Q^*$, exactly one of x_A and $-x_A$ is a quadratic residue, and we denote such one in H as m. From this we can calculate $x_A^{y_B}$ (in $\mathbb{Z}_Q^*$) by first calculating m^{y_B} in H and then multiplying the outcome with a number $b \in \{1, -1\}$ depending on $m = x_A$ or $m = -x_A$ and the parity of y_B. To formalize this, we use two boolean variables ℓ_A, t_B and set $\ell_A = 1$ in the case where x_A is a quadratic residue and $t_B = 1$ if y_B is odd. When $\ell_A = 1$ and $t_B = 1$, we know $x_A^{y_B} = -1 \cdot m^{y_B}$; and for all the other cases, we have $x_A^{y_B} = m^{y_B}$. This is equivalent to compute $(1 - 2 \cdot \ell_A \cdot t_B) \cdot m^{y_B}$, where additive shares of $(1 - 2 \cdot \ell_A \cdot t_B)$ can be computed using one $\mathsf{M2A}$ and $\mathsf{A2M}$. That is, we can think of Alice holding $w_A = 2\ell_A$, and Bob holding $w_B = t_B$, and view them as multiplicative shares of $w = w_A \cdot w_B$. We can then apply $\mathsf{M2A}$ to turn them into additive shares $w = w'_A + w'_B$ (which is $2\ell_A \cdot t_B$). Alice and Bob can locally compute $u_A = 1 - w'_A$, and $u_B = -w'_B$, so that $u = u_A + u_B$ is an additive share of $1 - 2\ell_A \cdot t_B$.
  2) Both parties send $\hat{x}_A = \mathsf{Enc}_{\mathsf{pk}_A}(x'_A)$ and $\hat{x}_B = \mathsf{Enc}_{\mathsf{pk}_B}(x'_B)$ to each other. Upon receiving $\hat{x}_A$ and $\hat{x}_B$, they can compute $\mathsf{Enc}_{\mathsf{pk}_A}(x'^{y_B}_A) = \hat{x}_A^{y_B}$ and $\mathsf{Enc}_{\mathsf{pk}_B}(x'^{y_A}_B) = \hat{x}_B^{y_A}$ using the multiplicatively homomorphic property of Enc. To protect their own privacy, they split these values into multiplicative shares, and send each other an encrypted share. For example, Bob splits the encrypted $x'^{y_B}_A$ into $(u_B \cdot x'^{y_B}_A) \cdot (u_B^{-1})$ for a random u_B, and sends an encrypted $(u_B \cdot x'^{y_B}_A)$ to Alice. Finally, they can locally combine their shares of $x'^{y_B}_A$ and $x'^{y_A}_B$ into shares of $x'^{y_B}_A \cdot x'^{y_B}_B$.

3) Both parties combines shares into the final output. From Step 1-2, they have $x_A^{y_B} = (1-2\cdot\ell_A\cdot t_B)\cdot x_A'^{y_B} = z_{A,1}\cdot z_{B,1}\cdot x_A'^{y_B}$ and $x_B^{y_A} = (1-2\cdot\ell_B\cdot t_A)\cdot x_B'^{y_A} = z_{A,2}\cdot z_{B,2}\cdot x_B'^{y_A}$; and from Step 3-8, $x_A'^{y_B}x_B'^{y_A} = z_{A,3}\cdot z_{B,3}$. These lead to $x_A^{y_B}x_B^{y_A} = (z_{A,1}\cdot z_{A,2}\cdot z_{A,3})\cdot(z_{B,1}\cdot z_{B,2}\cdot z_{B,3})$.

$\mathsf{A2M}$ Protocol $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{A2M}}$

1. Alice generates a pair of keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(1^k)$, and sends pk and $\hat{x}_A = \mathsf{Enc}_{\mathsf{pk}}(x_A)$ to Bob. (Recall that we require $N \geq 20\cdot 2^k Q^2$.)
2. Bob samples uniformly at random $u \leftarrow \mathbb{Z}_Q^*$ and $w \leftarrow [-N/10, N/10]$, computes $\hat{v} = \mathsf{Enc}_{\mathsf{pk}}(w + u\cdot(x_A + x_B))$ and sends $(\hat{v}, v')$ to Alice.
3. Alice outputs $z_A = (\mathsf{Dec}_{\mathsf{sk}}(\hat{v}) - v' \mod Q) = (u\cdot(x_A + x_B) \mod Q)$.
4. Bob outputs $z_B = u^{-1}$.

Fig. 5. Implementation of $\mathsf{A2M}$ using (additively) homomorphic encryption

SP Protocol $(z_A, z_B) \leftarrow (A(x_A, y_A), B(x_B, y_B))_{\mathsf{SP}}$
Let $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ be ElGamal encryption over H and $\left(\frac{a}{Q}\right)$ be the Legendre symbol.

1. Let $\ell_A = 1$ if $\left(\frac{x_A}{Q}\right) = -1$, and $\ell_A = 0$ if $\left(\frac{x_A}{Q}\right) = 1$. Alice sets $x'_A = x_A\cdot(-1)^{\ell_A}$. Similarly, let $\ell_B = 1$ if $\left(\frac{x_B}{Q}\right) = -1$ and 0 otherwise. Bob sets $x'_B = x_B\cdot(-1)^{\ell_B}$.
2. Let $t_A = y_A \mod 2$, and $t_B = y_B \mod 2$. Alice and Bob run two secure subprotocols (each with one $\mathsf{M2A}$ and one $\mathsf{A2M}$) to obtain $(z_{A,1}, z_{B,1})$ such that $z_{A,1}\cdot z_{B,1} = 1 - 2\cdot\ell_A\cdot t_B$, and $(z_{A,2}, z_{B,2})$ such that $z_{A,2}\cdot z_{B,2} = 1 - 2\cdot\ell_B\cdot t_A$.
3. Alice generates $(\mathsf{pk}_A, \mathsf{sk}_A) \leftarrow \mathsf{Gen}(1^k)$, and sends pk_A, $\hat{x}_A = \mathsf{Enc}_{\mathsf{pk}_A}(x'_A)$ to Bob.
4. Bob generates $(\mathsf{pk}_B, \mathsf{sk}_B) \leftarrow \mathsf{Gen}(1^k)$, and sends pk_B, $\hat{x}_B = \mathsf{Enc}_{\mathsf{pk}_B}(x'_B)$ to Alice.
5. Alice computes $\hat{u}_A = \mathsf{Enc}_{\mathsf{pk}_B}(u_A)$, and sends $\hat{v}_A = \hat{u}_A\cdot\hat{x}_B^{y_A}$ to Bob, for $u_A \leftarrow Z_Q^*$.
6. Bob computes $\hat{u}_B = \mathsf{Enc}_{\mathsf{pk}_A}(u_B)$, and sends $\hat{v}_B = \hat{u}_B\cdot\hat{x}_A^{y_B}$ to Alice, for $u_B \leftarrow Z_Q^*$.
7. Alice computes $z_{A,3} = u_A^{-1}\cdot\mathsf{Dec}_{\mathsf{sk}_A}(\hat{v}_B) = (u_B\cdot x_A'^{y_B})/u_A$.
8. Bob computes $z_{B,3} = u_B^{-1}\cdot\mathsf{Dec}_{\mathsf{sk}_B}(\hat{v}_A) = (u_A\cdot x_B'^{y_A})/u_B$.
9. Alice outputs $z_A = z_{A,1}\cdot z_{A,2}\cdot z_{A,3}$, and Bob outputs $z_B = z_{B,1}\cdot z_{B,2}\cdot z_{B,3}$.

Fig. 6. Implementation of SP using (multiplicatively) homomorphic encryption

Efficiency. Suppose the key generation has been done in a setup stage, we have:

- M2A protocol: Alice needs to do 1 encryption and 1 decryption. Bob needs to do 1 encryption and 1 exponentiation with exponent $x_B \in Z_Q$. The protocol consists of 1 round and 2 messages, where Alice's message is a ciphertext, and Bob's message is a ciphertext plus an element in $\mathbb{Z}_Q$.
- A2M protocol: Alice needs to do 1 encryption and 1 decryption. Bob needs to do 2 encryptions and 1 exponentiation with exponent $u \in Z_Q$. The protocol consists of 1 round and 2 messages, where Alice's message is a ciphertext, and Bob's message is a ciphertext plus an element in $\mathbb{Z}_Q$.
- SP protocol: To compute $x_A'^{y_B} x_B'^{y_A}$, both parties need to do 2 encryptions, 1 decryption, 1 multiplication, and 1 exponentiation with exponent in Z_Q. They exchange 4 messages, each consists of a ciphertext. In addition, each of $z_{A,1}z_{B,1}$ and $z_{A,2}z_{B,2}$ uses one invocation of M2A and one invocation of A2M. Hence, the total communication consists of $4 + (2 + 2) \cdot 2 = 12$ ciphertexts and 4 field elements, and these takes $1 + (1 + 1) = 3$ rounds (since the computation of $z_{A,1}z_{B,1}$, $z_{A,2}z_{B,2}$ and Step 3-4 can be parallelized).
- EXP protocol (for $x \neq 0$ and $y \leq Q/2$): The total communication consists of $2(\mathsf{A2M}) + 2(\mathsf{ModRed}) + 12(\mathsf{SP}) + 2(\mathsf{M2A}) = 18$ ciphertexts and 7 field elements, and these takes 5 rounds (running A2M and ModRed in parallel.)

For the general case (x might be 0, y might be greater than $Q/2$), we need one more equality test which costs 4 rounds and $12t$ secure multiplications with error probability 2^{-t} [19], and one more comparison protocol for implementing ModRed, which costs 15 rounds and $279\ell + 5$ secure multiplications [19]. We remark that a secure multiplication can be done using 2 invocations of M2A.

For integer exponentiation, we can choose a big enough Q and embed the integers into $\mathbb{Z}_Q$ in which we do all the arithmetic computations. We choose Q to be a big enough *safe prime* and use homomorphic encryption to build efficient secure integer exponentiation as mentioned. However, for realizing modular exponentiation for a general Q, we are not aware of any candidate encryption schemes that can be used in our scalar product protocol. On the other hand, our protocol to be described in the next section works for a general Q.

4 Implementation Using Oblivious Transfer

We use the noisy encoding techniques proposed by Ishai, Prabhakaran and Sahai [11] to implement our new protocols A2M and SP in Figure 7 and 9. We also describe M2A from [11] in Figure 8 for completeness.

We use $OT(m_0, m_1; \sigma)_{A \to B}$ to denote the OT protocol jointly run by Alice, who holds two messages $m^{(0)}, m^{(1)} \in \mathbb{Z}_Q$, and Bob, who holds the selection bit $\sigma \in \{0, 1\}$. Informally, security requires that after the protocol, Alice cannot learn σ, while Bob can only obtain $m^{(\sigma)}$, and receives no further information of $m^{(1-\sigma)}$. Similarly, let $\boldsymbol{v}$ be any vector and $\boldsymbol{v}_i$ be its i-th element. We use $OT(\boldsymbol{m}^{(0)}, \boldsymbol{m}^{(1)}; \boldsymbol{\sigma})_{A \to B}$ to denote the "vector version" of an OT protocol jointly run by Alice, who holds two vectors of messages $\boldsymbol{m}^{(0)}, \boldsymbol{m}^{(1)} \in (\mathbb{Z}_Q)^n$, and Bob, who holds the selection vector $\boldsymbol{\sigma} \in \{0, 1\}^n$ and wants to learn $\boldsymbol{m}_i^{(\sigma_i)}$ for $i \in [1, n]$.

Noisy Encoding. We review the noisy encoding scheme of Ishai, Prabhakaran and Sahai [11], which we are going to use as a building block. Encoding of $x \in \mathbb{Z}_Q$, denoted as $\mathsf{NoisyEnc}_n^{\mathbb{Z}_Q}(x)$ where n is a parameter of the encoding and $\mathbb{Z}_Q$ is the underlying field/ring, is computed by the following randomized procedure:

1. Pick a random bit-vector $\boldsymbol{\sigma} \leftarrow \{0,1\}^n$.
2. Pick a random vector $\boldsymbol{u} \in (\mathbb{Z}_Q)^n$ conditioned on $\sum_{i=1}^n \boldsymbol{u}_i = x$.
3. Pick a random vector pair $(\boldsymbol{v}^0, \boldsymbol{v}^1) \in ((\mathbb{Z}_Q)^n)^2$ conditioned on $\forall i, \boldsymbol{v}_i^{\boldsymbol{\sigma}_i} = \boldsymbol{u}_i$.
4. Output $(\boldsymbol{v}^0, \boldsymbol{v}^1, \boldsymbol{\sigma})$.

The encoding contains two parts: $(\boldsymbol{v}^0, \boldsymbol{v}^1)$ and $\boldsymbol{\sigma}$. It has been proven [11] that with sufficiently large n, the distribution of $(\boldsymbol{v}^0, \boldsymbol{v}^1)$ (without $\boldsymbol{\sigma}$) statistically hides x; while one can decode $(\boldsymbol{v}^0, \boldsymbol{v}^1)$ with $\boldsymbol{\sigma}$ to retrieve x.

Our Implementations. At a high level, in protocols A2M and M2A (Figure 7, 8), Bob computes a noisy encoding $(\boldsymbol{u}^0, \boldsymbol{u}^1, \boldsymbol{\sigma})$ of his input x_B and sends $(\boldsymbol{u}^0, \boldsymbol{u}^1)$ to Alice, which contains the information of x_B but statistically hides it from Alice. Alice can still compute another "re-randomized" encoding of $x_A + x_B$ or $x_A \cdot x_B$ from $(\boldsymbol{u}^0, \boldsymbol{u}^1)$ to protect her privacy, and use OT to let Bob learn the messages according to $\boldsymbol{\sigma}$. This is similar to our solution based on the homomorphic encryption. In protocol SP(Figure 9), both parties need to compute the noisy encoding of their inputs y_A, y_B and send the information to each other since they want the other party to compute $x_B^{y_A}$ and $x_A^{y_B}$ respectively. Similarly, they both do re-randomization and use OT to send the messages back.

A2M Protocol $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{A2M}}$

1. Bob computes $(\boldsymbol{u}^0, \boldsymbol{u}^1, \boldsymbol{\sigma}) \leftarrow \mathsf{NoisyEnc}_n^{\mathbb{Z}_Q}(x_B)$, and sends $(\boldsymbol{u}^0, \boldsymbol{u}^1)$ to Alice.
2. Alice does the following:
   - Pick a random $p \leftarrow \mathbb{Z}_Q^*$.
   - Pick a random vector $\boldsymbol{t} = (t_1, t_2, \ldots, t_n)$ conditioned on $x_A = \sum_{i=1}^n t_i$.
   - Compute $\boldsymbol{w}^0 = p \cdot (\boldsymbol{u}^0 + \boldsymbol{t})$ and $\boldsymbol{w}^1 = p \cdot (\boldsymbol{u}^1 + \boldsymbol{t})$.
   - Send to Bob with $OT((\boldsymbol{w}^0, \boldsymbol{w}^1); \boldsymbol{\sigma})_{A \to B}$.
   - Output $z_A = p^{-1}$.
3. Bob outputs $z_B = \sum_{i=1}^n \boldsymbol{w}_i^{\boldsymbol{\sigma}_i}$.

Fig. 7. Implementation of A2M using OT

Efficiency. We consider the running time of our protocol and the communication complexity in terms of the number of bits and the number of messages being sent. In protocol A2M and M2A, the operations we perform are: (1) multiplication, (2) addition, (3) sampling a random element and (4) oblivious transfer.

M2A Protocol [11] $(z_A, z_B) \leftarrow (A(x_A), B(x_B))_{\mathsf{M2A}}$

1. Bob computes $(\boldsymbol{u}^0, \boldsymbol{u}^1, \boldsymbol{\sigma}) \leftarrow \mathsf{NoisyEnc}_n^{\mathbb{Z}_Q}(x_B)$, and sends $(\boldsymbol{u}^0, \boldsymbol{u}^1)$ to Alice.
2. Alice does the following:
   - Pick a random $\boldsymbol{t} \leftarrow (\mathbb{Z}_Q)^n$.
   - Compute $\boldsymbol{w}^0 = \boldsymbol{u}^0 \cdot x_A + \boldsymbol{t}$ and $\boldsymbol{w}^1 = \boldsymbol{u}^1 \cdot x_A + \boldsymbol{t}$.
   - Send to Bob with $OT((\boldsymbol{w}^0, \boldsymbol{w}^1); \boldsymbol{\sigma})_{A \to B}$.
   - Output $z_A = -\sum_{i=1}^n \boldsymbol{t}_i$.
3. Bob outputs $z_B = \sum_{i=1}^n \boldsymbol{w}_i^{\boldsymbol{\sigma}_i}$.

Fig. 8. Implementation of M2A using OT

SP Protocol $(z_A, z_B) \leftarrow (A(x_A, y_A), B(x_B, y_B))_{\mathsf{SP}}$

1. Alice computes $(\boldsymbol{u}_A^0, \boldsymbol{u}_A^1, \boldsymbol{\sigma}_A) \leftarrow \mathsf{NoisyEnc}_n^{\mathbb{Z}_Q}(y_A)$, and sends $(\boldsymbol{u}_A^0, \boldsymbol{u}_A^1)$ to Bob.
2. Bob computes $(\boldsymbol{u}_B^0, \boldsymbol{u}_B^1, \boldsymbol{\sigma}_B) \leftarrow \mathsf{NoisyEnc}_n^{\mathbb{Z}_Q}(y_B)$, and sends $(\boldsymbol{u}_B^0, \boldsymbol{u}_B^1)$ to Alice.
3. Alice picks a random $\boldsymbol{t}_A \leftarrow (\mathbb{Z}_Q^*)^n$, computes $\boldsymbol{w}_{A,i}^0 = x_A^{\boldsymbol{u}_{B,i}^0} \cdot \boldsymbol{t}_{A,i}$, $\boldsymbol{w}_{A,i}^1 = x_A^{\boldsymbol{u}_{B,i}^1} \cdot \boldsymbol{t}_{A,i}$, and sends to Bob with $OT((\boldsymbol{w}_A^0, \boldsymbol{w}_A^1); \boldsymbol{\sigma}_B)_{A \to B}$.
4. Bob picks a random $\boldsymbol{t}_B \leftarrow (\mathbb{Z}_Q^*)^n$. compute $\boldsymbol{w}_{B,i}^0 = x_B^{\boldsymbol{u}_{A,i}^0} \cdot \boldsymbol{t}_{B,i}$, $\boldsymbol{w}_{B,i}^1 = x_B^{\boldsymbol{u}_{A,i}^1} \cdot \boldsymbol{t}_{B,i}$. and sends to Alice with $OT((\boldsymbol{w}_B^0, \boldsymbol{w}_B^1); \boldsymbol{\sigma}_A)_{B \to A}$.
5. Alice outputs $z_A = \prod_{i=1}^n \boldsymbol{w}_{B,i}^{\boldsymbol{\sigma}_{A,i}} / \prod_{i=1}^n \boldsymbol{t}_{A,i}$.
6. Bob outputs $z_B = \prod_{i=1}^n \boldsymbol{w}_{A,i}^{\boldsymbol{\sigma}_{B,i}} / \prod_{i=1}^n \boldsymbol{t}_{B,i}$.

Fig. 9. Implementation of SP using OT

Among all these operations, multiplication and oblivious transfer are the most expensive, so when we measure the efficiency, we count the number of these two operations. In protocol SP, we need another operation – exponentiation in $\mathbb{Z}_Q$, which is much more expensive than multiplication. Thus, we count the number of exponentiations and OTs in protocol SP. In the vector version of OT, the sender sends $2n$ elements in $\mathbb{Z}_Q$ and the receiver learns n of them. We consider this as n operations of basic OT in which the sender sends two elements in $\mathbb{Z}_Q$ and the receiver learns one of them. Under this measurement standard:

- M2A and A2M protocols: Alice performs $2n$ multiplications and n OTs. Bob only performs additions and sampling random elements. The protocol exchanges 2 messages: one with $2n$ elements in $\mathbb{Z}_Q$, and the other with n OTs.
- SP protocol: Both parties perform $2n$ exponentiations and n OTs, involving 4 message exchanges: two with $2n$ elements in $\mathbb{Z}_Q$, and two with n OTs.
- EXP protocol (for $x \neq 0$ and $y \leq Q/2$): The protocol consists of the four protocols (also protocol ModRed, which requires 1 call of M2A). With paral-

lelization, the round complexity just needs 5 message exchanges. The total communication complexity is $10n$ elements in $\mathbb{Z}_Q$, and $5n$ OTs.

For the parameter setting, we need to set $n = O(k + \log Q)$ to achieve a $2^{-\Omega(k)}$-security, where it is sufficient to take the hidden constant as 3. For online OT, we need to send 2 elements in $\mathbb{Z}_Q$ per OT, and thus the communication complexity is $10n + 2 \cdot 5n = 20n = 60k + 60 \log Q$ elements in $\mathbb{Z}_Q$. Due to the lack of space, security analysis of our protocols are deferred to the full version of this paper.

For the general case (x might be 0, and y might be greater than $Q/2$), as before, we need one more equality test in the beginning, which costs 4 rounds and $12t$ secure multiplications with error probability 2^{-t} [19], and one more comparison protocol for implementing ModRed, which costs 15 rounds and $279\ell+5$ secure multiplications [19]. Recall that we can perform a secure multiplication using 2 invocations of the M2A protocol.

5 Conclusion and Future Directions

In this paper, we propose a new framework to efficiently compute exponentiation when both the base and the exponent are shared among different parties. The goal is to build constant-round protocols with communication cost comparable to that of a secure multiplication. Instead of using bit-decomposition, we utilize the inter-conversion of additive and multiplicative sharing and "scalar product" over an "exponential module" (A2M, M2A, and SP).

We implemented A2M, M2A and SP in two ways – homomorphic encryption, and oblivious transfer (OT). We use both an additive homomorphic encryption and a multiplicative homomorphic encryption (but not a fully homomorphic one) to achieve efficiency. Our OT-based solution uses the noisy encoding techniques [11] and provides versatility of the underlying assumption. We remark that these protocols support precomputation for higher online performance.

The extension of the work runs through several directions. First, our framework of two-party exponentiation implies a framework of multiparty exponentiation without honest majority. So a goal is to devise efficient protocols for the underlying A2M and M2A for the multiparty setting. Second, to duel with malicious adversaries without loss of efficiency, instead of general knowledge proofs, a specific arithmetic proof is required. Furthermore, it is worth investigating the multiparty exponentiation in a general adversary setting. Finally, our solution for a general modulus requires one invocation of a secure comparison. Hence, a cheap secure comparison protocol, comparable to the cost of a secure multiplication protocol, would be a key to improve efficiency further.

References

1. Yao, A.C.C.: How to Generate and Exchange Secrets. In: Proc. 27th FOCS. (1986) 162–167

2. Goldreich, O., Micali, S., Wigderson, A.: How to Play any Mental Game - A Completeness Theorem for Protocols with Honest Majority. In: Proc. 19th STOC. (1987) 218–229
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In: Proc. 20th STOC. (1988) 1–10
4. Chaum, D., Crepéau, C., Damgård, I.: Multiparty Unconditionally Secure Protocols. In: Proc. 20th STOC. (1988) 11–19
5. Lindell, Y., Pinkas, B.: Privacy-Preserving Data Mining. Journal of the Cryptology **15**(3) (2002) 177–206
6. Bunn, P., Ostrovsky, R.: Secure Two-Party k-Means Clustering. In: Proc. 14th CCS. (2007) 486–497
7. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T.P., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure Multiparty Computation Goes Live. In: Proc. 13th Financial Cryptography. (2009) 325–343
8. Lindell, Y., Pinkas, B.: Secure Multiparty Computation for Privacy-Preserving Data Mining. Journal of the ACM **1**(1) (2009) 59–98
9. Damgård, I., Mikkelsen, G.L.: Efficient Robust and Constant-Round Distributed RSA Key Generation. In: Proc. 7th TCC. (2010) 183–200
10. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding Cryptography on Oblivious Transfer - Efficiency. In: Proc. 28th CRYPTO. (2008)
11. Ishai, Y., Prabhakaran, M., Sahai, A.: Secure Arithmetic Computation with No Honest Majority. In: Proc. 6th TCC. (2009) 294–314
12. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In: Proc. 3rd TCC. (2006) 285–304
13. Cleve, R.: Limits on the Security of Coin Flips when Half the Processors are Faulty. In: Proc. 18th STOC. (1986) 364–369
14. Gentry, C.: Fully Homomorphic Encryption using Ideal Lattices. In: Proc. 41st STOC. (2009) 169–178
15. van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully Homomorphic Encryption over the Integers. In: Proc. 29th EUROCRYPT. (2010) 24–43
16. Algesheimer, J., Camenisch, J., Shoup, V.: Efficient Computation Modulo a Shared Secret with Application to the Generation of Shared Safe-Prime Products. In: Proc. 22nd CRYPTO. (2002) 417–432
17. Kiltz, E., Leander, G., Malone-Lee, J.: Secure computation of the mean and related statistics. In Kilian, J., ed.: TCC. Volume 3378 of Lecture Notes in Computer Science., Springer (2005) 283–302
18. Damgård, I., Thorbek, R.: Linear Integer Secret Sharing and Distributed Exponentiation. In: Proc. 9th PKC. (2006) 75–90
19. Nishide, T., Ohta, K.: Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol. In: Proc. 10th PKC. (2007) 343–360
20. Beaver, D.: Commodity-based Cryptography (Extended Abstract). In: Proc. 29th STOC. (1997) 446–455
21. ElGamal, T.: A Public Key Cryptosystem and a Signature Scheme based on Discrete Logarithms. In: Proc. 5th CRYPTO. (1985) 223–238
22. Paillier, P.: Public-Key Cryptosystems based on Composite Degree Residuosity Classes. In: Proc. 18th EUROCRYPT. (1999) 223–238