Fast Multiparty Multiplications from shared bits

Ivan Damgård, Tomas Toft, and Rasmus Winther Zakarias*

Dept. of Computer Science, Aarhus University

Abstract. We study the question of securely multiplying N-bit integers that are stored in binary representation, in the context of protocols for dishonest majority with preprocessing. We achieve communication complexity $O(N)$ using only secure operations over small fields $\mathbb{F}_2$ and $\mathbb{F}_p$ with $\log(p) \approx \log(N)$. For semi-honest security we achieve communication $O(N)2^{O(\log^*(N))}$ using only secure operations over $\mathbb{F}_2$. This improves over the straightforward solution of simulating a Boolean multiplication circuit, both asymptotically and in practice.

1 Introduction

In multiparty computation, a number of players wish to carry out a computation on private inputs such that the desired result is the only new information released. A very large number of protocols exist for secure computations of general functions or even reactive functionalities. In this paper we focus on one extremely popular approach for this, which is to implement an ideal functionality known as an *arithmetic black-box*. Such a functionality $\mathsf{BB}_{\mathbb{F}}$ offers secure computation over some finite field $\mathbb{F}$: players can load values from the field into registers in the box, and can then ask for arithmetic operations to be done on data inside the box, and finally ask for results to be revealed to all players. In this paper we consider information theoretically secure implementations of arithmetic black-boxes, for the case of dishonest majority with preprocessing (see, e.g., [NNOB12,DPSZ12]). For all known implementations it is the case that addition is cheap because it does not require communication, while multiplication requires does require communication, of $\Omega(1)$ field elements.

The question we study in this paper is the following: suppose we are given an arithmetic black-box that offers operations over a small field. Suppose further that we are given two N-bit numbers a, b, where the box already holds $2N$ registers containing the bits of a and b, which we will denote by $[a]_{\mathsf{bits}}, [b]_{\mathsf{bits}}$ in the following. Suppose also N is large, i.e., much larger than the bit-size of the field elements the box can handle. How much communication is required to compute the product $[ab]_{\mathsf{bits}}$ securely?

* The authors acknowledge support from the Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, within which part of this work was performed; and also from the CFEM research center (supported by the Danish Strategic Research Council) within which part of this work was performed.

Before coming to our contribution, let us examine what is known about this problem, and also the motivation for the question. First, the most obvious solution is to take the most efficient known multiplication algorithm, formulate it as a binary circuit and have the box execute this circuit. Using for instance the well known Schönhage-Strassen algorithm, this will lead to communication complexity $O(NlogNloglogN)$ (and $O(\log(N))$ rounds). This can be slightly improved using the algorithm of Führer, [Für07], to $O(N \log N 2^{log^* N})$, and this is the best known complexity using this approach. It is natural to ask if this can be improved, say to linear complexity? Moreover, from a practical point of view, the most advanced multiplication methods are not useful for practical size numbers, which means that in practice the best known circuits will have size $O(N^c)$ for a constant $c > 1$, where c in practice can be expected to be 1.5 (as in Karatsuba's method[KO62]).

However, one might challenge the motivation for the question: after all, implementations of $\mathsf{BB}_{\mathbb{F}}$ are known for any size of field, so one could instead take an implementation over $\mathbb{F}_p$ for a prime $p > ab$, and assume that we get the inputs as values in $\mathbb{F}_p$. Now the multiplication can be done while communicating $O(1)$ field elements (ignoring dependency on the number of players), which is $O(N)$ bits. So why bother optimizing what we can do based on small fields?

A first answer to this is that assuming we can always get the inputs represented "in one piece" is too optimistic. The computations we want to do in practice are not just additions and multiplications, but also comparisons and related operations. These are easy when we have a bit-wise representation but very hard using only native field arithmetic. This means that in many cases, it will make sense to stay with a bit-wise representation throughout and simulate arithmetic using bit-wise operations - provided, of course, that the cost of this manageable, which brings us back to our question: while addition can clearly be done with linear communication, it is not clear if this is possible for multiplication.

Another issue is that for our case (dishonest majority with preprocessing), it is not practical to do computations over (very) large fields: known preprocessing protocols either scale badly with the size of the target field, or are based on OT-extension which naturally gives us implementations over $\mathbb{F}_2$ (and directly not over large fields). More specifically, if we base the preprocessing on some form of homomorphic public-key cryptosystem, as in [DPSZ12], then the parameters of that system grows with the size of the target field, and for the known schemes, the solution becomes impractical for $\mathbb{F}_p$ when p has more than a few hundred bits. If we use OT extension (as suggested in [NNOB12] for malicious adversaries and optimised in several later papers), then the only thing we obtain directly is secure arithmetic on bits. Of course, we can do preprocessing for larger fields by simulating it using Boolean operations, but then once again we run into the problem that multiplication circuits are larger that we would like and certainly at least superlinear. This means that a solution to the question we propose can either be used to enhance the preprocessing of an OT based protocol, or be used to optimise the on-line phase when the preprocessing only considered bits.

Fast integer multiplication has been studied intensively, state of the art methods for computing multiplications classically include Karatsuba [KO62], Toom-Cook [Too63,Coo66], Shoenhage-Strassen [SS71] and Martin Furer [Für07]. In the secure distributed setting early works by Rabin-Rabin-Gennaro [GRR98] followed up by Lory in [Lor09] achieve methods communicating $O(N^2)$ bits. More recent work carried out in practice include [HKS+10] by Schneider et al. where Karasuba's method is shown to be fast than the high-school-method for more than 20 bits using their TASTY framework.

Our contributions We consider Toom-Cook's technique for large integer multiplication and our goal is to utilize this algorithm in the correlated randomness model (aka the preprocessing model) studied in [IKM+13] by Ishai et al. We consider both the semi-honest and malicious security case where $n-1$ of the n players may be corrupted. Our constructions can be realised on top of one of many known protocols [NNOB12,DZ13,DPSZ12,DLT14], as they are all implementations of (variants of) arithmetic black boxes.

Our first result assumes access to a functionality $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$, defined as follows: p is a prime of length $O(\log(N))$ bits, we describe later exactly how the size of p is defined in terms of N, but it is sufficient if its bit length is at least $3\log(N)$. The box offers to store values and do secure arithmetic in both $\mathbb{F}_2$ and $\mathbb{F}_p$, more precisely we use the following notation:

- $[b]_{\mathsf{bits}}$ means that the box holds a bit b in a register. If $P = \sum_{i=0}^{N} P_i 2^i$ for binary values P_i, $[P]_{\mathsf{bits}}$ is shorthand for registers $[P_0]_{\mathsf{bits}}, ..., [P_N]_{\mathsf{bits}}$.
- Player can issue commands to add and multiply binary values, we write this as $[b]_{\mathsf{bits}} \oplus [b']_{\mathsf{bits}} = [b \oplus b']_{\mathsf{bits}}$ and $[b]_{\mathsf{bits}} \cdot [b']_{\mathsf{bits}} = [b \cdot b']_{\mathsf{bits}}$.
- $[a]_p$ means that the box holds a number $a \in \mathbb{F}_p$ in a register.
- Player can issue commands to add and multiply values in $\mathbb{F}_p$, we write this as $[a]_p + [a']_p = [a + a' \bmod p]_p$ and $[a]_p \cdot [a']_p = [a \cdot a' \bmod p]_p$.
- Player can issue commands to add and multiply public constants to stored values in $\mathbb{F}_p$, we write this as $[a]_p + c = [a + c \bmod p]_p$ and $[a]_p \cdot c = [a \cdot c \bmod p]_p$.
- Players can issue a command that causes the box to produce a pair of registers $([r]_{\mathsf{bits}}, [r]_p)$ for a randomly chosen $r \in \mathbb{F}_p$.

It is straightforward to implement $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$ based on known protocols in the pre-processing model, and even the preprocessing can be done quite efficiently: because $\mathbb{F}_p$ is very small, we do not run into the problems with large fields mentioned above.

Definition 1. *A protocol in the preprocessing model that securely implements* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$ *is said to be communication efficient if implementations of commands that do linear operations require no communication, while implementations of multiplication require communication of a constant number of field elements - and finally the command that creates pairs* $([r]_{\mathsf{bits}}, [r]_p)$ *does not require communication.*

Note that known protocols in the preprocessing model are typically communication efficient[1]. The command that produces $([r]_{\mathsf{bits}}, [r]_p)$ without communication is not directly supported by known protocols but can easily be handled by preprocessing a number of such pairs. The purpose of the pairs is to convert between binary and mod-p representation, as we shall see. Of course, if we were willing to assume such pairs for a prime so large that $p > PQ$, then we could convert P and Q to mod-p representation "in one piece" and then trivially do the multiplication with communication complexity $O(N)$ bits. But as discussed above, with known preprocessing protocols, this would either be impractical for large N, or we would need to use a (rather large) Boolean multiplication circuit in the preprocessing.

We show the following (where log() denotes the base-2 logarithm):

Theorem 1. *Assume we are given access to* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$. *There exists an information theoretically maliciously secure n-player protocol* π, *such that* $(\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}, \pi)$ *together implement an enhanced functionality that has all commands of* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$ *as well as a command that computes* $[PQ]_{\mathsf{bits}}$ *from* $[P]_{\mathsf{bits}}$ *and* $[Q]_{\mathsf{bits}}$, *for* N*-bit integers* P, Q. *Using any communication efficient implementation of* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_\mathsf{p}}$ *together with protocol* π *results in an implementation of the new command requiring communication of* $O(N)$ *bits and* $O(log(N))$ *rounds.*

For semihonest security we can make do with a weaker functionality, if we pay for this with a slightly larger complexity:

Theorem 2. *Assume we are given access to* $\mathsf{BB}_{\mathbb{F}_2}$. *There exists an information theoretically semi-honestly secure n-player protocol* π, *such that* $(\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2}, \pi)$ *together implement an enhanced functionality that has all commands of* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2}$ *as well as a command that computes* $[PQ]_{\mathsf{bits}}$ *from* $[P]_{\mathsf{bits}}$ *and* $[Q]_{\mathsf{bits}}$, *for* N*-bit integers* P, Q. *Using any communication efficient implementation of* $\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2}$ *together with protocol* π *results in an implementation of the new command requiring communication of* $O(N)2^{O(\log^*(N))}$ *bits and* $O(\log(N))$ *rounds.*

Note that $\log^*(N)$ is the number of times one needs to iterate the log function on N to get a value less than 1. This grows extremely slowly and is a constant less than 4 or 5 for any practical value of N.

We do not have a result for active security that uses only $\mathsf{BB}_{\mathbb{F}_2}$, we only have Theorem 1 but this is nevertheless useful with known preprocessing protocols: using homomorphic encryption as in [DPSZ12], we can implement preprocessing for both $\mathbb{F}_2$ and $\mathbb{F}_p$, and this is likely to be practical since p has only $\log(N)$ bits, where doing it for field elements of size N bits would not work in practice. If we want maliciously secure preprocessing from OT extension as in [NNOB12], we

[1] In the case of malicious security, multiplication also requires that we check some authentication codes, and these may be larger than the field size. However, the cost of such checks can be amortised over several multiplications such that the extra cost becomes insignificant. See [NNOB12] for details.

can preprocess operations mod p using Boolean circuits, which is no problem, again because p is small.

One should note that our results not only beat the best Boolean multiplication circuits for asymptotic communication complexity, but also have reasonable hidden constants making them useful for practical size numbers. This is not the case for a solution based on a circuit derived from Shönhage-Strassen. Our number of rounds is the same as the best one gets from a Boolean circuit based solution.

As for computational complexity, our construction has $O(N^3)$ complexity out of the box, but we give parameters for which the Fast Fourier Transform can be applied, essentially using Shönhage-Strassen's trick with our distributed Toom-variant obtaining $O(N \log N \log\log N)$ computational complexity. We note that the conjectured lower bound for multiplication with MPC of $O(N \log N)$ is surpassed by our results.

2 Main result

Let P and Q be N-bit numbers over the integers. We assume we have binary representations given: $[P]_{\mathsf{bits}}$ and $[Q]_{\mathsf{bits}}$, respectively. We will now describe a basic version our protocol following essentially follow the structure of the Toom-Cook [Too63,Coo66] algorithm describing the steps *Split*, *Evaluate*, *Multiply*, *Interpolate* and *Carrying* .

2.1 The *split* step

Let $\mathcal{B} = \{0, \ldots, 2^l - 1\}$ with $l = \log N$ and take the algebraic view of P and Q as base $\mathcal{B}$ numbers with κ-digits. That is, we write $P = \sum_{i=0}^{\kappa-1} P_i \mathcal{B}^i$ and like wise for $Q = \sum_{i=0}^{\kappa-1} Q_i \mathcal{B}^i$ where $Q_i, P_i \in \mathcal{B}$ and $\kappa = \frac{N}{\log N}$ [2].

With our base $\mathcal{B}$ view on P and Q it is straightforward to consider their "digits" as coefficients of polynomials:

$$\begin{aligned} P(x) &= P_0 + P_1 x + ... + P_{\kappa-1} x^{\kappa-1} \\ Q(x) &= Q_0 + Q_1 x + ... + Q_{\kappa-1} x^{\kappa-1} \end{aligned} \tag{1}$$

The first idea of the algorithm is now that if we multiply these two polynomials over the integers and evaluate the result in point $\mathcal{B}$, then we will obtain the result PQ that we are after. The next observation is that if we take the view of $P_i, Q_i \in \mathbb{F}_p$ by choosing a prime p of suitable size, then if we multiply the polynomials over $\mathbb{F}_p$ instead, we will get the same coefficients as before. This happens if we avoid overflow modulo p) and therefore we choose it such that $2 \cdot \mathcal{B}^2 N \geq p > \mathcal{B}^2 N$. [3]

[2] For now we assume that $\log N$ divides N evenly and also N is a power of two. Note if this is not the case we can pad with leading zeros to make it so.

[3] Such a prime always exists by the Bertrand - Chebyshev theorem.

Once we have the coefficients we can evaluate the product polynomial in point $\mathcal{B}$ over the integers, and we are done.

The point of multiplying the polynomials over $\mathbb{F}_p$ is that we can do it by evaluating both polynomials in the same set of points, do point wise multiplications, and then interpolate the product polynomial. Since evaluation and interpolation are linear operations, they will be cheap in our scenario.

2.2 The *evaluation* step

We now the functionality we are given ($\mathsf{BB}^{\mathsf{N}}_{\mathbb{F}_2,\mathbb{F}_p}$) for random pairs of values $[R_i]_{\mathsf{bits}}$ and $[R_i]_p$ where $R_i \in \mathbb{F}_p$. The goal is to obtain a $\mathbb{F}_p$-representation of P_i and Q_i, $[P_i]_p$, $[Q_i]_p$ for $i = 0, \ldots, \kappa - 1$. The parties employs a binary addition circuit (while issuing addition and multiplication commands as needed) to compute the value $[P_i - R_i \mod p]_{\mathsf{bits}}$. Note that such a circuit can be designed to have size linear in the bit length of p. This means that in total, $O(N)$ commands in $O(\log\log(N))$ rounds will be issued to compute all the required additions.

Denote $\Delta = P_i - R_i$, which we will open to all parties. All parties will compute $[P_i]_p = [R_i]_p + \Delta$. The parties agree on a $\omega_0, \ldots, \omega_{2\kappa-2}$ evaluation points and compute:

$$\begin{bmatrix} 1 & \omega_0 & \cdots & \omega_0^{2\kappa-2} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_{2\kappa-2} & \cdots & \omega_{2\kappa-2}^{2\kappa-2} \end{bmatrix} \times \begin{bmatrix} [P_0]_p \\ \vdots \\ [P_{\kappa-1}]_p \\ 0 \\ \vdots \\ 0 \end{bmatrix} = [P(\omega_0)]_p, \ldots, [P(\omega_{2\kappa-2})]_p.$$

And likewise for Q such that the parties have $[P(\omega_0)]_p, \ldots, [P(\omega_{2\kappa-2})]_p$ and $[Q(\omega_0)]_p, \ldots, [Q(\omega_{2\kappa-2})]_p$ between them. We emphasize that since our digits/coefficients are now represented in the $[\cdot]_p$ form, evaluating our polynomial in ω_i as depicted above requires *no* further communication.

2.3 The pointwise *multiplication* step

We simply ask the box to compute

$$[P()(\omega_0)]_p \cdot [Q(\omega_0)]_p = [P(\omega_0)Q(\omega_0)]_p, \ldots$$

$$\ldots, [P()(\omega_{2\kappa-2})]_p \cdot [Q(\omega_{2\kappa-2})]_p = [P(\omega_{2\kappa-2})Q(\omega_{2\kappa-2})]_p.$$

Note that, because of our choice of p, no coefficient overflows mod p, i.e., the coefficients have the same integer value they would have if we had multiplied the polynomials over the integers.

2.4 The *interpolation* step

The parties computes the following on their shares modulo p:

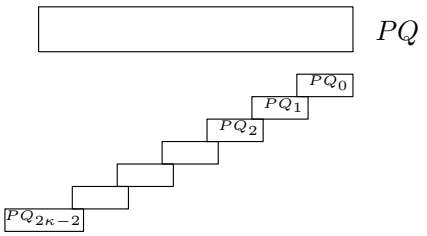
$$\begin{bmatrix} 1 & \omega_0 & \cdots & \omega_0^{2\kappa-2} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \omega_{2\kappa-2} & \cdots & \omega_{2\kappa-2}^{2\kappa-2} \end{bmatrix}^{-1} \times \begin{bmatrix} [P(\omega_0)Q(\omega_0)]_p \\ \vdots \\ [P(\omega_{2\kappa-2})Q(\omega_{2\kappa-2})]_p \end{bmatrix}$$

Yielding representations of the coefficient of the product polynomial $PQ(x) = PQ_0 + PQ_1 x + ... + PQ_{2\kappa-1-1}x^{2^{\kappa-1-1}}$, i.e., we have computed $[PQ_i]_p, i = 0, .., 2\kappa - 2$. This requires only issuing of commands that do linear operations.

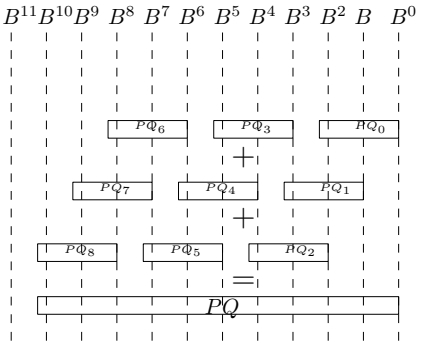
2.5 The *carrying* step

The final result we want can be computed as $PQ(\mathcal{B}) = PQ_0 + \cdots + PQ_{2\kappa-2}\mathcal{B}^{2\kappa-2}$. It can clearly be computed (over the integers) from the coefficients which we have in representation modulo p. But of course trying to to do this computation directly mod p will result in overflow.

Instead we will use a trick based on the fact that the absolute value of each PQ_i is strictly smaller than $\mathcal{B}^3$. In terms of bits we have that PQ_i is shorter than $3 \log N$ bits. Consider Figure 1: On Figure 1a we see that the



(a) Carrying overlapping coefficients.



(b) Carrying laid out as a three way sum, exemplified for $\kappa = 9$.

Fig. 1: How we handle carrying without overflowing modulo p.

product PQ can be "tiled" by the coefficients of $PQ(x)$ but they may overlap. These overlaps are the carries that we need to handle also. What we can do is align the coefficients PQ_i as three numbers, where no two coefficients in any of the numbers overlap. Now we use the pre-processing to provide more pairs $[R_i]_{\mathsf{bits}}, [R_i]_p$ to transform our coefficients to their bits representation. Then we

perform a Three-Way addition circuit as depicted on Figure 1b ending up with $[PQ]_{\mathsf{bits}}$. This requires an addition circuit of size $O(N)$ and depth $O(\log(N))$.

Now, by simple inspection of this and the previous subsections, one can verify that the protocol we have found is as promised in Theorem 1.

3 Semi-honest from bits only

In this section, we show how to apply our main result above to achieve multiplication even given only the functionality $\mathsf{BB}_{\mathbb{F}_2}$, in the semi-honest setting with corruption of $n-1$ of the n players.

We first observe that in this setting, we can define a representation of numbers in $\mathbb{F}_p$, denoted by $[\cdot]_p$ as above, namely for $a \in \mathbb{F}_p$ we will use $[a]_p$ as shorthand for $(a_1, \ldots, a_n)$ where the a_i are chosen at random such that $\sum_i a_i \bmod p = a$ and a_i is held by the i'th player.

It is now simple to get the pairs $([r]_{\mathsf{bits}}, [r]_p)$ required by our main protocol: Each player i chooses and stores $r_i \in \mathbb{F}_p$ at random, this already defines $[r]_p$, where $r = \sum_i r_i \bmod p$. Player i also provides the bits in r_i as input to $\mathsf{BB}_{\mathbb{F}_2}$. We now simulate an Boolean addition circuit that adds the r_i modulo p by issuing the appropriate commands to $\mathsf{BB}_{\mathbb{F}_2}$. As a result we obtain $[r]_{\mathsf{bits}}$. The number of commands depends linearly on the bit length of p (we consider the number of players as a constant here).

This means that the only step of the main algorithm that we cannot do with these extensions is the point-wise multiplication step. For this one, we simply call the same protocol recursively for each point-wise multiplication, and if the bitlength of p is below predefined constant, we will instead simulate the multiplication using a (constant size) Boolean multiplication circuit.

If we let $C(N)$ denote the communication complexity of this solution on N-bit input using a communication efficient underlying protocol, we see that

$$C(N) = c \cdot N + \frac{N}{\log(N)} C(3\log(N)),$$

since it is sufficient to have the bit length of p be $3\log(N)$. By unfolding this expression, one easily sees that we will get a sum of $\log^*(N)$ terms, where the dominating one is $O(N3^{\log^*(N)})$. Thus we have the protocol promised by Theorem 2.

4 Reducing computational complexity

In general the Toom-Cook method uses $O(N^3)$ multiplications in $\mathbb{F}_p$ during the evaluation and interpolation steps. Tailoring our parameters carefully to suit the fast fourier transform we can reduce this to $O(N \log N \log\log N)$ using Shönhage-Strassen.

For a given bit length N we can choose a slightly larger N', on the form:

$$N' = 2^{2^c}, s.t. N < N'$$

Hence we can apply the Fast Fourier transform for Evaluation and Interpolation running in time $O(N \log N \log \log N)$ with a small insignificant overhead from choosing $N' > N$. Notice here that the parties runs the fast fourier transform on their shares both of them and by linearity of the transform the sum of their results yields the correct points or coefficients of our polynomials.

5 Conclusion

We have demonstrated how the well known Toom-Cook algorithm for multiplications can be successfully applied in MPC. We give a variant obtaining communication complexity $O(N2^{\log^* N})$ using only secure binary operations, which is better than what we can hope for performing the best known binary circuit. Using also operations in a small field, we can even get linear communication.

References

[Coo66] Stephen A. Cook. *On the minimum computation time of functions*. PhD thesis, Harvard University, 1966. URL: `http://cr.yp.to/bib/entries.html#1966/cook`.

[DLT14] Ivan Damgård, Rasmus Lauritsen, and Tomas Toft. An empirical study and some improvements of the minimac protocol for secure computation. *IACR Cryptology ePrint Archive*, 2014:289, 2014.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, pages 643–662, 2012.

[DZ13] Ivan Damgård and Sarah Zakarias. Constant-overhead secure computation of boolean circuits using preprocessing. In *TCC*, pages 621–641, 2013.

[Für07] Martin Fürer. Faster integer multiplication. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 57–66, 2007.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified vss and fast-track multiparty computations with applications to threshold cryptography. In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 101–111, New York, NY, USA, 1998. ACM.

[HKS+10] Wilko Henecka, Stefan Kögl, Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. TASTY: tool for automating secure two-party computations. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 451–462, 2010.

[IKM+13] Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, Claudio Orlandi, and Anat Paskin-Cherniavsky. On the power of correlated randomness in secure computation. In *Proceedings of the 10th Theory of Cryptography Conference on Theory of Cryptography*, TCC'13, pages 600–620, Berlin, Heidelberg, 2013. Springer-Verlag.

[KO62] A. Karatsuba and Yu. Ofman. Multiplication of many-digital numbers by automatic computers. *Proceedings of USSR Academy of Sciences*, 145(7):293–294, 1962.

[Lor09] Peter Lory. Secure distributed multiplication of two polynomially shared values: Enhancing the efficiency of the protocol. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 286–291, June 2009.

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings*, pages 681–700, 2012.

[SS71] A. Schönhage and V. Strassen. Schnelle multiplikation großer zahlen. *Computing*, 7(3-4):281–292, 1971.

[Too63] Andrei L. Toom. The complexity of a scheme of functional elements realizing the multiplication of integers. *Soviet Mathematics Doklady*, 3:714–716, 1963.