MPC for $\mathcal{Q}_2$ Access Structures over Rings and Fields

Robin Jadoul[1], Nigel P. Smart[1], and Barry Van Leeuwen[1]

imec-COSIC, KU Leuven, Leuven, Belgium.
`robin.jadoul@esat.kuleuven.be`, `nigel.smart@kuleuven.be`
`barry.vanleeuwen@kuleuven.be`

Abstract. We examine Multi-Party Computation protocols in the active-security-with-abort setting for $\mathcal{Q}_2$ access structures over small and large finite fields $\mathbb{F}_p$ and over rings $\mathbb{Z}_{p^k}$. We give general protocols which work for any $\mathcal{Q}_2$ access structure which is realised by a multiplicative Extended Span Program. We generalize a number of techniques and protocols from various papers and compare the different methodologies. In particular we examine the expected communication cost per multiplication gate when the protocols are instantiated with different access structures.

1 Introduction

Secure multiparty computation (MPC) considers the situation where some set of parties $\mathcal{P}$ come together to compute a function, each with their own inputs. The security requirement is that no party is able to learn more than what the output of this computation and their own input would allow them to. From another perspective, this can be seen as a protocol that emulates a perfectly honest, trusted third party that obtains each party's input, performs the computation, and outputs the result.

We can distinguish different security notions based on the power an adversary can have. One axis along which to distinguish is whether the adversary is active or passive. A passive adversary, also sometimes called *honest but curious*, follows the protocol correctly, but tries to obtain more information from the parts of the transcript of the execution it can see. An active adversary on the other hand, is able to arbitrarily deviate from the protocol. In this situation we either require that the honest parties still obtain the correct output from the function, in which case we say that the protocol is robust, or we require that the honest parties abort the protocol with overwhelming probability, in which case we say the protocol is *actively-secure-with-abort*. In this paper we concentrate on protocols which are actively-secure-with-abort, as they are relatively fast and practical in a large number of situations. Those readers who are interested in robust active security should consult [1, 9].

Another axis to consider is how many or which subsets of parties the adversary can corrupt. If we have n parties then a full threshold adversary is one who is able to corrupt at most $n-1$ parties. In such a situation we can achieve active-security-with-abort, however this comes at the expense of a costly preprocessing phase; see [11, 6] for the case of MPC over finite fields, or over finite rings. Simpler protocols can be obtained if one restricts the adversary to corrupt less parties. The classic restriction is that of threshold adversaries who are allowed to corrupt up to $t < n$ parties. When $t < n/2$ very efficient MPC protocols can be realised, using a variety of methodologies to obtain active-security-with-abort. The natural generalisation of the threshold $t < n/2$ case is that of so-called $\mathcal{Q}_2$ adversary structure. A $\mathcal{Q}_2$ adversary structure is one where the union of no two unqualified sets contains the whole set of players $\mathcal{P}$. For threshold structures the set of unqualified sets are all subsets of $\mathcal{P}$ of size t, thus clearly no two sets can contain all of $\mathcal{P}$ when $t < n/2$. In this paper we will focus on $\mathcal{Q}_2$ access structures, again as they are relatively fast and practical in a large number of situations.

A third axis to consider is the underlying field or ring over which the MPC protocol is implemented. Traditionally the focus has been on MPC protocols over fields $\mathbb{F}_p$, either large finite fields or small ones (in particular $\mathbb{F}_2$). However, recently interest has shifted to also considering finite rings such as $\mathbb{Z}_{p^k}$, and in particular $\mathbb{Z}_{2^k}$. In this setting sometimes, to obtain active security, underlying protocols require the players to work in the extended ring $\mathbb{Z}_{2^{k+s}}$, for some security parameter s, and sometimes this is avoided. In this work we will consider all such possibilities.

The final axis to consider is the precise protocol to use. Almost all practical protocols which are actively-secure-with-abort for $\mathcal{Q}_2$ access structures divide the protocol into two, and sometimes three stages. The first stage, called the offline or pre-processing stage, is function independent and generates various forms of correlated randomness amongst the parties. A second stage, called the online stage, uses the pre-processing to compute the output of the function in a secure manner. Sometimes a third stage, called the post-processing stage, is required to ensure active-security.

The investigation of the combination of the second, third and fourth axes forms the basis of this work. We generalize, where needed, prior works in order to investigate as many prior protocol variants as possible, when instantiated over finite rings or fields. We also generalize results from specific $\mathcal{Q}_2$ access structures to general $\mathcal{Q}_2$ access structures so as to obtain a complete smorgasbord of options. We then analyse the different options, as it is unclear in which situation which protocol is to be preferred (even in the case of finite fields).

Prior Related Work: The majority of the literature has focused on the case where the underlying arithmetic is a finite field. These are often based, for general finite fields and $\mathcal{Q}_2$ access structures, on the classic multiplication protocol of Maurer [17], which works for an arbitrary multiplicative secret sharing scheme. In the case of small finite fields and small numbers of parties, for example $\mathbb{F}_2$ and three players it is common to utilize a multiplication protocol based on replicated secret sharing, which originally appeared in the Sharemind software [4]. The generalisation of this specific multiplication protocol to arbitrary fields and

$\mathcal{Q}_2$-access structures implemented by replicated secret sharing [16], the generalization to an arbitrary $\mathcal{Q}_2$ MSP was done in [18]. Both of these multiplication protocols we shall refer to as KRSW. There is a third passively secure multiplication protocol due to Damgård and Nielsen [10], which we shall refer to as DN multiplication. The DN multiplication protocol is often combined with a "king-paradigm" for opening a sharing, this reduces the total amount of data sent at the expense of doubling the number of rounds. As round complexity has often a bigger impact on execution time than data complexity we assume no king paradigm is used in our protocols[1] Thus before one even considers the various protocols, one has (at least) three base passively secure multiplication protocols to consider. In this work we will concentrate on these three, Maurer or KRSW or DN. The one which is more efficient depends on the precise context as we will show. From these, when using multiplication triples, one can derive a third passively secure multiplication triple which we shall call Beaver multiplication.

In more recent works, research has started to focus on MPC over finite rings, such as $\mathbb{Z}_{p^k}$, and $\mathbb{Z}_{2^k}$ in particular. For many cases, this choice is more natural, as it more closely aligns with the bitwise representation of numbers found in standard computing, and it can enable efficient high level operations such as bit-decomposition (which are very useful in practice). For example, working over $\mathbb{Z}_{2^{64}}$ would closely mimic the behaviour we have on most currently used CPUs. The main problem with working with such rings is the presence of zero-divisors.

A method to avoid the problem of zero-divisors in secret sharing schemes over rings with zero-divisors was presented in the SPD$\mathbb{Z}_{2^k}$ protocol of [6]. Originally, this was presented in the case of a full threshold adversary structure, but the basic trick used applies to any access structure. To avoid the problem of zero divisors when working modulo 2^k, the authors extend (for some protocols) the secret sharing to a large modulus 2^{k+s}, for some statistical security parameter s. This idea was extended to the case of simple $\mathcal{Q}_2$ access structures, using a replicated secret sharing schemes, in [12]. With some of the resulting protocols for $n = 3$ and $n = 4$ parties implemented in the MP-SPDZ framework [14].

Across the many papers on $\mathcal{Q}_2$ MPC we identify three forms of actively secure pre-processing used in the literature, which we generalise[2] to an arbitrary setting of p^k. The first, which we denote by $\mathsf{Offline}_1$, uses a passively secure multiplication protocol to obtain $2 \cdot N$ triples. These are then made actively secure using the classic technique of sacrificing (which effectively uses internally a Beaver multiplication), resulting in an output of N triples. This variant has been used in a number of papers, e.g. [18]. A second variant, which we denote by $\mathsf{Offline}_2$, generates N passively secure triples, and then checks these are correct

[1] Note the kind-paradigm can be used not only in DN multiplication but in any protocol which involves opening shares to all players, as long as suitable additional checks are performed to ensure active security.

[2] There are a few others which we do not consider, as they do not easily fit into our protocol descriptions below. For example the protocol of [2] looks at threshold structures and uses the multiplication protocol of [10] using a king paradigm.

using a different checking procedure, based on the underlying passively secure multiplication protocol of choice. This variant was used in [12].

A third offline variant, which we shall denote by $\mathsf{Offline}_3$, uses a passively secure multiplication protocol to obtain triples in the offline phase. These are then made actively secure using a cut-and-choose method, as opposed to sacrificing. The reason for this is that they are interested in MPC over $\mathbb{F}_2$ and classical sacrificing has a soundness error of one over the field size, and using cut-and-choose allows one to perform an actively secure offline phase without needing to pass to a ring of the form $\mathbb{Z}_{2^k}$. This methodology was presented in [3], and we shall also call this ABF pre-processing. This method seems very well suited to situations when p^k is small as it does not require extending the base ring to $\mathbb{Z}_{p^{k+s}}$.

From these one can derive a number of complete protocol variants. The first variant, which we shall denote $\mathsf{Protocol}_1$, exploits the error-detecting properties of a $\mathcal{Q}_2$ access structure to obtain a protocol which uses an actively secure offline phase, and then uses an online phase based on the classical Beaver multiplication method. Active-security-with-abort is achieved using the error detecting properties of the underlying secret sharing scheme. This has been considered in a number of papers in the case of threshold structures with $(n, t) = (3, 1)$, with the generalisation to arbitrary $\mathcal{Q}_2$ structures in the case of large finite fields being done in [18].

In [12] a three party protocol is presented which makes use of a different methodology, which we generalise to arbitrary $\mathcal{Q}_2$ access structures. Here the online phase is executed optimistically using a passively secure multiplication protocol. The multiplications are then checked to be correct at the end of the protocol using a post-processing phase. Depending on the method used to perform this checking, we can either generate auxiliary, passively secure triples in an offline phase, that can be used in a form of sacrificing in the post-processing phase (which we dub $\mathsf{Protocol}_2$), or we can completely remove the need for a preprocessing step (which we dub $\mathsf{Protocol}_3$).

The paper [3] also uses an optimistic passively secure online phase with a post-processing step, but combines this with an actively secure offline phase. By doing this the post-processing check is always checking possibly incorrect multiplications (from the online phase) against known-to-be-correct multiplications (from the offline phase). This means the post-processing check can be done using a method which is close to that of classical sacrificing, without the need to worry about the small field size. We call this variant $\mathsf{Protocol}_4$.

The final protocol variant we consider, which we dub $\mathsf{Protocol}_5$, comes from [5]. In this paper the authors dispense with the offline phase, and instead generate a shared MAC-key $[\alpha]$, a bit like in SPDZ, and evaluate the circuit on both $[x]$ and $[\alpha \cdot x]$ using a passively secure multiplication protocol. Thus, in some sense, the circuit is evaluated twice in the online phase. The correctness of the evaluation is then established using the MAC-Check protocol from the SPDZ protocol. Thus there is a post-processing step, but it is relatively light-weight, however the online phase is more expensive than other techniques.

We summarize these in five protocol variants in Table 1 as a means for the reader to maintain a quick overview as they read the paper.

	Offline Phase		Online	Post-Processing	
Protocol	Passive	Active	Phase	Heavy	Light
$\mathsf{Protocol}_1$	-	✓	Beaver	-	-
$\mathsf{Protocol}_2$	✓	-	Passive	✓	-
$\mathsf{Protocol}_3$	-	-	Passive	✓	-
$\mathsf{Protocol}_4$	-	✓	Passive	✓	-
$\mathsf{Protocol}_5$	-	-	$2 \times$ Passive	-	✓

Table 1. Summary of our five protocol variants. A "heavy" post-processing phase denotes a phase akin to sacrificing, where as a "light" post-processing denotes a phase akin to SPDZ-like MAC checking. A Passive online phase refers to an online phase using either Maurer or KRSW multiplication.

Our Contribution: In this work we unify all these protocols; in prior work they may have been presented for finite fields, or for rings of the form $\mathbb{Z}_{2^k}$, or for specific access structures. We consider, in all cases, the general case of MPC over rings of the form $\mathbb{Z}_{p^k}$; i.e. where we consider both the case of $k = 1$, large k, small p, and large p in one go. Our methodology applies to all multiplicative $\mathcal{Q}_2$ access structures over such rings. To do so we utilize the language of Extended Span Programs, ESPs, introduced in [13]. This allows us to consider not only replicated access structures, but also access structures coming from Galois Ring constructions. By considering such Galois Ring constructions as an ESP, we can maintain working over $\mathbb{Z}_{p^k}$ without the need to worry about complications arising from the Galois Ring.

We first show how one can create the necessary ESPs for a specific access structure, by constructing an associated MSP over the field $\mathbb{F}_p$ and then lifting it to $\mathbb{Z}_{p^k}$ in a trivial manner. This preserves the access structure, but it does not always preserve multiplicity (see [1] for a relatively contrived counter example). For all "natural" MSPs one might encounter in practice (arising from Shamir or Replicated secret sharing) the lifting does preserve multiplicity. In any case if the resulting ESP over $\mathbb{Z}_{p^k}$ is not multiplicative, it can be extended to a multiplicative ESP in the standard manner[3].

We show that the error-detection properties of [18] apply in this more generalized context of finite rings. This allows us to reduce the communication cost in our protocols for ESPs. Note the error-detection properties exploited in [18] are the precise generalization to arbitrary $\mathcal{Q}_2$ MSPs of the classical check for correctness performed in threshold systems for $(n, t) = (3, 1)$ based on replicated sharing.

[3] This is a standard result for MSPs over fields, but it is easily extended to ESPs over finite rings.

We also show that the trick of modulus extension from $\mathbb{Z}_{p^k}$ to $\mathbb{Z}_{p^{k+s}}$ also works in general, and we combine it with other tricks. For example we use Schwarz-Zippel over Galois rings to allow greater batching, and modulus extension even in the case of checking over finite fields. Indeed we show that one can also utilize modulus extension to avoid the problems with sacrificing when $k = 1$ and p is small. However, this comes at the expense of requiring to work modulo p^{k+s} and not working modulo p^k, which may be a problem in some instances (for example in the interesting case of $p^k = 2$). Thus our multiplication checking procedures in Section 3 generalise a number of earlier results, and unify various approaches. Note, that depending on the underlying protocol choice such modulus extensions may not be needed.

We finally examine the smorgasbord of options for the offline, online and post-processing which we outlined above in this general context and examine the various benefits and tradeoffs which result. Our cost metrics in this matter are the total number of rounds of communication, as well as the total amount of data sent per multiplication[4]. We consider the case where the user is interested in minimizing the total cost (i.e. the combined cost of all three phases), as well as the case where the user is interested in minimizing the costs of the online and post-processing phases only (i.e. where the user assumes that the offline phase can be done overnight for example and is not an important consideration).

2 Preliminaries

2.1 Notation

We let $\mathbb{F}$ denote a general finite field, and R denote a general finite commutative ring. We let $\mathbb{F}_p$ denote the specific finite field of p elements, and $\mathbb{Z}_{p^k}$ denote the ring of integers modulo p^k. For two sets X, Y we write $X \subset Y$ if X is a proper subset and $X \subseteq Y$ if X is not necessarily proper. For a set B, we denote by $a \leftarrow B$ the process of drawing a from B with a uniform distribution on the set B. For a probabilistic algorithm A, we denote by $a \leftarrow A$ the process of assigning a the output of algorithm A; with the underlying probability distribution being determined by the random coins of A.

For a vector $\mathbf{x}$ we let $\mathbf{x}^{(i)}$ denote it ith component, and for two vectors $\mathbf{x}$ and $\mathbf{y}$ of the same length we let $\langle \mathbf{x}, \mathbf{y} \rangle$ denote the dot-product, unless otherwise noted. We let $M_{n \times m}(K)$, where $K = \mathbb{F}$ or $K = R$, be the set of all matrices with n rows and m columns. For $M \in M_{n \times m}(K)$ denote the transpose by M^T. We let $\ker(M)$ to denote the subspace of K^m which maps to $\mathbf{0}$ under left multiplication by M, and we let $\mathrm{Im}(M)$ to be the subspace of K^n which is the image of all elements in K^n upon left multiplication by M. If V is a subspace of K^r for some r, we let $V^{\perp} = \{\mathbf{w} \in K^r \mid \forall \mathbf{v} \in V : \langle \mathbf{w}, \mathbf{v} \rangle = 0\}$ denote the orthogonal complement. Moreover, we let $\mathbf{0}$ and $\mathbf{1}$ be the all zero and all one vector of appropriate dimension (defined by the context unless explicitly specified) and let

[4] Note, as MPC protocols do not usually work *in practice* over arithmetic circuits this is only an approximation of the cost of the various options.

$\mathbf{e}_i$ be the ith canonical basis vector, that is $\mathbf{e}_i^{(j)} = \delta_{i,j}$ where δ is the Kronecker Delta.

2.2 Monotone and Extended Span Programs

As is standard we can associate linear secret sharing schemes over fields with Monotone Span Programs. In [13] these definitions are extended to linear secret sharing schemes over finite rings, such as $\mathbb{Z}_{p^k}$, with the associated structure being called an Extended Span Program. We recap on the relevant definitions here.

Access Structures: The set of parties that the adversary can corrupt is drawn from an access structure (Γ, Δ). The set Γ is the set of all qualified sets, whilst Δ is the set of all unqualified sets. The access/adversary structure is assumed to be monotone, i.e. if $X \subset X'$ and $X \in \Gamma$, then $X' \in \Gamma$ and if $X \subset X'$ and $X' \in \Delta$ then $X \in \Delta$, and we assume $\Gamma \cap \Delta = \emptyset$. We are only interested in this paper in access structures which are $\mathcal{Q}_2$:

Definition 2.1 ($\mathcal{Q}_2$ Access Structure). *Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be a set of parties, with access structure (Γ, Δ), then (Γ, Δ) is said to be a $\mathcal{Q}_2$ access structure if*

$$P \neq A \cup B \text{ for all } A, B \in \Delta.$$

In other words: An access structure (Γ, Δ) is $\mathcal{Q}_2$, if for any two sets in Δ the union of those sets does not cover $\mathcal{P}$. An access structure is called complete if for any $Q \in \Gamma$ it holds that $\mathcal{P} \backslash Q \in \Delta$ and vice versa. In this paper we will only consider complete access structures.

Monotone Span Programs over Fields: Using this notation, the definition of a Monotone Span Program follows.

Definition 2.2. *A Monotone Span Program (MSP), denoted $\mathcal{M}$, is a quadruple $(\mathbb{F}, M, \varepsilon, \varphi)$, where $\mathbb{F}$ is a field, $M \in M_{m \times d}(\mathbb{F})$ is a full-rank matrix for some m and $d \leq m$, $\varepsilon \in \mathbb{F}^d$ is an arbitrary non-zero vector called the target vector, and $\varphi : [m] \to \mathcal{P}$ is a surjective map of the rows of M to the parties in $\mathcal{P}$. The size of $\mathcal{M}$ is defined to be m, the number of rows of the matrix M.*

Given a set of parties $\mathcal{S} \subseteq \mathcal{P}$, the submatrix $M_{\mathcal{S}}$ is the matrix whose rows are indexed by the set $\{i \in [m] : \varphi(i) \in \mathcal{S}\}$. Similarly $\mathbf{s}_{\mathcal{S}}$ is the vector whose rows are indexed by the same set. We also define the supp-mapping, which maps the rows of a matrix M to a player in $\mathcal{P}$. Formally this is defined as $\mathsf{supp} : \mathbb{F}^d \to 2^{[d]}$ with $\mathbf{s} \mapsto \{i \in [d] : \mathbf{s}^{(i)} \neq 0\}$.

Extended Span Programs over Rings: In this paper we are not only interested in the Monotone Span Programs, but also their extensions to finite rings, which are known as Extended Span Programs, [13]. An Extended Span Program (ESP) over a ring R is a tuple $\mathcal{M} = (R, M, \varepsilon, \varphi)$ where $M \in M_{m \times d}(R)$ is a full-rank matrix for some m and $d \leq m$, $\varepsilon \in R^d$ is an arbitrary non-zero vector called the target vector, and $\varphi : [m] \to \mathcal{P}$ is a surjective map of the rows of M to the parties in $\mathcal{P}$.

Definition 2.3. *An ESP $\mathcal{M}$ is to compute an access structure (Γ, Δ) if for every set $A \subset 2^{\mathcal{P}}$ it holds that*

$$A \in \Gamma \Rightarrow \varepsilon \in \operatorname{Im}(M_A^T), \tag{1}$$

$$A \notin \Gamma \Rightarrow \exists \mathbf{v} \in \ker(M_A) \subset R^d : \langle \varepsilon, \mathbf{v} \rangle \in R^*. \tag{2}$$

For the rest of this paper we will only be considering MSPs over finite fields $\mathbb{F}_p$, or ESPs over the finite ring $\mathbb{Z}_{p^k}$. Let $\mathcal{P} = \{P_1, \ldots, P_n\}$ be the set of parties involved in our protocols. To implement our MPC functionality over $\mathbb{Z}_{p^k}$ we will utilize an ESP $(\mathbb{Z}_{p^k}, M, \varepsilon, \varphi)$ given by a matrix $M \in \mathbb{Z}^{m \times d}$, such that $M = M \pmod{p}$ (i.e. the entries of M are in the range $[0, \ldots, p)$), such that to share a value $x \in \mathbb{Z}_{p^k}$ one generates a vector $\mathbf{k} \in \mathbb{Z}_{p^k}^d$ such that $\langle \varepsilon, \mathbf{k} \rangle = x \pmod{p^k}$ and then compute the share values $\mathbf{s} = M \cdot \mathbf{k}$. The entries of $\mathbf{s}$ are passed to the players depending on the value of the function $\varphi : [m] \to \mathcal{P}$. i.e. player P_i gets $\mathbf{s}^{(j)}$ if $\varphi(j) = i$. Such a sharing $x \in \mathbb{Z}_{p^k}$ of a value will be denoted by $[x]_k$, note the subscript k which will be used to keep track of which ring we are considering at any given point.

2.3 Linear Secret Sharing Schemes Induced from MSPs and ESPs

When you have a Monotone/Extended Span Program it induces a Linear Secret Sharing Scheme (LSSS) using the method in Figure 1. Recombination works for qualified sets $A \in \Gamma$, since if A is qualified there exists a recombination vector λ_A such that $M_A^T \cdot \lambda_A = \varepsilon$, by requirement (1) of the MSP. Hence

$$\langle \lambda_A, \mathbf{s}_A \rangle = \langle \lambda, \mathbf{s} \rangle = \langle \lambda, M \cdot \mathbf{x} \rangle = \langle M^T \cdot \lambda, \mathbf{x} \rangle = \langle \varepsilon, \mathbf{x} \rangle = s.$$

Conversely, if $A \notin \Gamma$ then A is unqualified, hence by requirement (2) of the ESP, there is no λ that allows for reconstruction. We note that the reconstruction step 2 can be relatively expensive for large MSPs, i.e. those with large m. Thus it is common to only send "just enough" information to each player in order to allow reconstruction. How this is done in a manner which prevents active attacks is discussed in the full version.

Multiplicative Linear Secret Sharing Scheme A secret sharing scheme induced from a MSP/ESP is by definition linear, i.e. one can compute arbitrary linear functions of secret shared values without interaction. $\mathcal{Q}_2$ access structures are interesting as they allow us to also multiply secret shared values, but using interaction, if the underlying LSSS is multiplicative.

Recall a vector $\mathbf{s} = (s_i) = M \cdot \mathbf{k}$ is some sharing of a value s if we have that $\langle \varepsilon, \mathbf{k} \rangle = s$, with the shares distributed to party P_i being $\mathbf{s}_i = (s_j)_{\varphi(j)=i}$. We let the total number of shares held by party P_i be given by n_i. The local *Schur*

Induced LSSS from an MSP/ESP

Given a Monotone/Extended Span Program, $\mathcal{M} = \{\mathbb{Z}_{p^k}, M, \varepsilon, \varphi\}$ and a secret s, distribution and reconstruction for the associated secret sharing scheme are as follows:

Distribution:

1. Sample $\mathbf{x} \leftarrow \mathbb{Z}_{p^k}^d$ under the condition that $\langle \mathbf{x}, \varepsilon \rangle = s$.
2. Compute $\mathbf{s} = M \cdot \mathbf{x}$, such that $\mathbf{s} = (s_1 s_2 \ldots s_n)$ and distribute each s_i to the party indicated by $\varphi(i)$, such that each party P_j has the vector

$$\mathbf{s}_{P_j} = \begin{cases} s_i & \varphi(i) = P_j \\ 0 & \text{otherwise} \end{cases}$$

Reconstruction: Let $A \in \Gamma$ be a qualified set of players:

1. Define λ_A such that $M_A^T \cdot \lambda_A = \varepsilon$.
2. Each player $P_i \in A$ sends their shares to all other $P_j \in A$ and computes $\mathbf{s}_A = \sum_{P_i \in A} s_{P_i}$.
3. Compute $s^* = \langle \mathbf{s}_Q, \lambda_Q \rangle$.
4. Return s^*.

Fig. 1. Induced LSSS from a Monotone/Extended Span Program.

product of two sharings $\mathbf{x}_i$ and $\mathbf{y}_i$ of values x and y for party P_i are the n_i^2 terms given by $\mathbf{x}_i \otimes \mathbf{y}_i$, i.e. the terms $p_{i,j} = \mathbf{x}_i^{(v)} \cdot \mathbf{y}_i^{(v')}$ for $j = 1, \ldots, n_i^2$ and v, v' range over all values for which $\varphi(v) = \varphi(v') = i$. An MSP is said to be *multiplicative* if there are constants $\mu_{i,j}$ for $i = 1, \ldots, n$ and $j = 1, \ldots, n_i^2$ such that

$$x \cdot y = \sum_{i,j} \mu_{i,j} \cdot p_{i,j} \qquad (3)$$

for all valid sharings of x and y. By abuse of notation we shall refer to the MSP/ESP being multiplicative, and not just the induced LSSS.

Many "natural" MSPs/ESPs computing $\mathcal{Q}_2$ access structures are multiplicative, i.e. those arising from Shamir secret sharing, or replicated sharing. It is well known, see [8], that when you have an non-multiplicative MSP over a field that computes a $\mathcal{Q}_2$ access structure then it can be made multiplicative with only a small expansion of the dimensions of M. In the full version we prove the following theorem, generalising this result to ESPs over $\mathbb{Z}_{p^k}$,

Theorem 2.1. *There exists an algorithm which, on input of a non-multiplicative ESP $\mathcal{M}$ over $\mathbb{Z}_{p^k}$ computing a $\mathcal{Q}_2$ access structure (Γ, Δ) outputs a multiplicative ESP $\mathcal{M}'$ computing Γ and of size at most $4 \cdot |\mathcal{M}|$. This algorithm is effective if* $\ker(M^T)$ *admits a basis.*

3 Multiplication Check

We present various protocols which allow one to verify that a set of passively secure multiplications are indeed correct. In the context of generating triples, we note that, we are unable to "lift" a valid triple modulo p^k to a valid triple modulo p^{k+v}. Thus, if one needs to perform a check modulo p^{k+s}, one needs to generate the passively secure multiplication triples modulo the larger modulus first, even if one is only interested in computation modulo p^k.

We assume that the desired security level is 2^κ, i.e. the probability that an adversary can pass off an incorrect passively secure multiplication as correct should be $2^{-\kappa}$. To ensure this we define four (integer) parameters (u, v, w, B) for our protocols defined by, where $B_z = 0$ unless $B \neq 1$ in which case we set $B_z = 1$.

$$\begin{aligned} u &= \lceil (\kappa + B_z)/\log_2 p \rceil \\ v &= u - 1, \\ 1 \leq B &\leq 1 + (p^w - 1)/2^{\kappa + B_z}. \end{aligned}$$

The value u defines the size of the challenge space in our protocols, the value v defines how much bigger a modulus we need to work with, the value w defines the degree of any extension needed to allow the Schwartz-Zippel Lemma to apply, using a set S of size $p^w - 1$, whilst B defines the bucket size of the check (equivalently the degree of the polynomial used in the Schwartz-Zippel Lemma).

Our methods here are a natural generalisation of the methods given in [12, 2] which are themselves based on ideas used in [6]. We note for the case of $k = 1$ and a small prime p the following protocols produce more efficient "sacrificing" steps than the "traditional" method of repeating the protocol $\kappa/\log_2 p$ times.

3.1 $\mathsf{MultCheck}_1$

The first protocol, often called sacrifice, takes a set of N passively secure multiplication triples $([x_i]_{k+v}, [y_i]_{k+v}, [z_i]_{k+v})$, and checks whether indeed $z_i = x_i \cdot y_i \pmod{p^k}$, using another set of passively secure multiplication triples $([a_i]_{k+v}$, $[b_i]_{k+v}$, $[c_i]_{k+v})$. The "unchecked" triples $([a_i]_{k+v}, [b_i]_{k+v}, [c_i]_{k+v})$ need to be discarded at the end of the protocol (thus the term sacrificing). The output of the protocol is either an abort signal, or a set of N "actively" secure triple $([x_i]_k, [y_i]_k, [z_i]_k)$. The protocol is described in Figure 2 and is based internally on the Beaver multiplication protocol. For ease of exposition we assume B exactly divides N in the protocol, this can easily be removed.

The number of calls to the procedure $\mathsf{OpenToAll}(\cdot)$, which is the main cost of the protocol is given by $2 \cdot N + N \cdot w/B$, and the number of rounds of communication (for the $\mathsf{OpenToAll}$ calls) is bounded by two (if one executes the main j-loop in parallel). This means the communication cost, per output triple, is equal to the communication of $2 + w/B$ executions of $\mathsf{OpenToAll}(\cdot)$. In practice one would try to select w/B to be as small as possible. In such a situation we can treat the cost as two calls to $\mathsf{OpenToAll}(\cdot)$.

```
The Protocol MultCheck_1

Input: $([x_i]_{k+v}, [y_i]_{k+v}, [z_i]_{k+v})_{i=0}^{N-1}$ and $([a_i]_{k+v}, [b_i]_{k+v}, [c_i]_{k+v})_{i=0}^{N-1}$.
Output: abort or $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$.

1. Let $R$ denote a degree $w$ Galois ring over $\mathbb{Z}_{p^{k+v}}$.
2. Let $S$ denote the set from the Schwartz-Zippel Lemma of size $p^w - 1$.
3. $t \leftarrow \mathcal{F}_{\text{AgreeRandom}}(\mathbb{Z}_{p^u})$.
4. For $j \in [0, \ldots, N/B)$ do
   (a) $r \leftarrow \mathcal{F}_{\text{AgreeRandom}}(S)$.
   (b) For $i \in [0, \ldots, B)$ do
       i. $[\rho_i]_{k+v} \leftarrow t \cdot [a_{j \cdot B+i}]_{k+v} - [x_{j \cdot B+i}]_{k+v}$.
       ii. $[\sigma_i]_{k+v} \leftarrow [b_{j \cdot B+i}]_{k+v} - [y_{j \cdot B+i}]_{k+v}$.
   (c) $(\rho_i)_{i=0}^{B-1} \leftarrow (\mathsf{OpenToAll}([\rho_i]_{k+v}))_{i=0}^{B-1}$.
   (d) $(\sigma_i)_{i=0}^{B-1} \leftarrow (\mathsf{OpenToAll}([\sigma_i]_{k+v}))_{i=0}^{B-1}$
   (e) $[\tau]_{k+v} \leftarrow 0$.
   (f) For $i \in [0, \ldots, B)$ do
       i. $[d_i]_{k+v} \leftarrow t \cdot [c_{j \cdot B+i}]_{k+v} - [z_{j \cdot B+i}]_{k+v} - \sigma_i \cdot [x_{j \cdot B+i}]_{k+v} - \rho_i \cdot [y_{j \cdot B+i}]_{k+v} - \sigma_i \cdot \rho_i$.
       ii. $[\tau]_{k+v} \leftarrow [\tau]_{k+v} + r^i \cdot [d_i]_{k+v}$.
   (g) $\tau \leftarrow \mathsf{OpenToAll}([\tau]_{k+v})$
   (h) If $\tau \neq 0 \pmod{p^{k+v}}$ output abort and stop.
5. For $i \in [0, \ldots, N)$ do
   (a) $[x_i]_k \leftarrow [x_i]_{k+v} \pmod{p^k}$, $[y_i]_k \leftarrow [y_i]_{k+v} \pmod{p^k}$, $[z_i]_k \leftarrow [z_i]_{k+v} \pmod{p^k}$.
6. Output $([x]_k, [y]_k, [z]_k)_{i=1}^{N}$.
```

Fig. 2. The Protocol MultCheck$_1$

In the case of $k = 1$ and a large prime p, the values $w = 1$, $u = 1$, $v = 0$ and $B = 1$ give rise to *exactly* the traditional sacrifice protocol from SPDZ. However, for such large p, we could choose $w = 2$ and allow B to be sufficiently big, without needing an overly large amount of triples to check at once. Thus by utilizing our modified protocol one can achieve an improvement on the classical SPDZ sacrificing protocol. So for large p, for the classical SPDZ sacrifice, we have $w/B = 1$ and hence the cost is three calls to OpenToAll$(\cdot)$, but for our protocol we can achieve two calls to OpenToAll$(\cdot)$.

As long as we perform the calls to AgreeRandom only *after* the adversary had a chance to influence the triples, and the adversary is fully committed to any errors introduced in them, we can use the same random values for t and r over all instantiations. The practical advantage of this is that the data cost of these calls can then be amortized over all these executions, and we can consider it negligible. Due to the commit-reveal nature of the AgreeRandom sub-protocol, however, we still need to take a cost of two rounds of communication into account. All invocations of AgreeRandom that we need to generate the required t and r values can be executed in parallel, so the number of rounds we need does not grow as the number of times MultCheck$_1$ is executed grows.

In the full version we prove the following theorem which is an adaption of similar results in [6] (especially Claim 6 in that paper) and the papers [12, 2], but we have generalized the method to arbitrary p and also the case of potentially small k.

Lemma 3.1. *In the presence of an active adversary, who can introduce arbitrary additive errors into the input triples, the protocol* $\mathsf{MultCheck}_1$ *will output an invalid multiplication triple with probability* $(B-1)/(p^w-1)+p^{-u} \le 2^{-\kappa}$.

The Protocol $\mathsf{MultCheck}_1'$

```
Input: $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$ and $([a_i]_k, [b_i]_k, [c_i]_k)_{i=0}^{N-1}$.
Output: $\mathsf{abort}$ or $\mathtt{OK}$.

1. Let $R$ denote a degree $w$ Galois ring over $\mathbb{Z}_{p^k}$.
2. Let $S$ denote the set from the Schwartz-Zippel Lemma.
3. For $j \in [0, \dots, N/B)$ do
   (a) $r \leftarrow \mathcal{F}_{\text{AgreeRandom}}(S)$.
   (b) For $i \in [0, \dots, B)$ do
       i. $[\rho_i]_k \leftarrow [a_{j \cdot B+i}]_k - [x_{j \cdot B+i}]_k$.
       ii. $[\sigma_i]_k \leftarrow [b_{j \cdot B+i}]_k - [y_{j \cdot B+i}]_k$.
   (c) $(\rho_i)_{i=0}^{B-1} \leftarrow (\mathsf{OpenToAll}([\rho_i]_k))_{i=0}^{B-1}$.
   (d) $(\sigma_i)_{i=0}^{B-1} \leftarrow (\mathsf{OpenToAll}([\sigma_i]_k))_{i=0}^{B-1}$
   (e) $[\tau]_k \leftarrow 0$.
   (f) For $i \in [0, \dots, B)$ do
       i. $[d_i]_k \leftarrow [c_{j \cdot B+i}]_k - [z_{j \cdot B+i}]_k - \sigma_i \cdot [x_{j \cdot B+i}]_k - \rho_i \cdot [y_{j \cdot B+i}]_k - \sigma_i \cdot \rho_i$.
       ii. $[\tau]_k \leftarrow [\tau]_k + r^i \cdot [d_i]_k$.
   (g) $\tau \leftarrow \mathsf{OpenToAll}([\tau]_k)$
   (h) If $\tau \neq 0 \pmod{p^k}$ output $\mathsf{abort}$ and stop.
4. Output $\mathsf{OK}$.
```

Fig. 3. The Protocol $\mathsf{MultCheck}_1'$

3.2 $\mathsf{MultCheck}_1'$

We will also use the $\mathsf{MultCheck}_1$ protocol in the case where we are already guaranteed that the auxiliary triples $([a_i]_k, [b_i]_k, [c_i]_k)_{i=0}^{N-1}$ are correct, and we have $v = 0$ and $u = k$, and we are simply checking whether the passively secure triples $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$ are correct. We refer to this special case as $\mathsf{MultCheck}_1'$ and it is presented in Figure 3. The round complexity is the same as that of $\mathsf{MultCheck}_1$, except for the output, although now we can operate modulo p^k only, without needing to extend to working modulo p^{k+s}. In this special case we obtain the following result,

Lemma 3.2. *In the presence of an active adversary, who can introduce arbitrary additive errors into the input triples* $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$*, but not the input triples* $([a_i]_k, [b_i]_k, [c_i]_k)_{i=0}^{N-1}$*, the protocol* $\mathsf{MultCheck}_1'$ *will output* $\mathtt{OK}$ *incorrectly with probability* $(B-1)/(p^w-1) \le 2^{-(\kappa+B_z)}$.

3.3 MultCheck$_2$

Our third protocol comes from a combination of ideas from [6] and [15]. Instead of consuming previously produced multiplication triples (which themselves require a passively secure multiplication to produce) this second variant makes direct use of a passively secure multiplication protocol PassMult; which can be any of MaurerMult, KRSWMult or DNMult. The protocol, called MultCheck$_2$, is described in Figure 4. The argument for security is roughly the same as that for protocol MultCheck$_1$. These protocols use a PRSS functionality $\mathcal{F}_{\text{PRSS}}$ which is defined in the full version.

```
The Protocol MultCheck$_2$

Input: $([x_i]_{k+v}, [y_i]_{k+v}, [z_i]_{k+v})_{i=0}^{N-1}$.
Output: abort or $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$.

1. Let $R$ denote a degree $w$ Galois ring over $\mathbb{Z}_{p^{k+v}}$.
2. Let $S$ denote the set from the Schwartz-Zippel Lemma.
3. For $i \in [0, \ldots, N)$ do
   (a) $[a_i]_{k+v} \leftarrow \mathcal{F}_{\text{PRSS}}(k+v)$.
   (b) $[c_i]_{k+v} \leftarrow \mathsf{PassMult}([a_i]_{k+v}, [y_i]_{k+v})$.
4. $t \leftarrow \mathcal{F}_{\text{AgreeRandom}}(\mathbb{Z}_{p^u})$.
5. For $j \in [0, \ldots, N/B)$ do
   (a) $r \leftarrow \mathcal{F}_{\text{AgreeRandom}}(S)$.
   (b) For $i \in [0, \ldots, B)$ do
       i. $[\rho_i]_{k+s} \leftarrow t \cdot [x_{j \cdot B+i}]_{k+v} + [a_{j \cdot B+i}]_{k+v}$.
   (c) $(\rho_i)_{i=0}^{B-1} \leftarrow (\mathsf{OpenToAll}([\rho_i]_{k+v}))_{i=0}^{B-1}$.
   (d) $[\tau]_{k+v} \leftarrow 0$.
   (e) For $i \in [0, \ldots, B)$ do
       i. $[\tau]_{k+v} \leftarrow [\tau]_{k+v} + r^i \cdot (t \cdot [z]_{k+v} + [c]_{k+v} - \rho \cdot [y]_{k+v})$.
   (f) $\tau \leftarrow \mathsf{OpenToAll}([\tau]_{k+s})$
   (g) If $\tau \neq 0 \pmod{p^{k+v}}$ output abort and stop.
6. For $i \in [0, \ldots, N)$ do
   (a) $[x_i]_k \leftarrow [x_i]_{k+v} \pmod{p^k}$, $[y_i]_k \leftarrow [y_i]_{k+v} \pmod{p^k}$, $[z_i]_k \leftarrow [z_i]_{k+v}$
       $\pmod{p^k}$.
7. Output $([x_i]_k, [y_i]_k, [z_i]_k)_{i=0}^{N-1}$.
```

Fig. 4. The Protocol MultCheck$_2$

3.4 MacCheck

Our final protocol is the generalization of the MacCheck protocol from [11] to our situation. The protocol checks, for an input of a single secret shared value $[\alpha]_{k+v}$ and a series of pairs of secret shared values $([x_i]_{k+v}, [y_i]_{k+v})_{i=0}^{N-1}$, whether we have $y_i = \alpha \cdot x_i \pmod{p^{k+v}}$, or whether y_i is invalid up to an additive error. Note, unlike the MacCheck protocol from [11] we are not checking the MACs of opened values, but checking the consistency of pairs of unopened values with respect to the shared MAC key α, as such it is closer to the verification stage of the protocol in [5]. We note that with the instantiation given in Figure 5, this checking procedure "burns" the value $[\alpha]_{k+v}$, thus this does not allow for reactive computations. In [5] it is shown how to avoid this problem for *specific* secret sharing schemes. The protocol is given in Figure 5

The Protocol MacCheck

Input: $[\alpha]_{k+v}$ and $([x_i]_{k+v}, [y_i]_{k+v})_{i=0}^{N-1}$.
Output: abort or OK.

```
1. For $i \in [0, N)$ do $r_i \leftarrow \mathcal{F}_{\text{AgreeRandom}}(\mathbb{Z}_{p^u})$.
2. $[u] \leftarrow \sum_{i=0}^{N-1} r_i \cdot [x_i]_{k+v}$.
3. $[v] \leftarrow \sum_{i=0}^{N-1} r_i \cdot [y_i]_{k+v}$.
4. $[c]_{k+v} \leftarrow \mathcal{F}_{\text{PRSS}}(k+v)$.
5. $\alpha \leftarrow \mathsf{OpenToAll}([\alpha]_{k+v})$.
6. $[t]_{k+v} \leftarrow [v]_{k+v} - \alpha \cdot [u]_{k+v}$.
7. $[s]_{k+v} \leftarrow \mathsf{PassMult}([t]_{k+v}, [c]_{k+v})$.
8. $s \leftarrow \mathsf{OpenToAll}([s]_{k+v})$.
9. If $s = 0$ then return OK, else return abort.
```

Fig. 5. The Protocol MacCheck

Lemma 3.3. *Protocol* MacCheck *in Figure 5 on input of an invalid set of pairs* $([x_i]_{k+v}, [y_i]_{k+v})_{i=0}^{N-1}$ *will return* OK *with probability less than* $2^{-\kappa}$*. Where a pair being invalid means that* $y_i = \alpha \cdot x_i + e_i$*, for an* e_i *known to the adversary with* $e_i \neq 0 \pmod{p^k}$*.*

3.5 Summary

We summarize the costs of various protocols in Table 2 for a general ESP over $\mathbb{Z}_{p^k}$. These are given in terms of the row m and column d dimensions of the matrix generating the underlying ESP, the number of parties n, and the parameters w and B used in the protocols above. We let $|\mathbf{s}_i|$ denote the share size of player P_i for the given ESP. The data column indicates the total amount of data sent for *all* players[5] as a multiple of the underlying secret shared data size (i.e. either $k \cdot \log_2 p$ or $(k+v) \cdot \log_2 p$); we ignore rounds/data to check the running hash values H as these are amortized over many sub-protocol executions. A $\star$ in the table indicates that the value depends highly on the specific ESP, and thus a formula is hard to present. The cost $\star_1$ of OpenToAll is generally $n \cdot d - m$ for an MSP with no redundancy, but it can be larger than this if the MSP has more redundancy than necessary.

[5] i.e. not the per-player amount

We present three lines corresponding to $\mathsf{MultCheck}_2$ and $\mathsf{MacCheck}$ depending on whether the underlying passively secure multiplication is Maurer, KRSW or DN based. We assume $\mathcal{F}_{\mathrm{PRSS}}$ is executed non-interactively in all cases, that any calls to $\mathcal{F}_{\mathrm{AgreeRandom}}$ are amortized across many calls to $\mathsf{MultCheck}_i$, and that no king-paradigm is used in order to keep the number of rounds to a minimum. As mentioned in the discussion on the multiplication checks, we always consider w/B to be negligibly small.

	General MSP			
Protocol	Rounds	Data	PRSS/PRZS	Triples
Share	1	$m - \lvert \mathbf{s}_i \rvert$	0	0
$\mathsf{OpenToOne}$	1	$m - \lvert \mathbf{s}_i \rvert$	0	0
$\mathsf{OpenToAll}$	1	$\star_1$	0	0
$\mathsf{BeaverMult}$	1	$2 \cdot \star_1$	0	1
$\mathsf{MaurerMult}$	1	$(n-1) \cdot m$	0	0
$\mathsf{KRSWMult}$	1	$\star_2$	$\star_3$	0
DNMult	1	$n \cdot (n-1)$	2	0
$\mathsf{MacCheck}^M$	4	$(n-1) \cdot m + 2 \cdot \star_1$	1	0
$\mathsf{MacCheck}^K$	4	$\star_2 + 2 \cdot \star_1$	$1 + \star_3$	0
$\mathsf{MacCheck}^D$	4	$n \cdot (n-1) + 2 \cdot \star_1$	3	0
$\mathsf{MultCheck}_1$	4	$(2 + w/B) \cdot \star_1$	0	0
$\mathsf{MultCheck}_2^M$	5	$(n-1) \cdot m + (1 + w/B) \cdot \star_1$	1	0
$\mathsf{MultCheck}_2^K$	5	$\star_2 + (1 + w/B) \cdot \star_1$	$1 + \star_3$	0
$\mathsf{MultCheck}_2^D$	5	$n \cdot (n-1) + (1 + w/B) \cdot \star_1$	3	0

Table 2. Costs of the Base Protocols for a General Access Structures

To provide more concrete values we also give, in the full version, the values for the three different instantiations of threshold sharings for $(n,t) \in \{(3,1),(5,2),(10,4)\}$. The three different sharings have been selected as replicated (for general p^k), standard Shamir (for the case of $p > n$) and Shamir obtained via Galois rings (for the important case of $p = 2$).

4 Offline Preprocessing Protocols

Given the previous components there are a large number of variations one can deploy to obtain an MPC protocol for a $\mathcal{Q}_2$ access structure which is actively secure with abort. In many cases, some form of preprocessing is used to generate multiplication triples. In this section, we aim to give an overview of different methods to generate passive and active multiplication triples, and evaluate the associated cost in terms of their round and data complexity. We give one passively secure offline protocol, and three actively secure variants. To generate actively secure multiplication triples, we generally first generate passively secure triples, and then we check for correctness (against potential additive attacks) in different ways.

Some of these offline protocols inherently require working (internally) with an extension of the modulus p^{k+v}, whilst all can produce triples modulo p^k or p^{k+v} depending on whether the output protocol requires triples modulo p^k or p^{k+v}. Whether the output is modulo p^k or p^{k+v} will depend into which main protocol we will embed the offline protocol. When we want to distinguish these various cases we will write $\mathsf{Offline}_X(p^{\mathsf{output}}, p^{\mathsf{internal}})$ for an offline protocol which outputs triples modulo p^{output}, whilst working internally modulo p^{internal}. Note, if $\mathsf{output} = k + v$ then we must have $\mathsf{internal} = k + v$ as well. In all cases we assume that all PRSS and PRZS operations are performed non-interactively, and all passive secure multiplications will be assumed to be performed using which ever is the best out of KRSW or DN for the specific parameter sets[6].

$\mathsf{Offline}_{\mathsf{Pass}}$: When generating N passively secure multiplication triples, we take the approach of first generating $2 \cdot N$ random sharings by performing $2 \cdot N$ calls to PRSS. Following that, we perform a passively secure multiplication protocol N times in parallel to compute the product over pairs of those shares. Since we can perform the N required multiplications in parallel, for the multiplication we only need a single round of communication, with a total data cost of $N \cdot \mathsf{PassMult}_{\mathsf{data}}$, and a corresponding cost of $\mathsf{PassMult}_{\mathsf{data}}$ per triple produced.

$\mathsf{Offline}_1$: The first actively secure protocol, $\mathsf{Offline}_1$, will follow the ideas presented in [11], in that to generate N actively secure multiplication triples it starts by executing $\mathsf{Offline}_{\mathsf{Pass}}$ to produce $2 \cdot N$ triples. Then half of the obtained triples are sacrificed, using $\mathsf{MultCheck}_1$, so as to check the remaining half for correctness. The cost of $\mathsf{Offline}_1(p^k, p^{k+v})$ and $\mathsf{Offline}_1(p^{k+v}, p^{k+v})$ are identical.

$\mathsf{Offline}_2$: For the second active offline protocol, $\mathsf{Offline}_2$, we follow [12]. First N passively secure triples are generated using $\mathsf{Offline}_{\mathsf{Pass}}$. Then these triples are checked to be resistant to additive attacks by running $\mathsf{MultCheck}_2$ on the vector of N triples. Again, the cost of $\mathsf{Offline}_2(p^k, p^{k+v})$ and $\mathsf{Offline}_2(p^{k+v}, p^{k+v})$ are identical.

$\mathsf{Offline}_3$: For our third variant of the Offline protocol, which we call $\mathsf{Offline}_3$, we use the cut-and-choose methodology of [3, Protocol 3.1]. This is parametrized by four integer parameters (Bk, C, X, L), and it generates $N = (X - C) \cdot L$ triples in each iteration, given input of $T = (N + C \cdot L) \cdot (\mathsf{Bk} - 1) + N$ passively secure triples. The value Bk represents a bucket size for the final checking procedure. The advantage of this version of the Offline protocol is that we achieve active security without needing to extend the ring, i.e. we can work modulo p^k and not work p^{k+v} if we require triples modulo p^k as output.

[6] These are both cheaper than Maurer in terms of data transfer, although they requires more PRSS and PRZS calls.

The statistical security offered by this approach is $1/N^{\mathsf{Bk}-1}$ when used as a standalone offline procedure, or $1/N^{\mathsf{Bk}}$ when used with a specific online procedure (see the third protocol of [3] for the details); note in the latter case one needs to select $C \geq 3$ and that this corresponds to our Protocol 4 below. In [3] the authors, for $p^k = 2$, target a statistical security level of $\kappa = 40$ bits. Thus, they can select $N = 2^{20}$, $\mathsf{Bk} = 2$, $L = 512$ and $C = 3$ to achieve an offline cost of 12 bits per triple when utilized in Protocol 4 below.

To provide a fair comparison between all protocols in this paper we target a statistical security level of $\kappa = 128$. Thus when using $\mathsf{Offline}_3$ in Protocol 1 below we use the parameters $(N, \mathsf{Bk}, L, C) = (2^{22}, 7, 512, 1)$ and when using $\mathsf{Offline}_3$ in Protocol 4 below we use the parameters $(N, \mathsf{Bk}, L, C) = (2^{22}, 6, 512, 3)$.

4.1 Comparing Actively Secure Offline Protocols

Having analysed the three actively secure offline protocols one could compare them theoretically, using the formulae. This is alas however not that illuminative, due to the complexity of the various parameters etc for $\mathsf{Offline}_3$. Comparing $\mathsf{Offline}_1$ vs $\mathsf{Offline}_2$, is simpler as $\mathsf{Offline}_1$ is better in terms of number of rounds of communication, whereas $\mathsf{Offline}_2$ is better in terms of the amount of data sent per multiplication.

5 Complete Protocols

We now examine the five (main) protocol variants we discussed in the introduction. For each of the following protocols, if an actively secure offline phase is required we can utilize the protocols $\mathsf{Offline}_x$, for x either 1, 2 or 3, given in Section 4. There are two basic metrics here that one could be interested in (assuming to a first order approximation we are processing arithmetic circuits over $\mathbb{Z}_{p^k}$), namely, the amount of data transferred per multiplication in the online phase only, or the amount of data transferred per multiplication in the combined online and offline phases. In all cases we assume we are processing an arithmetic circuit with N multiplication gates in a circuit of multiplicative depth d.

$\mathsf{Protocol}_1$: This protocol executes an actively secure offline phase to produce N triples in $\mathbb{Z}_{p^k}$, i.e. we execute $\mathsf{Offline}_x(p^k, p^\star)$ for $\star$ being either k or $k + v$, depending on the precise protocol choice x. Note, this means we have three choices for $\mathsf{Protocol}_1$ depending on which offline protocol the main protocol is combined with. The online phase is executed, using these triples, using $\mathsf{BeaverMult}$ as the multiplication procedure. Since the Beaver multiplication is instantiated with actively secure triples the output will also be actively secure, and no post-processing check is necessary. The online cost does not depend on the choice of offline phase. In Table 3 we refer to the three combined costs per multiplication as Total_x, depending on which Offline phase we are utilizing.

Protocol$_2$: In this protocol we optimistically use a passively secure online multiplication protocol PassMult to execute the online phase, and a passively secure Offline protocol to generate N passively secure multiplication triples, all over $\mathbb{Z}_{p^{k+v}}$. These are then checked using a post-processing methodology, based on MultCheck$_1$, to ensure active security. This approach of optimistic, passively secure online multiplication was first suggested in [12].

Protocol$_3$: This proceeds very much as Protocol$_2$ except instead of using an offline phase and the MultCheck$_1$ procedure, one uses the MultCheck$_2$ procedure. As there is no offline phase, online and post-processing costs are the total costs of the protocol. Again all operations needs to be performed over $\mathbb{Z}_{p^{k+v}}$.

Protocol$_4$: This protocol variant follows the pattern from [3] and thus is particularly suited to small values of p^k. It can be applied using any of the actively secure offline protocols, but is better suited (for small p^k) to be used with Offline$_3$.

In the offline phase we generate N actively secure multiplication triples in $\mathbb{Z}_{p^k}$. In the online phase a standard passively secure online phase is executed, using PassMult. Then in the post-processing the triples produced in the offline phase are checked against the 'triples' resulting from the passively secure multiplications, using MultCheck$'_1$. The entire procedure can be executed in $\mathbb{Z}_{p^k}$ without the need to extend to $\mathbb{Z}_{p^{k+v}}$. Again in Table 3 we will refer to the three different combined costs per multiplication as Total$_x$.

Protocol$_5$: Our final approach is based upon the technique in [5]. At the start of the protocol, in a (very short) offline phase a sharing for an unknown, secret random value $[\alpha]_{k+v}$ is generated. This value is used as an information theoretic MAC key, similar to the SPDZ approach.

In the online phase each wire value x is held as two shared values $\{[x]_{k+v}, [\alpha \cdot x]_{k+v}\}$. To multiply two values x and y we execute a passively secure multiplication twice, once with $[x]_{k+v}$ and $[y]_{k+v}$ to obtain $[x \cdot y]_{k+v}$, and one with $[x]_{k+v}$ and $[\alpha \cdot y]_{k+v}$ to obtain $[\alpha \cdot x \cdot y]_{k+v}$. In a short post-processing phase the MAC values on *all multiplication gates* and *all input and output wires* are checked using the MacCheck procedure. To ensure the security of the MacCheck procedure all computation need to be performed in $\mathbb{Z}_{p^{k+v}}$.

We can now present a summary (in Table 3) of all these options, by way of presenting their respective online and total communications costs (in number of bits communicated per multiplication), for a variety of different scenarios, access structures and base rings. In the table we mark in blue the online variant which is most efficient for a given access structure, ring, and ESP. This is almost always Protocol$_1$. We also mark in gray the most efficient protocol option when one is interested in the total cost. For small rings this is always Protocol$_4$ with Offline$_3$ chosen as the pre-processing, for the others it is Protocol$_5$.

Note, that in the case of Protocol$_4$ and Offline$_3$ the paper [3] obtains a total cost of 21 bits per multiplication operation. As explained earlier this is because

Access				$\mathsf{Protocol}_1$				$\mathsf{Protocol}_2$		$\mathsf{Protocol}_3$	
Structure	Ring	Scheme	Mult	Online	Total_1	Total_2	Total_3	Online	Total	Online	Total
(3, 1)	$\mathbb{F}_2$	Replicated	KRSW	6	1554	1167	63	1161	1548	1161	1161
(3, 1)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	6	2328	1941	84	1548	2322	1935	1935
(3, 1)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	768	3840	3072	8067	2304	3072	2304	2304
(3, 1)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	768	5376	4608	10756	3072	4608	3840	3840
(3, 1)	$\mathbb{F}_p$	Replicated	KRSW	768	3840	3072	8067	2304	3072	2304	2304
(3, 1)	$\mathbb{F}_p$	Shamir	KRSW	768	3840	3072	8067	2304	3072	2304	2304
(5, 2)	$\mathbb{F}_2$	Replicated	KRSW	40	7780	5200	350	6450	7740	5160	5160
(5, 2)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	40	10360	7780	420	7740	10320	7740	7740
(5, 2)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	5120	20480	15360	44820	12800	15360	10240	10240
(5, 2)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	5120	25600	20480	53782	15360	20480	15360	15360
(5, 2)	$\mathbb{F}_p$	Replicated	KRSW	5120	20480	15360	44820	12800	15360	10240	10240
(5, 2)	$\mathbb{F}_p$	Shamir	KRSW	2560	12800	10240	26891	7680	10240	7680	7680
(10, 4)	$\mathbb{F}_2$	Replicated	KRSW	1680	231300	122940	12116	223170	229620	121260	121260
(10, 4)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	260	57020	40250	2451	45150	56760	39990	39990
(10, 4)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	215040	670720	455680	1550803	442880	455680	240640	240640
(10, 4)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	33280	145920	112640	313735	89600	112640	79360	79360
(10, 4)	$\mathbb{F}_p$	Replicated	KRSW	215040	670720	455680	1550803	442880	455680	240640	240640
(10, 4)	$\mathbb{F}_p$	Shamir	KRSW	10240	56320	46080	116528	33280	46080	35840	35840

Access				$\mathsf{Protocol}_4$				$\mathsf{Protocol}_5$	
Structure	Ring	Scheme	Mult	Online	Total_1	Total_2	Total_3	Online	Total
(3, 1)	$\mathbb{F}_2$	Replicated	KRSW	9	1557	1170	57	774	774
(3, 1)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	12	2334	1947	78	1548	1548
(3, 1)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	1152	4224	3456	7299	1536	1536
(3, 1)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	1536	6144	5376	9988	3072	3072
(3, 1)	$\mathbb{F}_p$	Replicated	KRSW	1152	4224	3456	7299	1536	1536
(3, 1)	$\mathbb{F}_p$	Shamir	KRSW	1152	4224	3456	7299	1536	1536
(5, 2)	$\mathbb{F}_2$	Replicated	KRSW	50	7790	5210	310	2580	2580
(5, 2)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	60	10380	7800	380	5160	5160
(5, 2)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	6400	21760	16640	39696	5120	5120
(5, 2)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	7680	28160	23040	48659	10240	10240
(5, 2)	$\mathbb{F}_p$	Replicated	KRSW	6400	21760	16640	39696	5120	5120
(5, 2)	$\mathbb{F}_p$	Shamir	KRSW	3840	14080	11520	24329	5120	5120
(10, 4)	$\mathbb{F}_2$	Replicated	KRSW	1730	231350	122990	10435	12900	12900
(10, 4)	$\mathbb{F}_2$	Shamir $\mathbb{Z}_{2^k}$	DN	350	57110	40340	2191	23220	23220
(10, 4)	$\mathbb{Z}_{2^{128}}$	Replicated	KRSW	221440	677120	462080	1335642	25600	25600
(10, 4)	$\mathbb{Z}_{2^{128}}$	Shamir $\mathbb{Z}_{2^k}$	DN	44800	157440	124160	280432	46080	46080
(10, 4)	$\mathbb{F}_p$	Replicated	KRSW	221440	677120	462080	1335642	25600	25600
(10, 4)	$\mathbb{F}_p$	Shamir	KRSW	16640	62720	52480	106280	25600	25600

Table 3. Costs of the Full Protocols in number of bits per multiplication, for various access structures; $\kappa = 128$, $p \approx 2^{128}$

they target a statistical security level of $\kappa = 40$, instead of our security level of $\kappa = 128$.

Note that even when $\mathsf{Protocol}_1$ is not the most efficient choice, in practice one might still prefer using this protocol as our analysis assumes the only interaction occurs for multiplication. Most MPC protocols make use of $\mathsf{OpenToAll}$ executions to open masked data for use in various function specific optimizations. Using $\mathsf{Protocol}_1$ enables these protocol specific $\mathsf{OpenToAll}$ executions to be merged easily with the $\mathsf{OpenToAll}$ executions used in multiplication; thus reducing the total round count. For other online protocols this merging can be more complex.

Acknowledgements

We would like to thank Daniel Escudero and Tim Wood for conversions on aspects of this work whilst it was carried out.

This work has been supported in part by ERC Advanced Grant ERC-2015-AdG-IMPaCT, by the Defense Advanced Research Projects Agency (DARPA) and Space and Naval Warfare Systems Center, Pacific (SSC Pacific) under contract FA8750-19-C-0502, by the FWO under an Odysseus project GOH9718N, and by CyberSecurity Research Flanders with reference number VR20192203.

Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of any of the funders. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

1. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Rambaud, M., Xing, C., Yuan, C.: Asymptotically good multiplicative LSSS over Galois rings and applications to MPC over $\mathbb{Z}/p^k\mathbb{Z}$. In: Moriai, S., Wang, H. (eds.) ASIACRYPT 2020, Part III. LNCS, vol. 12493, pp. 151–180. Springer, Heidelberg (Dec 2020)
2. Abspoel, M., Dalskov, A., Escudero, D., Nof, A.: An efficient passive-to-active compiler for honest-majority MPC over rings. Cryptology ePrint Archive, Report 2019/1298 (2019), https://eprint.iacr.org/2019/1298
3. Araki, T., Barak, A., Furukawa, J., Lichter, T., Lindell, Y., Nof, A., Ohara, K., Watzman, A., Weinstein, O.: Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In: 2017 IEEE Symposium on Security and Privacy. pp. 843–862. IEEE Computer Society Press (May 2017)
4. Bogdanov, D., Laur, S., Willemson, J.: Sharemind: A framework for fast privacy-preserving computations. In: Jajodia, S., López, J. (eds.) ESORICS 2008. LNCS, vol. 5283, pp. 192–206. Springer, Heidelberg (Oct 2008)
5. Chida, K., Genkin, D., Hamada, K., Ikarashi, D., Kikuchi, R., Lindell, Y., Nof, A.: Fast large-scale honest-majority MPC for malicious adversaries. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 34–64. Springer, Heidelberg (Aug 2018)
6. Cramer, R., Damgård, I., Escudero, D., Scholl, P., Xing, C.: SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part II. LNCS, vol. 10992, pp. 769–798. Springer, Heidelberg (Aug 2018)
7. Cramer, R., Damgård, I., Ishai, Y.: Share conversion, pseudorandom secret-sharing and applications to secure computation. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 342–362. Springer, Heidelberg (Feb 2005)
8. Cramer, R., Damgård, I., Maurer, U.M.: General secure multi-party computation from any linear secret-sharing scheme. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 316–334. Springer, Heidelberg (May 2000)
9. Cramer, R., Rambaud, M., Xing, C.: Asymptotically-good arithmetic secret sharing over $Z/(p^\ell Z)$ with strong multiplication and its applications to efficient MPC. Cryptology ePrint Archive, Report 2019/832 (2019), https://eprint.iacr.org/2019/832
10. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: Menezes, A. (ed.) CRYPTO 2007. LNCS, vol. 4622, pp. 572–590. Springer, Heidelberg (Aug 2007)

11. Damgård, I., Pastro, V., Smart, N.P., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (Aug 2012)
12. Eerikson, H., Keller, M., Orlandi, C., Pullonen, P., Puura, J., Simkin, M.: Use your brain! Arithmetic 3PC for any modulus with active security. In: Kalai, Y.T., Smith, A.D., Wichs, D. (eds.) ITC 2020. pp. 5:1–5:24. Schloss Dagstuhl (Jun 2020)
13. Fehr, S.: Span programs over rings and how to share a secret from a module (1998), MSc Thesis, ETH Zurich
14. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 20. pp. 1575–1590. ACM Press (Nov 2020)
15. Keller, M., Orsini, E., Scholl, P.: MASCOT: Faster malicious arithmetic secure computation with oblivious transfer. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 830–842. ACM Press (Oct 2016)
16. Keller, M., Rotaru, D., Smart, N.P., Wood, T.: Reducing communication channels in MPC. In: Catalano, D., De Prisco, R. (eds.) SCN 18. LNCS, vol. 11035, pp. 181–199. Springer, Heidelberg (Sep 2018)
17. Maurer, U.M.: Secure multi-party computation made simple. Discrete Applied Mathematics 154(2), 370–381 (2006)
18. Smart, N.P., Wood, T.: Error detection in monotone span programs with application to communication-efficient multi-party computation. In: Matsui, M. (ed.) CT-RSA 2019. LNCS, vol. 11405, pp. 210–229. Springer, Heidelberg (Mar 2019)