Minimal-Latency Secure Function Evaluation

Donald Beaver [⋆]

CertCo, Inc.

Abstract. Sander, Young and Yung recently exhibited a protocol for computing on encrypted inputs, for functions computable in NC^1. In their variant of secure function evaluation, Bob (the "CryptoComputer") accepts homomorphically-encrypted inputs (x) from client Alice, and then returns a string from which Alice can extract $f(x, y)$ (where y is Bob's input, or *e.g.* the function f itself). Alice must not learn more about y than what $f(x, y)$ reveals by itself. We extend their result to encompass NLOGSPACE (nondeterministic log-space functions).

In the domain of multiparty computations, constant-round protocols have been known for years [BB89,FKN95]. This paper introduces novel parallelization techniques that, coupled with the [SYY99] methods, reduce the constant to 1 with preprocessing. This resolves the conjecture that NLOGSPACE subcomputations (including log-slices of circuit computation) can be evaluated with latency 1 (as opposed to just $O(1)$).

1 Introduction

We consider variants of the now-classic problem raised by Yao [Y86] in which Alice and Bob wish to compute $f(x, y)$ while keeping their respective inputs x and y private. Roughly speaking, their computation should be as "secure" as if a trusted third party had accepted their inputs and provided nothing but the final output. This problem, *Secure Function Evaluation*, has a rich history of investigation, with a great deal of attention given to minimizing needed assumptions and communication complexity.

CEI. One particular variant is that of *Computing on Encrypted Inputs (CEI)*, in which Alice provides Bob with encryptions of x (or its bits), and Bob must enable Alice to determine $C(x)$ without revealing his "program" C. Mathematically, C can itself be encoded as an input to a universal circuit, hence this variant can be subsumed in general secure function evaluation. But the ground rules for CEI are somewhat different, in that Alice provides her input in the form of encryptions rather than through an inventor's flexibly chosen alternative (such as indirectly through oblivious transfer [R81]).

This is somewhat different than the general 2-party setting, in which encryptions can be used as an implementation tool but are not required. Moreover, the encryptions used in "Yao gates" and other earlier techniques are usually meant to encrypt random secrets that indirectly represent secret bits, as opposed to

[⋆] CertCo, 55 Broad St., New York, NY 10004; `beaverd@certco.com`.

encrypting the secret bits themselves. (A concrete hint if this is confusing: Alice often gets to learn one secret x_0 or another secret x_1, each of which is itself random; but the actual value represented by this process is 0 if Alice learns x_0, or 1 if Alice learns x_1.)

In the general SFE setting for two parties, preprocessing obviates the need for "encrypted inputs" and other extra work, since a "scrambled universal circuit" can be prepared in advance and then applied in one round as soon as the actual inputs become available. The challenge is therefore to achieve a one-round protocol without preprocessing (other than public-key initialization and the like).

Recently, Sander, Young and Yung provided a novel construction that enables *non-interactive* computing on encrypted inputs for functions in NC^1, namely functions computed by bounded-fan-in log-depth circuits. ("Non-interactive" means that Bob can complete his computation and return the result to Alice without conversation beyond receiving the initial message from Alice; obviously, Alice's inputs must be communicated to Bob in some form.) Alice simply drops off her input, Bob processes it, and Alice picks up the results. Dubbed *"crypto-computing"* by [SYY99], this methodology has applications to mobile computing and other settings.

Our contribution to non-interactive CEI (cryptocomputing) is to extend the class of functions to NLOGSPACE, *i.e.* non-deterministic logspace, a superclass of NC^1. This extension relies on matrix techniques from Feige, Kilian and Naor [FKN94], but also employs a newly contributed *inversion-free reduction* (§5.5) to compute products of secret group elements in one pass. With these methods, functions in NLOGSPACE can be evaluated in 1 round from scratch, answering a challenge left open by [SYY99], namely whether complexity beyond NC^1 is attainable for non-interactive computing on encrypted inputs.

MSC. Another twist on Secure Function Evaluation introduces some number n of parties, each holding a private input x_i, who wish to compute some function $f(x_1, \ldots, x_n)$ [GMW86,GMW87,BGW88,CCD88]. This version, known as *Multiparty Secure Computation (MSC)*, has also been the subject of extensive analysis.

When "computational" security is considered (as opposed to information theoretic), it is in fact possible to reduce any poly-depth circuit to a protocol with $O(1)$ rounds [BMR90]. With preprocessing and complexity-theoretic assumptions, those methods enable the results to be ready in one round after the inputs are provided, as mentioned above for the 2-party case.

Instead, we focus on the challenge of information theoretic security. For efficient solutions (polynomial message size and local computation), the number of rounds of communication is generally related to the circuit depth for f. Bar-Ilan and Beaver introduced techniques to reduce the computation by log-factors [BB89]; thus functions in NC^1 can be computed in a constant expected number of rounds. In fact, the methods from [FKN94] extend this to functions in NLOGSPACE. But the $O(1)$ constants, while small, still exceed 1.

Unlike the CEI setting, we do focus here on minimizing the *latency* of the computation, namely the number of rounds from when the inputs are supplied to when the output is ready. In [B91] it was shown that 1-round latency for secret multiplication (among other things, such as multiplicative inversion) is achievable.

Applying that work in a brute-force fashion to the [BB89,FKN94] solutions still gives a constant latency exceeding 1, because of the need to compute multiplicative inverses prior to evaluating components of a large product. We apply the methods of [SYY99] to reduce the latency to 1 for NC^1. The final construction provides a particularly elegant view of multiparty computation expressed as a secret linear combination of inputs.

With the inversion-free reduction described in this work, we also show how to achieve a latency of 1 for NLOGSPACE secret computations by avoiding the two-phase process in earlier works. Especially attractive is the fact that, apart from the preprocessing, no broadcast is needed. Thus a single dissemination (broadcast message without agreement) from each honest participant suffices for each NLOGSPACE-subcomputation.

2 Background and Definitions

We consider two different cases for function evaluation: the two-party case and the multiparty case. In the two-party case, hidden values can be represented through encryption, through oblivious transfer, or other such constructs. In the multiparty case, values can be represented through encryption, of course, but more interestingly through secret sharing [S79,B79].

It should be noted that the manipulations of these fundamental representations – encryptions or shares – are quite similar. Thus we may speak of "secret addition" to mean a homomorphic encryption (*e.g.* $E(a)E(b) = E(a \oplus b)$) or to mean a homomorphic additive sharing (*e.g.* $h(x) = f(x) + g(x)$ from $h(i) = f(i) + g(i)$). In general, "secret $\langle$*operation*$\rangle$" can be interpreted according to context, whereby the representation of the result is calculated from the representations of the inputs – be it encryption or sharing or otherwise.

Likewise, the "reconstruction" or "revelation" will refer to interpolation of shares, or decrypting of values and propagation through trees, *etc.*.

We note these observations explicitly to avoid doubling the size of the exposition, since, for example, the use of multiplicative inverses will be discussed both in the context of encrypted-representations and shared-representations.

2.1 Secret Sharing and Multiparty Computation

We refer the reader to [S79,B79,GMW87,BGW88,CCD88] for more detailed exposition. A secret value x can be shared among n parties, at most t of whom are colluding, by selecting t random coefficients $a_t, \ldots, a_1$ and defining $f(u) = a_t u^t + \cdots + a_1 u + x$. Player i receives the **share** $f(i)$. With $t+1$ correct shares,

$f(0) = x$ can be determined. With t or fewer shares, no information about x is revealed.

If $f(u)$ represents $f(0) = x$ and $g(u)$ represents $g(0) = y$, then $h(u) = f(u) + g(u)$ represents $x + y$, and shares $h(i)$ are easily calculated without interaction as $f(i) + g(i)$. Multiplication protocols are more model-dependent, but generally within a small number of rounds of interaction, a polynomial representation of xy can be obtained as well.

There are a variety of models to which our results apply – for example, $t < n/2$ covers one class, and $t < n/3$ covers another. (One can also withstand general adversary structures such as "Q2" and "Q3.") Broadcast is assumed (at least for preprocessing), but we place no restrictions on the computational power of attackers.

For simplicity, we consider t-adversaries who make a static choice of whom they will corrupt at the outset, and we investigate independence rather than simulatability. (Generalizations are possible.) Let $f(x_1, \ldots, x_n)$ be a function on n inputs, each of size m, producing a result of size m, and described by a boolean circuit family C_f. A multiparty protocol for f is a collection $\{P_1, \ldots, P_n\}$ of interactive Turing machines, each taking input m and a private m-bit argument x_i, and producing a final output y_i. A t-adversary is allowed to substitute (and coordinate) up to t of the programs. Two inputs are T-equivalent if they are identical on inputs in T and evaluate under f to the same result.

Definition 1. *A protocol $\Pi = \{P_1, \ldots, P_n\}$ for f is (information theoretically)* **t-secure** *if, for any coalition $T \subseteq \Pi$ of size $|T| \leq t$, and for any T-equivalent input $(x_1, \ldots, x_n)$, the view obtained by T is identically distributed.*

To complicate the analysis, we may allow the inputs x_i to be supplied at some round ρ after the protocol starts. The number of rounds of preprocessing (independent of inputs) is then ρ, and the **latency** is the total number of rounds less ρ. When considering protocols that divide the computation of f into "slices" (*i.e.*, subcomputations), we also consider the latency of computing each slice as the maximal number of rounds from when the previous slice is completed to when the current slice is done.

2.2 Computing on Encrypted Inputs

In CEI, we would like to capture the challenge of dropping off encrypted inputs which are then manipulated in a somewhat black-box fashion to produce a result for the client. This requires a bit more than postulating a homomorphic encryption scheme, as we now discuss.

One of the earliest and most fundamental techniques for two-party circuit evaluation is due to Yao [Y86]. In this method, Bob prepares a scrambled circuit in which each gate is represented by an encrypted table (called a "Yao gate"), and each wire w is represented by a pair of hidden triggers, *i.e.* numbers w_0 and w_1. The value of wire w is 0 if Alice discovers w_0; it is w_1 if Alice discovers w_1; in other cases it is undefined. By propagating the discovered triggers through each encrypted table, Alice is able to calculate the trigger for the output wire of the gate. She is not told how to interpret the triggers, *i.e.* to which bit the trigger corresponds – except for the final output wire.

The needed interaction is minimal, on the same order as the [SYY99] setting. Alice must obtain initial wire triggers that correspond to her secret input bits. This is achieved through chosen-1-out-of-2 Oblivious Transfer [R81]: Bob supplies w_{i0} and w_{i1} for an input wire w_i; Alice learns precisely one, namely w_{i,x_i}; but Bob does not learn which Alice chose. Bob also sends Alice the scrambled circuit. Subsequently, Alice can calculate the output value without further interaction.

Given a homomorphic encryption scheme, one quick way to implement the OT is by way of a generalization of Den Boer's method [dB90]. Alice sends $E(c)$ where c is a bit describing her choice. Bob responds with $\{E(b_0), E(a)/E(b_0)\}$, $\{E(b_1), E(a)E(1)/E(b_1)\}$, with sets and members permuted randomly. Given proper behavior, Alice decrypts the sets to $\{0,1\}$ and $\{b_c, b_c\}$, hence she obtains b_c. The authors of [SYY99] invoke a variety of options to demonstrate good behavior without introducing interaction; those options apply here as well. Note that Bob can send the scrambled circuit along with his OT reply, making the whole interaction non-interactive, so to speak.

Thus, if Bob can employ a homomorphic encryption secure against Alice, an immediate solution is possible for any polynomial-time f, not just one in NC^1. This solution makes an end run around the spirit of the problem. Since it is hard to provide a formal test that captures whether Bob's computations are nothing "more" than a manipulation of encrypted values (there are a lot of clever and indirect things he can do), we turn to a simple requirement: the protocol must be information-theoretically secure against Alice.

Definition 2. *A* **CEI protocol** *for function f represented by circuit C_f is a two-party protocol consisting of a message from Alice to Bob followed by one in return. The protocol is* **correct** *if for all inputs (x,y), Alice's output is $f(x,y)$ except with negligible probability. A CEI protocol is* **private** *if it is computationally-private against Bob and information-theoretically private against Alice.*

Concrete Examples of Encryptions A couple of common encryptions make suitable candidates. One is the Goldwasser-Micali encryption [GM84] in which N is a public Blum integer with a private key consisting of its factors P, Q. Bit b is encrypted as $(-1)^b r^2$ for a random r. This is secure assuming that quadratic residues are indistinguishable from non-residues (the Quadratic Residuosity Assumption, or QRA).

A second candidate is a variant of El-Gamal encryption with primes P, Q satisfying $P = 2Q + 1$, and a generator g of Z/QZ. Corresponding to private key x is the public key $y = g^x$. To encrypt message m taken from the space of quadratic residues, compute $\{g^r, y^r m\}$. Encryption of 0 and 1 uses two fixed, public quadratic residues m_0, m_1. The security of this method is equivalent to Decision Diffie-Hellman [TY98,NR97].

In each of these cases, given some $E(b)$, it is easy to see how to sample random encryptions uniformly from the set of all encryptions of b, or of $1 - b$, even without knowing b.

3 Pyramid Representation

The foundation for the recent 1-round protocol of Sander, Young and Yung is an ingenious tree representation for a circuit output. We will build multiparty protocols around their architecture, thus we give details here; the familiar reader can skip to the next section.

Let us coin the term **pyramid representation** to describe the data structure employed in [SYY99]: a complete 4-2 tree, *i.e.* a tree with degree 4 at root and even-level nodes, and with degree 2 at all odd-level nodes. We take the root to be level $2d$ and the leaves to be at level 0. There are 8^d leaves.

There are three important aspects to the SYY construction. First, the nodes can be evaluated in terms of a given circuit, resulting in the root being assigned a value equal to the output of the circuit. Second, the pyramid representation can be constructed from encrypted leaf values without knowing what the cleartext bits are. Third, the pyramid representation can be randomized so that it appears chosen uniformly at random from all such representations that evaluate to the given root value.

The authors of [SYY99] refer to the construction and randomizing as *inattentive* computing, suggesting that the party who performs the tasks need not pay attention to the actual values themselves. The manipulations are oblivious to the contents.

Decoding. In slightly more detail for completeness, we first summarize how evaluation/decoding takes place, given bit assignments to the leaves. (Ultimately, each leaf corresponds to an encryption, and the value of the leaf node is the decrypted bit.) Propagating upward, a node at level $2k + 1$ has two children, (a, b), and is assigned the value $a \oplus b$. A node at level $2k$ has four children (a, b, c, d), and is assigned the value 0 if three are labelled 0 and one is 1, or respectively 1 if three children are labelled 1 and one is 0. (All other cases are syntactically unacceptable and are given an `undef` label.) This three-of-one-kind representation is critical.

Construction. To construct a pyramid representation of the value of some function f applied to input bits $x_1, \ldots$, one must apply the gates of a circuit C_f for f to the nodes in the representation. Inputs and constants lie at the leaves. Without loss of generality, express C_f as NOT and OR gates. We briefly summarize the SYY construction using the following procedures, which depend on the level of the node in the tree:

- NOT(x:level 0) gives y:level 0.
  - set $y = x \oplus 1$. (Later, 0 and 1 may be encoded by $(0, 1)$ and $(1, 0)$, in which case this is operation is instead a swap.)

- $\textsc{not}(x\text{:level } 2k+2)$ gives $y\text{:level } 2k+2$.
  - return $((\textsc{not}(a_1), a_2), (\textsc{not}(b_1), b_2), (\textsc{not}(c_1), c_2), (\textsc{not}(d_1), d_2))$, where $x = ((a_1, a_2), (b_1, b_2), (c_1, c_2), (d_1, d_2))$.
- $\textsc{or}(x\text{:level } 2k, y\text{:level } 2k)$ gives $z\text{:level } 2k+2$.
  - return $((x, 0), (y, 0), (x, y), 1')$ where 0 denotes a level $2k$ zero, and $1'$ denotes a level $2k+1$ one.

The ingenious motivation behind the three-of-one-kind representation is now more clear. Negating each individual bit in the multiset $\{0, 0, 0, 1\}$ provides a three-of-one-kind result $\{1, 1, 1, 0\}$, and vice versa. More importantly, the results of the OR routine are always in a three-of-one-kind configuration, when interpreted at a higher level. Explicitly:

x	y	$(x, 0)$	$(y, 0)$	(x, y)	1’	OR
0	0	0’	0’	0’	1’	{0’,0’,0’,1’}
0	1	0’	1’	1’	1’	{0’,1’,1’,1’}
1	0	1’	0’	1’	1’	{1’,0’,1’,1’}
1	1	1’	1’	0’	1’	{1’,1’,0’,1’}

where the primed values are interpreted at the next layer up ($(0, 0)$ and $(1, 1)$ are written 0’, *etc.*).

Randomization. [SYY99] show that the following straightforward method turns a particular pyramid representation of some result z into a randomly-chosen valid pyramid representation of z, thereby hiding the inattentive steps used to construct the original representation. The randomization method is itself inattentive to the contents of the pyramid.

- $\textsc{randomize}(x\text{:level } 2k+2)$ gives $y\text{:level } 2k+2$.
  - let x be $((x_{11}, x_{12}), (x_{21}, x_{22}), (x_{31}, x_{32}), (x_{41}, x_{42}))$;
  - for $i = 1..4$ and $j = 1..2$, set $b_{ij} \leftarrow \textsc{randomize}(x_{ij})$;
  - for $i = 1..4$, set c_i by random choice to be (b_{i1}, b_{i2}) or $(\textsc{not}(b_{i1}), \textsc{not}(b_{i2}))$;
  - choose random permutation $\sigma \in S_4$ and return $(c_{\sigma(1)}, c_{\sigma(2)}, c_{\sigma(3)}, c_{\sigma(4)})$.

3.1 Non-Interactive Computing on Encrypted Inputs

In the [SYY99] paper, this construction is applied to encrypted inputs. That is, Alice presents CryptoComputer Bob with encryptions $E(x_i)$ of each of her input bits x_i, along with their inverses $E(1 - x_i)$. This enables Bob to create the level 0 leaf labels. Note that Bob can also encrypt his secret inputs x_j, as well as any known constants, thereby filling in any other needed labels.

Now, without knowing the contents of the encryptions, Bob can invoke the NOT and OR routines, and finally the RANDOMIZE routine. The result is a pyramid representation whose root value is $f(x, y)$. Bob sends this to Alice.

Alice is able to decrypt the labels on the leaves and can subsequently evaluate the root value.

4 Multiparty Secure Computation

With the pyramid data structure in place, we are now ready to give a multiparty secure computation for NC^1.

4.1 Latency vs. Cost: Circuit Randomization

When calculating from scratch, our MSC results will generally incur a minimal cost of one secret multiplication. While still better than previously published results, this falls short of the most desirable bound of 1 round, period.

Instead, we focus on **latency**, defined as the number of rounds from when the inputs to a computation phase are provided until the output (whether secret or public) is complete. Preprocessing is acceptable (and likely required), but it must be independent of the inputs to be used.

Latency is particularly important when evaluating a depth-D circuit using $(\log n)$-slices to speed up the number of rounds. A brute-force approach would normally require $C\dot{D}/\log n)$ multiplications with C much larger than 1 (and even including our results below, it would be at least $D/\log n$ multiplications). If, however, the later slices benefit from preprocessing that is performed during the first slice, then the net running time can be drastically reduced. That is, one multiplication plus $D-1$ rounds is far better than D sequential multiplications.

One way to improve latency was shown by Beaver, using a technique called *circuit randomization* [B91]. With appropriate preprocessing, this enables each secret multiplication to finish in one round, an order of magnitude faster than the cost of a secret multiplication from scratch.

The preprocessing is simple, consisting of computing secret products on secret, random inputs. Thus, for example, secrets a, b, c with $c = ab$ are created in advance. When x and y are ready to be multiplied, the differences ("corrections") $\Delta x = x - a$ and $\Delta y = y - b$ are published. The "correction" to c, namely $\Delta z = xy - c$, then becomes a straightforward linear combination with public coefficients (the Δx, Δy values). The bottom line is that secret multiplication has a latency of 1 round.

We shall see below that the same conclusion applies to NC^1 (and to NLOGSPACE): secret NC^1 computations have a latency of 1 round. Interestingly, the following result can be derived in different ways, with or without the recent SYY methods.

Claim. Let f be represented by a circuit C_f of polynomial size. There exists a secure MSC protocol to compute NC^1 slices of C_f with a latency of 1 round.

4.2 NC1 Via Secret Quadratic Forms and SYY

The first of two ways to achieve Claim 4.1 employs [SYY99] with secretly shared values in place of encrypted bits. The "inattentive" creation of a pyramid representation on secrets is done as a multiparty computation in a straightforward manner.

The calculation of NOT at level 0 is simple: non-interactively compute $1 - x$ secretly. Second, the RANDOMIZE step can be calculated using a secret quadratic form applied to the inputs – or in other words, a "linear" combination of input values in which the coefficients are themselves secrets. These coefficients are chosen randomly but with certain restrictions.

There are only two steps in RANDOMIZE in which random choices are made. In the 2-party Computing on Encrypted Inputs setting, the "CryptoComputer" would make these choices and ensure that they remain secret. In the MSC application, these choices are also kept secret. We must ensure that they can be selected and applied efficiently.

Referring to §3, there are two main steps for applying random choices. First is the choice between (b_{i1}, b_{i2}) and $(\textsc{not}(b_{i1}), \textsc{not}(b_{i2}))$. This choice can be executed by creating a new secret bit d_i, then setting (at leaves):

$$\begin{aligned} b_{i1} &= d_i x_{i1} + (1 - d_i)(1 - x_{i1}) \\ b_{i2} &= d_i x_{i2} + (1 - d_i)(1 - x_{i2}) \end{aligned}$$

The manipulation at higher level nodes is similar: the multiplication by d_i is propagated to the children.

Similarly, the random selection of a permutation from S_4 can be modelled by a secret permutation matrix $A = [a_{ij}]$, so that the resulting quadruple is (y_1, y_2, y_3, y_4) where $y_i = \sum_j a_{ij} c_j$.

At each odd-level node in the pyramid representation, then, a secret random bit is generated. At each even-level node above 0, a secret random S_4 permutation is generated.

If these operations are composed, the result is a collection of coefficients C_{ij} such that leaf i is $C_{i0} + \sum_j C_{ij} x_j$. These coefficients are products of the coefficients assigned on the path down to leaf i. Thus they can be efficiently calculated (secretly, of course) in a preprocessing phase.

Noting that [B91] enables quadratic forms on secrets to be evaluated with 1-round latency, Claim 4.1 is satisfied.

4.3 Some Details

For concreteness, here are some ugly programming steps for the protocol. The SYY construction induces at each node a tree with formulas in it. One can apply a syntactic NOT operation to a leaf label s by replacing s by $1 - s$. One can apply a NOT to a higher node by applying NOT recursively to each of the left grandchildren (as in SYY). One can also perform linear combinations recursively on two trees of formulas in a direct manner:

$$\begin{aligned} a((t_1, t_2), (t_3, t_4), (t_5, t_6), (t_7, t_8)) + b((u_1, u_2), (u_3, u_4), (u_5, u_6), (u_7, u_8)) = \\ ((at_1 + bu_1, at_2 + bu_2), (at_3 + bu_3, at_4 + bu_4), \\ (at_5 + bu_5, at_6 + bu_6), (at_7 + bu_7, at_8 + bu_8)). \end{aligned}$$

The first (non-interactive) preparation creates a raw tree of formulas:

1. Start with circuit C_f which is applied to input bits $x_1, \ldots, x_m$.

2. Create a *raw pyramid program*: Each node contains a tree of formulas using x_i's and constants. Place x_i's and constants at the leaves according to C_f. Propagating upward, create a formula tree at each node according to the Construction in 3. (For example, at level 2, OR(NOT(x_1),x_2) would be labelled with the formula tree $((1-x_1,0),(x_2,0),(1-x_1,x_2),(0,1).)$

The second (non-interactive) preparation adds symbols that correspond to the randomization:

1. For each odd-level node v, create symbol $d(v)$. For each even-level node v, create 16 symbols $a(v,i,j)$ for $1 \leq i,j \leq 4$.
2. Create a *randomized pyramid program*: Propagating upwards from leaves, apply randomization symbols.
   2A. Replace the current T by $T' = d(v)T + (1-d(v))NOT(T)$. (This involves recursively applying $d(v)$ symbols and NOT's; the result is a tree of formulas over inputs, constants, and d_i's.)
   2B. Now say the current T is $((b_{11},b_{12}),(b_{21},b_{22}),(b_{31},b_{32}),(b_{41},b_{42}))$. Replace b_{im} with $\sum_j a(v,l,j)b_{jm}$. (Again, the $a(v,l,j)$ symbols trickle to the leaves.)

The result is a pyramid of formulas in which each formula can be written as $C_{i0} + \sum_j C_{ij}x_j$, where the C's are formulas on the randomization symbols and constants alone.

This gives an $O(8^d)$-sized "program" for the preprocessing phase, where d is the circuit depth of C_f. Generate random secret bits for each of the $d(v)$ symbols. Generate random secret permutation matrices for each set $\{a(v,i,j)\}$. Evaluate each C_{i0} and C_{ij} *secretly.* The preprocessing takes constant rounds.

We now have a pyramid in which each leaf i contains an expression $C_{i0} + \sum_j C_{ij}x_j$. Following the approach of [B91], these results can be precomputed at random values $\hat{x}_i$. When the x_j inputs are provided, "corrections" $(x_i - \hat{x}_i)$ are announced, corrections to the pyramid entries are disseminated (no broadcast/agreement needed), and each player then calculates the entries himself. Each player then evaluates the pyramid according to the instructions of [SYY99] (see §3).

Preparing the Coefficients We digress with a few remarks on alternatives for obtaining the C_{ij} coefficients. Several avenues present themselves:

- generation by precomputation;
- generation by Trusted Third Party (TTP) or Server;
- generation by composition.

The previous section considered precomputation.

In a hybrid model more akin to [SYY99], one can rely on a TTP who supplies secret shares of the coefficients to the participants. While zero-knowledge proofs can ensure correctness (*i.e.* the coefficients are a proper permutation), one must trust that the TTP does not leak the coefficients. This trust model is similar to the CryptoComputer model of [SYY99]; secrecy relies on maintaining secrecy of the RANDOMIZE step.

Finally, verified sets of coefficients from the TTP's can be composed. This corresponds to allowing each TTP to execute the RANDOMIZE step. As long as one TTP maintains discretion, the conclusions of [SYY99] will apply and the results will be secure. Of course, if the TTP's are taken to be the participants themselves (*eg.* $t+1$ of them), then a secret matrix product on several matrices is required, which gets us back to the initial problem.

5 Matrix Representations

We present a background for matrix-based computing and finish the section with our new inversion-free reduction.

5.1 Secret Group Operations

The following subroutines are applicable to 2-party and to multiparty settings. Note that the group need not be abelian, thus matrices are perfectly fine candidates. The costs are $O(1)$ multiplications; hence if secret multiplication takes $O(1)$ rounds, the net cost is $O(1)$ rounds. (As described later, secret multiplication generally has 1-round *latency* after preprocessing, so these routines are very short in terms of latency.)

Inverses. The authors of [BB89] demonstrated how to compute a secret inverse of a secret group element X in $O(1)$ multiplications using the following trick: choose secret element U; secretly calculate $Y = XU$ and reveal Y; publicly calculate Y^{-1}; secretly multiply $Z = UY^{-1}$. Clearly, $Z = X^{-1}$, yet Y is distributed uniformly at random, revealing nothing about X (as long as U remains uncompromised).

Polynomial-Length Products. Let $M_1, \ldots, M_N$ be secret group elements. The goal is to calculate $M = \prod_i M_i$ secretly. The following application (with minor differences) arose in [K88,K90] and [BB89]:

$$M = R_0^{-1} R_0 M_1 R_1^{-1} R_1 M_2 R_2^{-1} \cdots R_{N-1} M_N R_N^{-1} R_N$$

where $R_0, \ldots, R_N$ are secret, random, invertible group elements (R_0 can be set to the identity). Let $S_i = R_{i-1} M_i R_i^{-1}$. Then the set $\{S_i\}$, if made public, reveals nothing about $\{M_i\}$; it appears uniformly random, subject to producing the same overall product.

A protocol that follows this structure (compute R_i's and inverses, compute and reveal S_i's) will incur $O(1)$ multiplications plus the cost of generating random invertible elements. It will nevertheless exceed 1 round.

5.2 3×3 Products for NC1

Building on a result of Barrington [B86], Ben-Or and Cleve [BC88] showed that NC^1 computations are equivalent to products of polynomially-many 3×3 matrices. In their representation, inputs are supplied as an identity matrix with the top right $(1, 3)$ zero replaced by the input value. The final result is also read from the $(1, 3)$ entry of a specified product of such "input" matrices interspersed with certain constant matrices. In fact, the final product is simply an identity matrix with the top right zero replaced by $f(x_1, \dots, x_n)$.

Without going into further detail, we mention simply that the number of matrices involved in a depth-d computation will be some $N = O(4^d)$, and that each matrix is either a well-known constant or simply contains an input variable (possibly negated) in the $(1, 3)$ entry as above.

5.3 $N \times N$ Products for NLOGSPACE

More recently, Feige, Kilian and Naor [FKN94] described how to formulate NLOGSPACE computations as a product of $N \times N$ matrices, where N is polynomial in the input size. In their setup, the top right $(1, N)$ entry of the final product M indicates the final output: 0 if the entry is zero, or 1 if the entry is nonzero.

Because [FKN94] used the $N \times N$ construction to solve a slighly different task, in which Alice and Bob provide sufficient data to a Combiner so that the Combiner can calculate $f(x, y)$ without learning x and y, they also focused on leaving $f(x, y)$ (and nothing else) in the output. While this occurs automatically in the 3×3 matrix case (for NC^1), [FKN94] had to provide additional secret matrices Q_L and Q_R to randomize the final product matrix. With Q_L and Q_R of a particular, randomized form, they showed that $Q_L M Q_R$ was uniformly random subject to the entry at $(1, N)$ being either zero if the output was 0 or random and nonzero if the output was 1.

It is not hard to verify that secret Q_L and Q_R matrices can be generated in a constant expected number of rounds.

5.4 Direct Output or Slice Output

There is a distinction between producing the final result of a function in some public fashion (known to one or more parties) and producing a secret representation of the final result. The latter can be used to speed up larger circuit evaluations by slicing them [K88,K90,BB89] into (for example) log-depth layers.

In any case, it is often simple to convert a direct-output computation to one that preserves the output as a secret for input to further computation. Simply create an additional secret r, directly output the result of $f() - r$, and implicitly add the public value $f() - r$ to the secretly represented r.

(This does not obviate the use of Q_L and Q_R in [FKN94], however, since there are a host of other entries ($N^2 - 1$ of them, in fact) whose public revelation may compromise sensitive information. Their approach was to open the final matrix completely.)

5.5 Multiplication Without Secret Inverses

One of the difficulties with using the matrix multiplication methods described in §5.1 is that they are *prima facie* interactive in nature. To calculate an inverse, one must publicly reveal the randomized product, which is then interactively fed back into another pass. To calculate a long product of elements, one first reveals the intermediate products of triples, then calculates their product and feeds it back into another phase (multiplying by secrets on left and/or right).

Here, we propose an *inversion-free reduction* from a product to a list of publicized matrices which can be combined to calculate the original product. (While no inversions are needed in the reduction, some of the resulting matrices must be inverted before multiplying them together.)

Starting with a polynomial-length product $M = \prod M_i$, we create secret, invertible elements $R_0, \ldots, R_N$ as before. But now, also create secret, invertible elements $\hat{R}_0, \ldots, \hat{R}_N$. Write:

$$\begin{aligned} M = (\hat{R}_0)(R_0\hat{R}_0)^{-1}(R_0M_1\hat{R}_1)(R_1\hat{R}_1)^{-1}(R_1M_2\hat{R}_2)\cdots \\ \cdots(R_{N-1}\hat{R}_{N-1})^{-1}(R_{N-1}M_N\hat{R}_N)(R_N\hat{R}_N)^{-1}(R_N). \end{aligned}$$

Let $S_i = R_{i-1}M_i\hat{R}_i$, and let $\hat{S}_i = R_i\hat{R}_i$. Then:

$$M = \hat{R}_0\hat{S}_0^{-1}S_1\hat{S}_1^{-1}S_2\cdots\hat{S}_{N-1}^{-1}S_N\hat{S}_N^{-1}R_N.$$

It is not hard to generalize [K88,K90,BB89] to show that each S_i and $\hat{S}_i$ leaks no information. Define $S = \hat{S}_0^{-1}S_1\hat{S}_1^{-1}S_2\cdots\hat{S}_{N-1}^{-1}S_N\hat{S}_N^{-1}$. Then $M = \hat{R}_0SR_N$. While inverses are applied to the public values ($\hat{S}_i^{-1}$), no inversion is required to reduce the original product secretly to the list of public multiplicands.

6 Multiparty Secure Computation Revisited

6.1 Achieving NC1 for Multiparty Secure Computation

Claim 6.2 can now be demonstrated by an alternative approach. The inversion-free reduction of §5.5 enables MSC protocols with 1-round latency for NC^1 without relying on [SYY99], as the following indicates. Precompute the R_i and $\hat{R}_i$ matrices and reveal the $\hat{S}_i$ values.

Let $I(i)$ be the index of the secret input variable appearing in matrix M_i (if any). When each $R_iM_i\hat{R}_{i+1}$ product is expanded, each of the nine entries $\{s_{ikl}\}_{1\leq k,l\leq 3}$ in S_i is of the form $\alpha_{ikl}+\beta_{ikl}x_{I(i)}$. (If no variable appears, $\beta_{ikl}=0$.) Secretly precompute the α_{ikl} and β_{ikl} values.

Finally, when the input variables are supplied, it remains to publish each $\alpha_{ikl} + \beta_{ikl}x_{I(i)}$ in order to reveal the S_i matrices. This involves a single multiplication, which the methods of [B91] reduce to latency 1. (The product is precomputed on random inputs; the single round consists of disseminating an adjustment to the precomputed result.)

At this point, the $\hat{S}_i$ and S_i matrices have been revealed. The overall result can be evaluated without further interaction, or fed secretly into the next layer of computation.

6.2 Achieving NLOGSPACE for Multiparty Secure Computation

The generation of secret nonsingular $N \times N$ matrices, and appropriate secret Q_L and Q_R matrices, can be done in expected $O(1)$ rounds. Thus we find (as already claimed in [FKN94]) that there is a secure multiparty protocol for any NLOGSPACE function, using expected $O(1)$ rounds. But we can now strengthen that conclusion by applying the methods of the previous section to $N \times N$ matrices:

Claim. Let f be represented by a composition of D NLOGSPACE-computable functions each with output size polynomial in the size of f's input. There exists a secure MSC protocol to compute each NLOGSPACE-computable subfunction with a latency of 1 round. The overall protocol incurs $D + O(1)$ rounds.

7 Two Parties: Computing on Encrypted Inputs

In the case of Computing on Encrypted Inputs, we do not have the flexibility to allow preprocessing. Instead, we turn back to the [SYY99] for bootstrapping the product of $N \times N$ matrices.

The selection of random, secret, nonsingular matrices, and the individual computation of each of the Q_L, Q_R, S_i and $\hat{S}_i$ matrices can be performed in NC^1. Note that input bits and extra random bits are re-used in different, parallel sub-executions.

Thus, on a higher level, the protocol for NLOGSPACE consists of some number N of executions of various NC^1 calculations. These calculations provide Alice with the values for Q_L, Q_R, S_i and $\hat{S}_i$, which in turn enable her to compute the final bit. According to the proofs presented in [FKN94], these matrices provide no extra information. More details are below.

7.1 Computing NLOGSPACE on Encrypted Inputs

For a given function f in NLOGSPACE, the construction in [FKN94] produces a pair of adjacency matrices, A and B. The binary entries in A depend only on Alice's inputs (or on no inputs at all), and the entries in B depend only on Bob's inputs. The $(1, N)$ entry of $(AB)^N$ will be nonzero if and only if the result of f is 1; otherwise f is 0. To hide the other entries in $(AB)^N$, which may leak information, two extra secret matrices Q_L and Q_R are used, and the desired product is $M = Q_L(AB)^N Q_R$. Bob will enable Alice to find the product of these $2N + 2$ $N \times N$ matrices.

In our application, only Alice will learn the S_i and $\hat{S}_i$ matrices. Unlike other settings, this permits us to have Bob learn or set the randomizing matrices himself, as long as Alice doesn't.

1. For each input bit x_i held by Alice, Alice encrypts and sends $E(x_i)$ to Bob.

2. Bob selects $2N+1$ random R_i matrices and $2N+1$ random $\hat{R}_i$ matrices (set $R_{2N+2} = \hat{R}_{2N+2} = I$). Bob selects Q_L and Q_R at random according to the constraints in [FKN94]. He sets matrix B according to the inputs to f that he holds. Let $M_1 = Q_L$, $M_{2N+2} = Q_R$, and for $i = 1..N$ let $M_{2i} = A$ (values unknown to Bob) and $M_{2i+1} = B$.
3. Bob invokes N instances of the [SYY99] protocol. In instance i he uses Alice's encryptions to evaluate (for Alice) the result $S_{2i} = R_{2i-1} M_{2i} \hat{R}_{2i}$. In addition, Bob directly sends the following results to Alice: $S_1 = Q_L \hat{R}_1$, $S_{2N+2} = R_{2N+1} Q_R \hat{R}_{2N+2}$, $S_{2i+1} = R_{2i} B \hat{R}_{2i+1}$ for $1 \leq i \leq N$, and $\hat{S}_j = R_j \hat{R}_j$ for $1 \leq j \leq 2N+2$.
4. Alice receives pyramids for S_{2i} $(1 \leq i \leq N)$ and calculates S_{2i} accordingly. She then calculates $M = S_1 \hat{S}_1^{-1} \cdots \hat{S}_{2N+1}^{-1} S_{2N+2} \hat{S}_{2N+2}^{-1}$. If entry $(1, N)$ in M is nonzero, Alice outputs 1, else she outputs 0.

By inspection, the protocol takes one round. By arguments in [FKN94] and [SYY99], Alice's view of the pyramids and the direct matrices provides her no greater knowledge than the final result itself (from which she can construct the view). The product is clearly correct.

8 Closing Remarks

We have extended the reach of earlier results by applying new parallelization constructs. Two results obtain. Multiparty Secure Computation can be speeded up by creating subtasks of complexity NLOGSPACE, where the latency of computing each subtask is not just $O(1)$ but exactly 1. Likewise, Computing on Encrypted Inputs can be achieved non-interactively for functions in NLOGSPACE, not just NC^1.

We presented two approaches to achieving NC^1 computations for MSC with 1-round latency. One, based on [SYY99], has message size complexity of $O(8^d)$ (where d is circuit depth). The other requires $O(4^d)$. On closer inspection, the culprit seems to be the use of $(0,1)/(1,0)$ representations. In the MSC application, it can be removed, collapsing the SYY pyramid to size $O(4^d)$. It is remarkable that two distinct constructions converge to the same complexity, which may suggest a deeper relationship.

Acknowledgements The author gratefully acknowledges helpful discussions and inspirations from Moti Yung. Several referees made extremely helpful comments on content and presentation.

References

[BB89] J. Bar-Ilan, D. Beaver. "Non-Cryptographic Fault-Tolerant Computing in a Constant Expected Number of Rounds of Interaction." *Proceedings of PODC,* ACM, 1989, 201–209.

[B86] D. Barrington. "Bounded Width Polynomial Size Branching Programs Recognize Exactly those Languages in NC^1." *Proceedings of the* 18^{th} *STOC,* ACM, 1986, 1–5.

[B91] D. Beaver. “Efficient Multiparty Protocols Using Circuit Randomization.” *Advances in Cryptology – Crypto ’91 Proceedings,* Springer–Verlag LNCS 576, 1992, 420–432.

[BMR90] D. Beaver, S. Micali, P. Rogaway. “The Round Complexity of Secure Protocols.” *Proceedings of the* 22^{nd} *STOC,* ACM, 1990, 503–513.

[BC88] M. Ben-Or, R. Cleve. “Computing Algebraic Formulas Using a Constant Number of Registers.” *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 254–257.

[BGW88] M. Ben-Or, S. Goldwasser, A. Wigderson. “Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation.” *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 1–10.

[CCD88] D. Chaum, C. Crépeau, I. Damgaard. “Multiparty Unconditionally Secure Protocols.” *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 11–19.

[CDG87] D. Chaum, I. Damgard, J. van de Graaf. “Multiparty Computations Ensuring Secrecy of Each Party’s Input and Correctness of the Output.” *Advances in Cryptology – Crypto ’87 Proceedings,* Springer–Verlag LNCS 293, 1988.

[B79] G.R. Blakley. “Safeguarding Cryptographic Keys.” *Proceedings of AFIPS 1979 National Computer Conference*, NY **48**, 1979, 313–317.

[dB90] B. den Boer. “Oblivious Transfer Protecting Secrecy.” *Advances in Cryptology – Eurocrypt ’90 Proceedings,* Springer–Verlag LNCS 547, 1990, 31–45.

[FiSh86] A. Fiat, A. Shamir. “How to Prove Yourself: Practical Solutions to Identification and Signature Problems.” Proc. of Crypto ’86.

[FFS88] U. Feige, A. Fiat, A. Shamir. “Zero Knowledge Proofs of Identity.” *J. Cryptology* **1**:2, 1988, 77–94.

[FKN94] U. Feige, J. Kilian, M. Naor. “A Minimal Model for Secure Computation.” *Proceedings of the* 26^{th} *STOC,* ACM, 1994, 554–563.

[GMW86] O. Goldreich, S. Micali, A. Wigderson. “Proofs that Yield Nothing but their Validity and a Methodology of Cryptographic Protocol Design.” *Proceedings of the* 27^{th} *FOCS,* IEEE, 1986. pages 174–187. IEEE, 1986.

[GMW87] O. Goldreich, S. Micali, A. Wigderson. “How to Play Any Mental Game, or A Completeness Theorem for Protocols with Honest Majority.” *Proceedings of the* 19^{th} *STOC,* ACM, 1987, 218–229.

[GM84] S. Goldwasser, S. Micali. Probabilistic Encryption. *JCCS*, **28**:2, 270–299, 1984.

[K88] J. Kilian. “Founding Cryptography on Oblivious Transfer.” *Proceedings of the* 20^{th} *STOC,* ACM, 1988, 20–29.

[K90] J. Kilian. Uses of Randomness in Algorithms and Protocols. Cambridge, MIT Press, 1990.

[NR97] M. Naor, O. Reingold. “Number-Theoretic Consturctions of Efficient Pseudo-Random Functions.” *Proceedings of the* 38^{th} *FOCS,* IEEE, 1997.

[R81] M.O. Rabin. “How to Exchange Secrets by Oblivious Transfer.” TR-81, Harvard, 1981.

[SYY99] T. Sander, A. Young, M. Yung. “Non-Interactive CryptoComputing for NC1.” *Proceedings of the* 40^{th} *FOCS,* IEEE, 1999.

[S79] A. Shamir. “How to Share a Secret.” *Communications of the ACM,* **22**, 1979, 612–613.

[TY98] Y. Tsiounis, M. Yung. “On the Security of ElGamal-based Encryption.” Proceedings of PKC ’98, LNCS, Springer-Verlag, 1998.

[Y82] A. Yao. “Protocols for Secure Computations.” *Proceedings of the* 23^{rd} *FOCS,* IEEE, 1982, 160–164.

[Y86] A. Yao. “How to Generate and Exchange Secrets.” *Proceedings of the* 27^{th} *FOCS,* IEEE, 1986, 162–167.