More efficient comparison protocols for MPC

...

Nillion

Abstract. In 1982, Yao introduced the problem of comparing two private values, thereby launching the study of protocols for secure multi-party computation (MPC). Since then, comparison protocols have undergone extensive study and found widespread applications.
We survey state-of-the-art comparison protocols for an arbitrary number of parties, decompose them into smaller primitives and analyse their communication complexity under the usual assumption that the underlying MPC protocol does preprocessing and computes linear operations without communication. We then develop two new comparison protocols and explain why they are faster than similar protocols, including those that are commonly used in practice: they reduce the number of online multiplications, without increasing preprocessing or round complexity. More concretely, online bandwidth is reduced by more than half for the standard comparison protocols whose round complexity is logarithmic in the bit-length, whereas for constant round comparison protocols the reduction is two-thirds.

1 Introduction

In 1982, Yao introduced the problem of comparing two private values in a privacy-preserving manner, now known as "Yao's Millionaires' problem" [48]. Over time this led to the development of many secure multi-party computation (MPC) protocols, which enable a group to jointly perform a computation without disclosing any private inputs [31]. Originally most MPC protocols would carry out a generic computation by rewriting it as a Boolean circuit or an arithmetic circuit, and then execute each addition (or XOR) and multiplication gate in this circuit using certain subprotocols. As many practical computations involve secure comparisons (which securely determine whether one secret integer is larger than another one) and secure fixed-point or floating-point arithmetic, many MPC protocols subsequently developed specialised subprotocols to handle those subcomputations as efficiently as possible.

Moreover, almost all practical MPC protocols nowadays tend to compute addition gates without requiring any communication; this includes those protocols that use linear secret sharing schemes (LSSS) such as Shamir's secret sharing [47,11,30,25,32] and additive secret sharing like SPDZ [26,10,37]. Real-world applications of LSSS-based MPC protocols involving secure comparisons and secure fixed-point arithmetic include:

- *Auctions.* The first ever large-scale MPC application happened in 2008, when the bids of 1229 farmers in the Danish beet market were encrypted in order to protect their (financial) privacy [15], and the market clearing price was determined by comparing these secret bids. A more advanced version of the same platform was later used by the Norwegian 1800 MHz Spectrum Auction to trade approximately US$ 100 million in spectrum rights [3].
- *Descriptive statistics.* The first wide area network deployment of MPC came in 2010, when the Estonian IT industry consortium wanted to gather sensitive data for a market survey; the published analysis required sorting of private data and division [14]. Later in Denmark, banks computed credit score analyses of farmers using confidential accounting data [21]. Another example is the Boston wage gap MPC computation by the Boston Women's Workforce Council, demonstrating that the wage gap is larger than previously estimated by the U.S. Bureau of Labor Statistics [39].[1]
- *Evidence-based policy making.* In 2015, Estonian statisticians looked for correlations between working during university studies and failing to graduate in time. This required combining private tax and education data [13].

[1] Here the contents of certain cells were only revealed if a certain threshold was met.

– *Fraud detection.* In 2015, the Estonian Tax and Customs Board estimated VAT tax fraud by comparing detailed but confidential purchase and sales data between companies [12].
– *Biometric authentication.* Unlike ordinary passwords, biometric identifiers tend to be hard to change once compromised. Consequently, biometric authentication services are sometimes deployed with MPC; typically the underlying algorithm determines whether the distance between two feature vectors is below some threshold (e.g. [8, §6.2.3] or [1]).

Nevertheless, the deployment of MPC protocols to perform such computations still leads to serious efficiency penalties, and research efforts have focused on improving the performance of the underlying primitives. In contemporary MPC literature on secure comparison [45,18,40,22,41,27,7] and fixed- [19,18,27] and floating-point arithmetic [6,35,17] protocols for groups of arbitrary size $n \geq 3$, they all largely reduce to subprotocols which compare the bits of a secret integer with the bits of a nonsecret integer. Sometimes these subprotocols already vary in their outward properties (e.g. offering statistical rather than perfect privacy), and sometimes the differences are all hidden "under the hood". Even in the latter case, a careful deconstruction can already lead to performance improvements.

For example, while most LSSS-based MPC protocols convert generic functions to (generalisations of) arithmetic circuits in some domain $\mathbb{Z}/m\mathbb{Z}$ with $m > 2$, comparing two integers usually comes down to comparing bits which is more naturally a binary operation. This leads to two different approaches: perform this bitwise comparison computation entirely inside of the original arithmetic domain $\mathbb{Z}/m\mathbb{Z}$, or employ random values which are secret shared in both the original domain and in (an extension of) the binary domain $\mathbb{Z}/2\mathbb{Z}$ (e.g. [28]) to do the computation inside the latter domain, before transforming it back to the arithmetic domain again. The former approach is taken by [45,40,27], and the latter by [22,41,7]. Nevertheless, the two binary bit comparison protocols of [22,41] differ ([7] reuses the latter); while both protocols were selected from [18] (which has protocols for either domain) and are functionally interchangeable, one will outperform the other one due to having significantly lower communication complexity, whilst [7] should use neither.

Similarly, [45] notes that his comparison protocol can be improved in communication complexity by making the underlying bit comparison protocol (which itself cannot be deployed over $\mathbb{Z}/2\mathbb{Z}$) compare "two bits at a time". It appears that this idea was forgotten by the literature since then; we explain that it also speeds up the state-of-the-art bit comparison circuit of [29], regardless of domain, and provide an in-depth analysis of the consequences for communication complexity when comparing $\nu \geq 1$ bits "simultaneously". Setting $\nu = 2$ indeed improves both online and total bandwidth, but for most situations $\nu = 3$ or $\nu = 4$ is even better for the bit comparison protocol of [45].

In this paper, we present two new (bit)comparison protocols, both of which can be finetuned by a parameter $\nu \geq 1$. In their respective sections, we explain why they are faster and more efficient than similar protocols in the MPC literature. In the appendices we also recall and make slight tweaks to other families of comparison protocols. A rough guide to choosing between secure comparison protocols for any LSSS-based MPC protocol with preprocessing could be as follows:

– If the round complexity is the primary bottleneck, the fastest protocol might be $\mathcal{P}^{\mathtt{3R},\nu}_{\mathtt{LessThan}}$; in the setting of honest majority MPC it should be executable within 3 rounds, and within 4 rounds otherwise. A default setting here might be $\nu = 3$, possibly increasing it to $\nu = 4$ for lower online bandwidth or lowering it to $\nu = 2$ for slightly lower total bandwidth (see Table 3).
– If a round complexity of roughly $\lceil \log_2 \ell \rceil$ is acceptable when comparing two values of bit-length ℓ, then the protocol of $\mathcal{P}^{\mathtt{logR},\nu}_{\mathtt{LessThan}}$ should yield the lowest total bandwidth of all protocols considered when $\nu = 2$, see Table 1. Its online bitwise comparison component and some of its preprocessing can be executed over a small field (employing say [28]) for even lower offline and much lower online bandwidth (assuming n is not too large).

Our contributions

(i) We carefully deconstruct the comparison protocols of [45,18,40,22,41,27,7], explaining their commonalities and differences.

(ii) We present a constant-round secure comparison protocol $\mathcal{P}^{\texttt{3R},\nu}_{\texttt{LessThan}}$ with communication complexity linear in the number of bits, based on the ideas of [45] and the 3-round protocol [18, LTZC1]. Compared to the latter, the offline bandwidth is typically lower and it only uses a third of the online bandwidth when $\nu = 3$ (see Table 3).

(iii) We present a secure comparison protocol $\mathcal{P}^{\texttt{logR},\nu}_{\texttt{LessThan}}$ whose round complexity is logarithmic and bandwidth complexity is linear in the number of bits, based on [29]. It costs one round fewer than similar protocols [18, LTZL] and [29,22,41,27] (which are the comparison protocols used most often in practice for LSSS-based MPC) and has less than half the online banwidth, whilst typically also slightly lowering total bandwidth (see Table 4).

If one were to model the underlying MPC protocol with its access structure as an arithmetic black-box $\mathcal{F}_{\texttt{ABB}}$ [24] providing security (possibly with abort) against an active or passive adversary, and also describe the secure comparison operation as an ideal functionality $\mathcal{F}_{\texttt{Comparison}}$ in the UC framework [16], a more formal security statement is as follows:

Theorem 1. *The protocols $\mathcal{P}^{\texttt{3R},\nu}_{\texttt{LessThan}}$ and $\mathcal{P}^{\texttt{logR},\nu}_{\texttt{LessThan}}$ information-theoretically UC-realise $\mathcal{F}_{\texttt{Comparison}}$ in the $\mathcal{F}_{\texttt{ABB}}$-hybrid model against an adversary corrupting an authorised subset of parties.*

Acknowledgements ...

2 Notation

- All logarithms are base 2, unless specified otherwise. The set of natural numbers without zero is denoted $\mathbb{N}_1$, the set of integers by $\mathbb{Z}$.
- The arithmetic computation domain is denoted $\mathbb{Z}/m\mathbb{Z}$, for some integer m. Traditionally m was assumed to be prime [47], but Shamir's secret sharing scheme is increasingly deployed for m nonprime (e.g. [5,4]).
- In addition to the usual linear operations, multiplication $\mathcal{P}_{\texttt{Mult}}$ and reveal $\mathcal{P}_{\texttt{Reveal}}$ subprotocols, the underlying MPC protocol may have a special "public multiplication" subprotocol $\mathcal{P}_{\texttt{PubMult}}$ (which given secret sharings $[x]$ and $[y]$ publicly outputs the value xy), rather than naively composing $\mathcal{P}_{\texttt{Mult}}$ with $\mathcal{P}_{\texttt{Reveal}}$; to the best of our knowledge this notion first appeared in [18]. If the MPC protocol is say UC-secure [16], then so are the protocols in this paper since they only reveal values that have been shifted by random numbers.
- $[\cdot]$ denotes a secret shared value the underlying MPC protocol can securely perform all of its usual operations on, e.g. in the context of threshold secret sharing it is a sharing of degree t, where t is the maximum number of adversaries. $[\![\cdot]\!]$ denotes a sharing on which linear operations and reveals are possible but not necessarily multiplications; in the same context it could denote a sharing of any degree less than the number of parties n. We set $N := \lceil \log n \rceil$.
- We let $\ell := \lceil \log(m-1) \rceil$ denote the bit-length of $m-1$, so that $2^{\ell-1} < m \le 2^{\ell}$. Unless statistical slack is used (e.g. [18]) this value will be only slightly larger than the bit-length of the inputs x, x' for the comparison subprotocols. For bandwidth computations we implicitly assume that m is close to 2^{ℓ}, so that the cost of constructing a bitwise shared random number is roughly the same as the cost of constructing ℓ bits when using e.g. [46]. We write $L := \lceil \log \ell \rceil$.
- The arity of certain bit operations is denoted by $\nu \in \mathbb{N}_1$, and we write $\ell_\nu := \lceil \ell/\nu \rceil \approx \ell/\nu$. We use a superscript $\bullet$ in the notation of protocols to denote two parameters: their round complexity and this arity ν.
- The least significant bit of an element $z \in \mathbb{Z}/m\mathbb{Z}$ is denoted z_1, the next one by z_2, etc., so that $z = \sum_{i=1}^{\ell} 2^{i-1} z_i$. We let $[z]_{\text{bit}}$ denote a bitwise sharing, i.e. a sharing of all of the bits of z; implicitly we may assume that a sharing of z is included as well. This is generalised by $[z]_{\nu\text{-bit}}$, which reduces to $[z]_{\text{bit}}$ in the case $\nu = 1$; see Notation 43.

3 Survey of comparison protocols

In this section we survey state-of-the-art secure comparison protocols for an arbitrary number of parties; recall that the problem is to compute a sharing $[x < x']$ of the Boolean $x < x'$ from a pair of secret sharings $[\![x]\!], [\![x']\!]$ of integers $x, x' \in \mathbb{Z}$. In the setting of honest majority threshold LSSS, allowing the degree of these sharings to be larger than $t+1$ here can reduce a round of communication when one (or both) of the inputs are obtained as for example the output of a multiplication or inner product gate.

The comparison problem is usually reduced to comparing a public value c with a secret random value whose bits are secret-shared in some domain, typically either the original domain or (some extension of) $\mathbb{Z}/2\mathbb{Z}$. For simplicity we will refer to such a secret as a *bitwise-shared*, and to such a comparison as a *bitwise comparison*. The only exception to this approach that we're aware of is [38, §II.B], but that protocol has some issues; see Appendix B.

Originally, comparison protocols used an expensive online bit decomposition protocol [23]; shortly afterwards, it was noticed that this can often be replaced with a "shifted bit decomposition" [42], as follows:

Protocol `ShiftedBitDecomp`

$$(z + r, [r]_{\nu\text{-bit}}) \longleftarrow \mathcal{P}_{\texttt{ShiftedBitDecomp}}([\![z]\!], \nu)$$

1. The nodes retrieve from preprocessing a bitwise sharing $[r]_{\nu\text{-bit}}$, and locally compute $[\![z + r]\!] := [\![z]\!] + [r]$.
2. The nodes then reveal this value: $z + r \leftarrow \mathcal{P}_{\texttt{Reveal}}([\![z + r]\!])$.

Fig. 1: Protocol to compute a shifted bit decomposition of a shared value

3.1 Range and the arithmetic domain

In practice the integers x and x' are usually restricted to lie inside some range of integers $[x_{\min}, x_{\max}) \subset \mathbb{Z}$ of size $s := x_{\max} - x_{\min}$; typical choices are the unsigned integers in the range $[0, 2^k)$ and signed integers in the range $[-2^{k-1}, 2^{k-1})$, for some natural number $k \in \mathbb{N}_1$. This range is subsequently embedded by the underlying arithmetic MPC protocol into the ring $\mathbb{Z}/m\mathbb{Z}$ through the natural projection map

$$[x_{\min}, x_{\max}) \hookrightarrow \mathbb{Z} \twoheadrightarrow \mathbb{Z}/m\mathbb{Z} \cong [0, \ldots, m). \tag{1}$$

The comparison subprotocol then usually has one of the following two properties:

(i) It is capable of computing $[z < z']$ for all elements z, z' in $[0, \ldots, m)$; by shifting the range of $[x_{\min}, x_{\max})$ with $x_{\min}$ just before the start of this subprotocol, any $[x < x']$ can then be computed as long as $s \leq m$.
   - [42, §6.3] does this at the cost of three bitwise shared secret random values, three parallel bitwise comparisons followed by two consecutive multiplications; they require that m is prime but this condition is not necessary.
   - [41, Π_{LTS}] costs two bitwise shared secret random values, three parallel bitwise comparisons and a binary adder. If their binary adder is moved to preprocessing their preprocessing cost will be slightly higher than [42, §6.3], but require a few less online rounds. Their correctness is probabilistic if m is not a power of 2.

(ii) It is capable of computing $[z < z']$ for all elements z, z' within a range of size $m - 2^{\ell-1}$ or $m/2$. Depending on m and privacy requirements, these protocols only cost 1 or 2 two bitwise shared secret random values plus bitwise comparisons, rather than 3.

The latter approach thus requires the underlying MPC system to increase the range of $\mathbb{Z}/m\mathbb{Z}$ (adding one or two bits), but is much more efficient in communication complexity otherwise. Mathematically, this approach is done by:

3.2 Comparison with zero

Note that the difference $x - x'$ lies in the range $(-s, s)$. Thus if $\mathbb{Z}/m\mathbb{Z}$ can accommodate this range, we obtain

Protocol LessThan

$$[x < x'] \longleftarrow \mathcal{P}^{\bullet}_{\texttt{LessThan}}(\llbracket x \rrbracket, \llbracket x' \rrbracket)$$

1. The nodes locally compute $\llbracket y \rrbracket := \llbracket x - x' \rrbracket$ and then execute

$$[x < x'] \longleftarrow \mathcal{P}^{\bullet}_{\texttt{LessThanZero}}(\llbracket y \rrbracket).$$

Fig. 2: Protocol to compute the sharing of the inequality of two shared values

We can distinguish two modern approaches to determining the boolean $y < 0$:

(i) Suppose that $s \leq m - 2^{\ell-1} \leq m/2$. The negative range $(-s, 0)$ then gets mapped under the projection map (1) to $(m-s, m) \subseteq (2^{\ell-1}, m)$ and the positive range to $[0, s) \subseteq [0, 2^{\ell-1})$. Thus it suffices to compute the most significant bit of y. For this, a sharing of $y \mod 2^{\ell-1}$ is computed (and then subtracted from y). This computation costs a bitwise shared secret random number plus reveal, followed by a bitwise comparison [18] and a second bitwise comparison if no statistical slack is used. Then the factor $2^{\ell-1}$ is removed for free if m is a prime [18], otherwise with a random bit plus reveal [22].

(ii) Suppose that $s - 1 < m/2$ and that m is odd, then it suffices to compute the least significant bit of $2y$. This idea is due to [42]; in [27] it is implemented with a bitwise shared secret random value plus reveal, a bitwise comparison and finally a multiplication.[2] It is also used in [45], where the output of the bitwise comparison is a bit different (see the next subsection) and consequently the last step requires another second bitwise shared secret random number plus reveal instead.

Protocol LessThanZero

$$[y < 0] \longleftarrow \mathcal{P}^{\bullet}_{\texttt{LessThanZero}}(\llbracket y \rrbracket)$$

1. The nodes locally compute $\llbracket z \rrbracket = 2\llbracket y \rrbracket$, then they compute

$$[y < 0] \longleftarrow \mathcal{P}^{\bullet}_{\texttt{LSB}}(\llbracket z \rrbracket).$$

Fig. 3: Protocol to compute a sharing of the sign of a secret number

3.3 Protocols for bitwise comparison

There are several protocols to compute a bitwise comparison; let us briefly review them. We assume that one input z is bitwise shared (so its bits z_i are secret-shared) whilst the other input c is public, and for simplicity we assume that the bit-length ℓ is an even power of 2. The complexity of the protocols considered here (in terms of the number of multiplication gates) is linear in the number of bits; for a protocol whose complexity is logarithmic in the number of bits, see Appendix C. With the exception of the last approach, each of the following works over any ring $\mathbb{Z}/m\mathbb{Z}$, including the binary domain $\mathbb{Z}/2\mathbb{Z}$.

Notation 31. We write $[c \overset{j}{\neq} z] := (c_j \neq [z_j]) := c_j \oplus [z_j] := c_j + [z_j] - 2c_j[z_j]$ for a sharing of the XOR of the j-th bits of c and z, and similarly for other bit operations.

[2] This final multiplication can usually be integrated into the bitwise comparison protocol; if ℓ is not a power of two (which happens when statistical slack is used, as in e.g. [18]), this typically reduces the number of rounds by one.

(i) The original protocol of [23] uses the identity (which is also used by [42,41,7])

$$[c < z] = \sum_{i=1}^{\ell}(1 - c_i)\Big(\bigvee_{j=i}^{\ell}[c \overset{j}{\neq} z] - \bigvee_{j=i+1}^{\ell}[c \overset{j}{\neq} z]\Big).$$

The natural approach to compute the right-hand-side is then to deduce it from a "prefix OR"; originally this was done in

- 5 rounds at the cost of 3 prefix multiplications over ℓ terms[3] and 2ℓ multiplications [42],[4] but is nowadays done in
- $\log(\ell)$ rounds with $\ell\log(\ell)/2$ multiplications [18] (using the scan of [33]), and can alternatively be done in
- in 2 rounds at the cost of 1 prefix multiplication and 1 bitwise shared integer [18, `PreOrC`], when only using statistical security,
- and is also done in 1 round after switching to a garbling protocol for $\ell\log(\ell)/2$ multiplications [7].[5]

(ii) The protocol `BitLTL` in [18] (which is also used by [22]) computes the right-hand-side of

$$[c < z] = [c + (2^k - z) < 2^k]$$

through a carry bit algorithm, in $\log(\ell)$ rounds with $2\ell - \log(\ell) - 2$ multiplications.[6]

(iii) [29] (which is also the circuit employed by [27]) computes the right-hand-side of

$$[c < z] = \sum_{i=1}^{\ell}[c \overset{i}{<} z] \prod_{j=i+1}^{\ell} [c \overset{j}{=} z] = \sum_{i=1}^{\ell}(1 - c_i)[z_i] \prod_{j=i+1}^{\ell} \big(1 - c_j + (2c_j - 1)[z_j]\big)$$

in $\log(\ell)$ rounds through $2\ell - \log(\ell) - 2$ multiplications.

(iv) Finally, the 3-round `LTZC1` and 5-round `LTZC2` protocols of [18] and the constant-round protocols of [45] compute $[c < z]$ by first computing

$$\sum_{i=1}^{\ell}[c \overset{i}{<} z]2^{\sum_{j=i+1}^{\ell}[c \overset{j}{\neq} z]} = \sum_{i=1}^{\ell}(1 - c_i)[z_i] \prod_{j=i+1}^{\ell} \big(1 + c_j + (1 - 2c_j)[z_j]\big), \tag{2}$$

and then extracting its least significant bit (by two slightly different methods, due to the difference in privacy).

At first glance, the final approach seems less efficient than the others:

- Extracting the least significant bit requires an extra bitwise shared secret random number [45] or a reduction from information-theoretic security and perfect privacy to computational security and statistical privacy [18].
- If $m \neq 2^\ell$ there is a chance of overflow, yielding only probabilistic correctness (but this is fixed with $\nu > 1$, see Remark 3(i)).
- Unlike the other approaches, it does not correspond to a binary computation which can be performed over $\mathbb{Z}/2\mathbb{Z}$.

However, it comes with two advantages:

[3] The actual algorithm performs $3\sqrt{\ell}$ prefix multiplications of $\sqrt{\ell}$ terms, which has the same cost.
[4] This method of requires a ring containing $[0, \ldots, \ell + 1]$.
[5] They state they use the circuit of [41], which seems to use the circuit of [18], but one might as well use a circuit with ℓ multiplications.
[6] In [18] it uses $2\ell - 2$ multiplications, but one can be skipped in each round [36].

– The terms in the prefix product are always nonzero, which allows for the prefix multiplication to be done in one online round without extra tricks [9,42,18], leading to a protocol that is significantly faster than the constant-round protocols of approach (i). Moreover, invertibility allows for fast bit-decompositions [44] (in the setting of computational security).
– The same formula works when comparing not one but multiple bits simultaneously, and this can be much faster. In the next section we give a careful analysis explaining why this is indeed optimal for reducing total bandwidth, and use the same trick in a different protocol.

4 Comparing multiple bits simultaneously

In the paper [45], several methods are described to construct a constand-round comparison protocol. Of particular note is the the idea of [45] to compare 2 bits simultaneously, and apply this speed up the online computation of equation (2). More generally, one could consider comparing ν bits simultaneously; we will use this in our 2-round bitwise comparison protocol of subsection §5.1, which largely follows [45], but also in our logarithmic-round protocol of subsection §5.2 which is inspired by [29]. For the latter protocol, we have added another improvement which is described in subsection §5.2.2. This section is dedicated to a careful analysis of the implications on communication complexity of varying over the natural number ν.

4.1 Möbius inversion

Recall that a Boolean function f with arity k can be extended to a multilinear polynomial p_f in the polynomial ring $\mathbb{Z}[X_1, \ldots, X_k]$ through the formula

$$p_f(X_1, \ldots, X_k) = \sum_{a \in \{0,1\}^k} f(a) \prod_{i: a_i = 0} (1 - X_i) \prod_{i: a_i = 1} X_i. \tag{3}$$

This section focuses on Boolean functions f_ν on two inputs of size $\nu \leq \ell$ (so these functions have arity 2ν):

Notation 41. For $s \in \{0,1\}^\nu$ let s_i denote the i-th bit of s and consider the multilinear monomial $x^s := x_1^{s_1} \cdots x_\nu^{s_\nu}$ (and similarly for c). Now given a Boolean function $f_\nu : \{0,1\}^\nu \times \{0,1\}^\nu \to \{0,1\}$, let us expand and then rewrite the polynomial (3) as

$$p_{f_\nu}(c, x) := p_{f_\nu}(c_1, \ldots, c_\nu, x_1, \ldots, x_\nu) = \sum_{\tilde{s}, s \in \{0,1\}^\nu} p_{f_\nu}^{\tilde{s},s} c^{\tilde{s}} x^s = \sum_{s \in \{0,1\}^\nu} p_{f_\nu}^s(c) x^s,$$

where the coefficients $p_{f_\nu}^{\tilde{s},s}$ lie in $\mathbb{Z}$ and the polynomials $p_{f_\nu}^s(\cdot)$ are thus defined as $p_{f_\nu}^s(c) := \sum_{\tilde{s} \in \{0,1\}^\nu} p_{f_\nu}^{\tilde{s},s} c^{\tilde{s}}$. See also Examples 1 and 2.

Furthermore, recall the Hamming weight $|s| := \sum_{i=1}^\nu s_i$ of a bit vector, and recall the partial ordering on bit vectors which is given by

$$s' \leq s \qquad \text{if and only if} \qquad s_i = 1 \text{ when } s'_i = 1.$$

Given a Boolean function f_ν we can precompute (and store as an associative array) the coefficients $p_{f_\nu}^{\tilde{s},s}$ using Möbius inversion, which we will interpret as the data for the function p_{f_ν}:

Protocol `ArithFunc`

$$p_{f_\nu} \longleftarrow \mathcal{P}_{\texttt{ArithFunc}}(f_\nu)$$

1. For all pairs $\tilde{s}, s$ in $\{0,1\}^\nu$, the nodes locally compute the coefficients

$$p_{f_\nu}^{\tilde{s},s} = (-1)^{|\tilde{s}|+|s|} \sum_{\tilde{s}' \leq \tilde{s}, s' \leq s} (-1)^{|\tilde{s}'|+|s'|} f_\nu(\tilde{s}', s')$$

Fig. 4: Protocol to compute the coefficients of a Boolean function in two variables

Given the value c, the coefficients $p^s_{f_\nu}(c)$ of the partial functions $p_{f_\nu}(c,\cdot)$ can be computed straightforwardly:

Protocol `PartArithFunc`

$$p_{f_\nu}(c,\cdot) \longleftarrow \mathcal{P}_{\texttt{PartArithFunc}}\big(f_\nu, c\big)$$

1. The nodes retrieve from preprocessing a list of integers: $p_{f_\nu} \leftarrow \mathcal{P}_{\texttt{ArithFunc}}(f_\nu)$.
2. The nodes locally compute for each s the integers

$$p^s_f(c) = \sum_{\tilde{s}\in\{0,1\}^\nu} p^{\tilde{s},s}_{f_\nu} c^{\tilde{s}}.$$

Fig. 5: Protocol to compute the coefficients of the partial application of a Boolean function in two variables

Proposition 1. *Consider the Boolean functions $c < x$ and $c = x$, acting on a pair of positive integers c, x of bit-length ν in the usual way, as functions on their bits $c_1, \dots, c_\nu, x_1, \dots, x_\nu$. First extend both of these functions to polynomials $p_{c<x}$ and $p_{c=x}$ in $\mathbb{Z}[c_1, \dots, c_\nu, x_1, \dots, x_\nu]$, and then expand these polynomials into monomials for the second set of variables $x_1, \dots, x_\nu$, so*

$$p_{c<x} = \sum_{s\in\{0,1\}^\nu} p^s_{c<x}(c_1,\dots,c_\nu)x^s, \qquad \textit{and} \qquad p_{c=x} = \sum_{s\in\{0,1\}^\nu} p^s_{c=x}(c_1,\dots,c_\nu)x^s.$$

Then:

(i) *For all $s \neq (0,\dots,0)$, the constant term of $p^s_{c<x}(c_1,\dots,c_\nu)$ is given by $(-1)^{|s|+1}$.*
(ii) *For all s, the constant term of $p^s_{c=x}(c_1,\dots,c_\nu)$ is given by $(-1)^{|s|}$.*

In particular, for those elements s the monomials $p^s_{c<x}$ and $p^s_{c=x}$ are never zero.

Proof. By a form of Möbius inversion applied to the function $f(c,x)$ (see Figure 4 and 5), it follows that

$$p^s_f(0) = (-1)^{|s|} \sum_{s'\le s} (-1)^{|s'|} f(0,s').$$

(i): For each $s' \le s$ not equal to the zero vector, we have $f(0,s') = 1$, so by the binomial theorem

$$p^s_f(0) = (-1)^{|s|} \sum_{1\le i\le |s|} \binom{|s|}{i} (-1)^i = (-1)^{|s|}\big((1-1)^n - \binom{|s|}{0}(-1)^0\big) = (-1)^{|s|+1}.$$

(ii): If $s' \le s$ is equal to the zero vector then $f(0,s') = 1$, and otherwise $f(0,s') = 0$. □

Example 1. Suppose the bit-length ν is 1. Then

$$p_{c<x} = (1-c_1)x_1 \qquad \text{and} \qquad p_{c=x} = 1 - c_1 + (2c_1 - 1)x_1.$$

Example 2. Suppose the bit-length ν is 2. Then

$$\begin{aligned} p_{c<x} &= (1-c_1)(1-c_2)x_1 + (1-c_2)x_2 + (1-c_1)(2c_2-1)x_1x_2, \\ p_{c=x} &= (1-c_1)(1-c_2) + (2c_1-1)(1-c_2)x_1 + (1-c_1)(2c_2-1)x_2 \\ &\quad + (2c_1-1)(2c_2-1)x_1x_2. \end{aligned}$$

Proposition 2. *In order to construct the arithmetic analog (3) of the Boolean function $c < x$ or $c = x$ from variables $x_1, \ldots, x_\nu$, we need to do $2^\nu - \nu - 1$ multiplications on these variables; this can be done in $\nu - 1$ rounds.*

Proof. By the previous proposition, we need to construct the monomial x^s for each subset $s \in \{0,1\}^\nu$. We do this inductively: pick i such that the i-th component of s is nonzero, then $x^s = x^{s'} x_i$ where s' is the same as s except that it is zero at the i-th coordinate. Thus x^s is obtained in a single multiplication from $x^{s'}$; from the binomial theorem we now find that the required number of multiplications is equal to the number of degree > 1 terms, which is $\sum_{k=2}^{\nu} \binom{\nu}{k} = 2^\nu - \nu - 1$. Since none of these terms can be obtained "for free" from others (by linear independence), this is the most efficient construction possible. □

Notation 42. Let x be a positive integer of bit-length ν. We then denote by $[x]_{\nu\text{-bit}}$ the set of sharings $[x^s]$ for all $s \in \{0,1\}^\nu$, i.e. sharings of all possible products of the bits of x, but leaving out $[x^{(0,\ldots,0)}]$ and sometimes implicitly replacing it by the constant value 1.

Example 3. Suppose the bit-length ν is 2. As is evident from Example 2, both $p_{c<x}$ and $p_{c=x}$ are linear combinations of the monomials x_1, x_2, x_1x_2. Out of these only x_1x_2 is not of degree 1; it can be constructed in a single multiplication by multiplying x_1 and x_2, and indeed $1 = 2^2 - 2 - 1$. By definition, $[x]_{\nu\text{-bit}} := \{[x_1], [x_2], [x_1x_2]\}$.

When c and x have bit-length ν, with the former value public and the latter secret-shared, we can locally compute secret-sharings of their function evaluations as follows:

Protocol `PartArithFuncSharing`

$$[f(c,x)] \longleftarrow \mathcal{P}_{\texttt{PartArithFuncSharing}}\big(f_\nu, c, [x]_{\nu\text{-bit}}\big)$$

1. The nodes locally compute a list of integers: $p_f(c,\cdot) \leftarrow \mathcal{P}_{\texttt{PartArithFunc}}\big(f_\nu, c\big)$.
2. They locally compute the secret sharing

$$[f(c,x)] = \sum_{s\in\{0,1\}^\nu} p_f^s(c)\,[x^s].$$

Fig. 6: Protocol to compute a sharing of the partial application of a Boolean function on a public value and a bitwise shared value

Once a value c in $\{0,1\}^\ell$ in a bitwise comparison protocol is revealed, we split it into values lying in $\{0,1\}^\nu$:

Notation 43. For i in $\{1, \ldots, \lceil \ell/\nu \rceil\}$ we let $c_{\nu,i} \in \{0,1\}^\nu$ denote the value obtained by using the ν bits $c_{1+(i-1)\nu}, \ldots, c_{i\nu}$ of c (setting the superfluous bits between ℓ and $\nu\lceil \ell/\nu \rceil$ to zero), and similarly for $x_{\nu,i}$. We then denote by $[x]_{\nu\text{-bit}}(i)$, for each i in $\{1, \ldots, \lceil \ell/\nu \rceil\}$, the following data: all possible products of the sharings of the ν bits $x_{1+(i-1)\nu}, \ldots, x_{i\nu}$ of $x_{\nu,i}$. We let $[x]_{\nu\text{-bit}}$ denote the collection of this data for all i in $\{1, \ldots, \lceil \ell/\nu \rceil\}$.

For each i we can then apply f_ν on $c_{\nu,i}$ and $[x]_{\nu\text{-bit}}(i)$ and store the result in an array $\mathbf{f}_\nu^{c,x}$:

Protocol `PartArithFuncSharings`

$$[\mathbf{f}_\nu^{c,x}] \longleftarrow \mathcal{P}_{\texttt{PartArithFuncSharings}}\big(f_\nu, c, [\mathbf{x}]_{\nu\text{-bit}}\big)$$

1. For each i in $\{1, \dots, \lceil \ell/\nu \rceil\}$, the nodes locally compute

$$[\mathbf{f}_\nu^{c,x}(i)] \leftarrow \mathcal{P}_{\texttt{PartArithFuncSharing}}\big(f_\nu, c_{\nu,i}, [\mathbf{x}]_{\nu\text{-bit}}(i)\big).$$

Fig. 7: Protocol to compute multiple sharings of the partial application of a Boolean function on a public value and a bitwise shared value

Definition 1. *Let c and x be two positive integers of bit-length ℓ and pick a natural number $\nu \in \mathbb{N}_1$. By*

$$c \underset{\nu}{\overset{i}{=}} x \qquad \text{and} \qquad c \underset{\nu}{\overset{i}{<}} x$$

we then denote the usual ν-bit comparison operators acting on the i-th tuple of ν bits of c and x as in Notation 43, meaning that if the bits of c and x are given by $c_1, \dots, c_\ell$ and $x_1, \dots, x_\ell$, then it applies those operators on the bits $c_{1+(i-1)\nu}, \dots, c_{i\nu}$ and $x_{1+(i-1)\nu}, \dots, x_{i\nu}$.

Corollary 1. *Pick a natural number $\nu \in \mathbb{N}_1$. Constructing the secret-shared ν-bit operators*

$$[c \underset{\nu}{\overset{i}{=}} x] \qquad \text{and} \qquad [c \underset{\nu}{\overset{i}{<}} x]$$

for a public value c and bitwise-shared value $[x]_{\text{bit}}$ of bit-length ℓ, costs at most $(2^\nu - 1)\lceil \ell/\nu \rceil - \ell$ multiplications, with equality if ν divides ℓ.

Proof. This follows from Proposition 2 and

$$(2^\nu - \nu - 1)\lceil \ell/\nu \rceil \leq (2^\nu - 1)\lceil \ell/\nu \rceil - \ell. \qquad \square$$

Briefly denoting these operators by lt and eq, we finally obtain:

Protocol `nAryComps`

$$([\mathbf{eq}_\nu^{c,z}], [\mathbf{lt}_\nu^{c,z}]) \longleftarrow \mathcal{P}_{\texttt{nAryComps}}\big(c, [\mathbf{x}]_{\nu\text{-bit}}\big)$$

1. For f in $\{\text{eq}, \text{lt}\}$, the nodes locally compute the array $[\mathbf{f}] \leftarrow \mathcal{P}_{\texttt{PartArithFuncSharings}}(f_\nu, c, [\mathbf{x}]_{\nu\text{-bit}})$.

Fig. 8: Protocol to compute a sharing of the least significant bit of a bounded shared value

5 Our bitwise comparison protocols

In this section we describe two bitwise comparison protocols, $\mathcal{P}^{\texttt{2R},\nu}_{\texttt{BitLessThan}}$ and $\mathcal{P}^{\texttt{logR},\nu}_{\texttt{BitLessThan}}$, both of which are parametrised by the bit operator arity ν. The first is based on [45], the second one on [29]. Their bandwidth when compiled into a full comparison protocol using MPC protocol [2] is approximated in Table 1. Since these second family of bitwise comparison protocols can be executed over $\mathbb{Z}/2\mathbb{Z}$, we have added quick communication complexity estimates when doing so using (passive security) edaBits [28].

Remark 1. While the idea of [25] of using "king" nodes doubles the number of communication rounds (which we have neglected in Table 1), the paper [32] showed that they also allow one to subsequently reduce the depth of arithmetic circuits (consisting of only addition and multiplication gates) by half again. The comparison protocols in this paper benefiting from that reduction are the ones under discussion in subsection §5.2. But employing this trick for such protocols yields new protocols which are quite similar in shape to the original ones, and therefore the performance comparisons between the protocols in §5.2 are reasonable.

Protocol	[2] Total bandwidth		Rounds	[2] with edaBits [28] Total bandwidth		Rounds
	Offline	Online		Offline	Online	
$\mathcal{P}^{\mathtt{3R},2}_{\mathtt{LessThan}}$	$18\ell^2$	$1\ell^2$	3			
$\mathcal{P}^{\mathtt{3R},3}_{\mathtt{LessThan}}$	$19.33\ell^2$	$0.67\ell^2$	3			
$\mathcal{P}^{\mathtt{3R},4}_{\mathtt{LessThan}}$	$24\ell^2$	$0.5\ell^2$	3			
$\mathcal{P}^{\mathtt{logR},2}_{\mathtt{LessThan}}$	$8.86\ell^2$	$1.86\ell^2$	$L+1$	$4.5n\ell N$	$1.86\ell N$	$L+2$
$\mathcal{P}^{\mathtt{logR},3}_{\mathtt{LessThan}}$	$11.61\ell^2$	$1.28\ell^2$	$L+1$			

Table 1: Bandwidth approximation of various comparison protocols, in terms of *bits* sent per node; here m is odd, and for the meaning of ℓ, L and N see §2.

5.1 A 2-round protocol

In this subsection we describe a constant-round bitwise comparison protocol, based on [45], resulting in a constant-round comparison protocol $\mathcal{P}^{\mathtt{3R},\nu}_{\mathtt{LessThan}}$ depending on a natural number ν. Correctness is probabilistic when $\nu = 1$, and is perfect otherwise. Concretely, we have made the following two improvements:

- Just like [18, `BitLTC1`] but also deploying it for $\nu > 1$, we use public multiplications to significantly reduce in the honest majority setting the offline and online round complexity of computing the sharing
$$[\![\varphi_\nu^{c,z}]\!] := \sum_{i=1}^{\ell_\nu} [c \overset{i}{\underset{\nu}{<}} z] 2^{\sum_{j=i+1}^{\ell} [c \overset{j}{\underset{\nu}{\neq}} z]}.$$
- We slightly improve the algorithm of [45] that is used to extract the least significant bit of $[\![\varphi_\nu^{c,z}]\!]$. As a side-effect the bit-size of the ring is halved, and hence our total bandwidth is less than that of [45].

The only similar comparison protocols that we are aware of are the 3-round [18, `BitLTC1`] and the 5-round [18, `BitLTC2`], but neither of those are information-theoretically secure (ITS) and their other improvements are minor. In particular, the efficiency gains they obtain from using statistical privacy are relatively small, so in Table 2 we estimate the communication complexity of their protocols *after making the following modifications:* bitwise-shared random values are constructed using a standard ITS algorithm [46] rather than relying on pseudorandom secret sharing, and are used to extract the desired least significant bit (see Figure 10). If preferred the reader can undo these modifications and make the analogous modifications to our protocol; the relative analysis barely changes.

Theorem 2. *The subprotocol decomposition of the protocol $\mathcal{P}^{\mathtt{3R},\nu}_{\mathtt{LessThan}}$ is described in Table 2. If the bit-length ℓ is not too small, its communication complexity when using MPC protocol [2] can be approximated as follows, in terms of elements sent per node in $\mathbb{Z}/m\mathbb{Z}$:*

- *Offline bandwidth:* $6\ell + (8 + 2^{2+\nu})\ell_\nu$.
- *Online bandwidth:* $4 + 2\ell_\nu$.

Example 4. For arity $1 \leq \nu \leq 5$ we used the approximation $\ell_\nu \approx \ell/\nu$ and compared it to the constant-round protocols of [18] in Table 3. All subprotocols take 3 communication rounds for honest majority MPC and 4 rounds for dishonest majority MPC, except [18, `LTZC2`] which takes two extra rounds in either setting. Note that the relative performance of the latter degrades somewhat in the dishonest majority setting due to its use of a dot product.

Protocol	Subprotocols						
	Offline			Online			
	$\mathcal{P}_{\texttt{Rand}}$	$\mathcal{P}_{\texttt{Mult}}$	$\mathcal{P}_{\texttt{PubMult}}$	$\mathcal{P}_{\texttt{Mult}}$	$\mathcal{P}_{\texttt{PubMult}}$	$\mathcal{P}_{\texttt{Reveal}}$	$\mathcal{P}_{\texttt{DotProd}}$
[18, `LTZC1`]	4ℓ	1ℓ	3ℓ	0	ℓ	2	0
[18, `LTZC2`]	3ℓ	0.5ℓ	2.5ℓ	0.5ℓ	0.5ℓ	2	1
$\mathcal{P}^{\texttt{3R},2}_{\texttt{LessThan}}$	3ℓ	1ℓ	2.5ℓ	0	0.5ℓ	2	0
$\mathcal{P}^{\texttt{3R},3}_{\texttt{LessThan}}$	2.67ℓ	1.67ℓ	2.33ℓ	0	0.33ℓ	2	0
$\mathcal{P}^{\texttt{3R},4}_{\texttt{LessThan}}$	2.5ℓ	3ℓ	2.25ℓ	0	0.25ℓ	2	0
$\mathcal{P}^{\texttt{3R},\nu}_{\texttt{LessThan}}$	$2(\ell+\ell_\nu)$	$2^\nu\ell_\nu-\ell$	$2\ell+1\ell_\nu$	0	$1\ell_\nu$	2	0

Table 2: Approximate subprotocol decomposition of some MPC comparison protocols into smaller primitives; for notation ℓ and ℓ_ν see §2. The first two protocols were slightly modified to make them information-theoretically secure. The number of $\mathcal{P}_{\texttt{Rand}}$'s and $\mathcal{P}_{\texttt{PubMult}}$'s decreases by ℓ for all of the $\mathcal{P}^{\texttt{3R},\nu}_{\texttt{LessThan}}$ protocols when m is even, and the input of $\mathcal{P}_{\texttt{DotProd}}$ are two vectors of size ℓ_ν.

Protocol	[18, `LTZC1`]	[18, `LTZC2`]	$\nu=1$	$\nu=2$	$\nu=3$	$\nu=4$	$\nu=5$
Offline	23ℓ	17ℓ	22ℓ	18ℓ	19.33ℓ	24ℓ	33.2ℓ
Online	2ℓ	2ℓ	2ℓ	1ℓ	0.66ℓ	0.5ℓ	0.4ℓ

Table 3: Approximation of the bandwidth of ITS-versions of comparison protocols from [18] and protocol $\mathcal{P}^{\texttt{3R},\nu}_{\texttt{LessThan}}$ for various bit operator arities ν in the setting of MPC protocol [2], in terms of the number of *elements* of $\mathbb{Z}/m\mathbb{Z}$ sent per node. All protocols have 3 communication rounds, except for [18, `LTZC2`] which has 5.

Remark 2. As we are unsure how exactly [45] intends to compute $[\![\varphi^{c,z}_\nu]\!]$ for $\nu > 1$ (but does note that $\nu = 2$ is more efficient), we have left it out. It is possible that $\mathcal{P}^{\texttt{3R},2}_{\texttt{LessThan}}$ is quite similar to the "intented" protocol of [45] (but with half the ring bit-length), but as our protocol seems to outperform the two constant-round protocols of the later [18] (its 3-round protocol is still the one used in [36] and [17]) it appears that this intention went unnoticed.

Finally,we note that [7] provides a 4-round comparison protocol by executing the bitwise comparison using garbled circuits, but both the offline and online bandwidth will be significantly higher; for the latter, observe that keys need to be broadcasted by each node for each of the ℓ bits, resulting in $\ell\kappa(n-1)$ bits, where κ is the computational security parameter. As usually $\kappa \geq 40$ and $\ell \leq 64$, we usually have $\ell\kappa(n-1) \geq 2\ell^2$. With regards to preprocessing, note that [7] uses a circuit which is optimised for a low number of rounds (though as we noted earlier, not the best one to use for that purpose), whereas in the garbled setting it would be better to use a circuit which is focused on minimising the number of multiplications. (Since it uses a pseudorandom function, this method moreover is not information-theoretically secure).

In the next two subsections we explain our considerations in-depth, before presenting our protocol.

5.1.1 On prefix multiplications and public dot products

Although phrased somewhat ambiguously, [45] seems to employ the identity $x^b = 1 + (x-1)b$ for bits b with $x = 2$ and also $x = 2^{-2}$ to first decompose

$$\begin{aligned}
2^{\sum_{j=i+1}^{\ell}[c \overset{j}{\neq} z]} &= 2^{\sum_{j=i+1}^{\ell} c_j + [z_j] - 2c_j[z_j]} \\
&= 2^{\sum_{j=i+1}^{\ell} c_j} 2^{\sum_{j=i+1}^{\ell}[z_j]} 2^{-2\sum_{j=i+1}^{\ell} c_j[z_j]} \\
&= \prod_{j=i+1}^{\ell} 2^{c_j} \prod_{j=i+1}^{\ell} 2^{[z_j]} \prod_{j=i+1}^{\ell} (2^{-2})^{c_j[z_j]} \\
&= \prod_{j=i+1}^{\ell} (1 + c_j) \prod_{j=i+1}^{\ell} (1 + [z_j]) \prod_{j=i+1}^{\ell} \big(1 + (2^{-2} - 1)c_j[z_j]\big)
\end{aligned}$$

and then compute these products as follows:

- The first product $\prod_{j=i+1}^{\ell}(1 + c_j)$ is computed online, locally.
- The second product $\prod_{j=i+1}^{\ell}(1 + [z_j])$ is computed online through a standard prefix multiplication algorithm [9,23].
- The third product $\prod_{j=i+1}^{\ell}(1 - \frac{3}{4}c_j[z_j])$ is computed online through a modified prefix multiplication algorithm: offline for each j there are two multiplications of $1 - \frac{3}{4}c_j[z_j]$ with its prefix "mask", namely for each c_j in $\{0, 1\}$, and online only the share is revealed corresponding to the c_j that appears.

Thus this algorithm requires doing two prefix multiplications of size ℓ in parallel, one of which has extra preprocessing, and then a final multiplication is required (which can be preprocessed) for each j to multiply the second and third product. We note that:

- The preprocessing of two multiplications for the third product is not necessary when using public multiplications as in [18], as the public multiplication of $1 - \frac{3}{4}c_j[z_j]$ with its mask happens online instead.
- In fact, for the same reason the expressions

$$2^{\sum_{j=i+1}^{\ell_\nu}[c \overset{j}{\underset{\nu}{\neq}} z]} = \prod_{j=i+1}^{\ell_\nu} (1 + [c \overset{j}{\underset{\nu}{\neq}} z]) \tag{4}$$

  can be computed using just one prefix multiplication algorithm of size ℓ_ν.

For $\nu = 1$ these observations are also implicit in [18, `BitLTC1`], but we will end up using them for $\nu > 1$. Another feature of [18, `BitLTC1`] is to locally compute the entire sharing $[\varphi_\nu^{c,z}]$ from this prefix multiplication, however:

- This trick does not seem to extend to $\nu > 1$, but setting $\nu = 2$ is a major improvement to offline and online efficiency.
- The next step is to execute a reveal algorithm; we will merge that with the dot product required to turn (4) into $[\varphi_\nu^{c,z}]$, yielding a public dot product which is almost as efficient and works for arbitrary ν.

The 5-round [18, `BitLTC2`] protocol is somewhat reminiscent of our $\nu = 2$ protocol but by doing too much work in the online phase that protocol seems to partially miss the point that [45] was trying to make.

In the dishonest majority setting, the final dot product instead requires $\lceil \ell/\nu \rceil$ multiplications (which can be preprocessed into the prefix multiplication to save one round):

Protocol CompPhi

$$[\![\varphi_\nu^{c,z}]\!] \longleftarrow \mathcal{P}_{\texttt{CompPhi}}\big(c, [\mathbf{z}]_{\nu\text{-bit}}\big)$$

1. The nodes locally compute $([\mathbf{neq}_\nu^{c,z}], [\mathbf{lt}_\nu^{c,z}]) \leftarrow \mathcal{P}_{\texttt{nAryComps}}(c, [\mathbf{z}]_{\nu\text{-bit}})$, then locally compute the array $[\mathbf{1}+\mathbf{neq}_i] := 1 + [\mathbf{neq}_i]$.
2. The nodes run a prefix multiplication [18] to compute
$$\big([1+\mathrm{neq}_{\ell_\nu}], \dots, [\prod_{i=2}^{\ell_\nu} 1+\mathrm{neq}_i]\big) \longleftarrow \mathcal{P}_{\texttt{PrefixMult}}\big([\mathbf{1}+\mathbf{neq}]\big),$$
where the ordering of the array $[\mathbf{1}+\mathbf{neq}]$ is reversed and its last element is dropped.
3. Renaming this array $\big([a_{\ell_\nu}], \dots, [a_1]\big)$ and setting $a_{\ell_\nu} := 1$, the nodes locally compute the dot product
$$[\![\varphi_\nu^{c,z}]\!] = \sum_{i=1}^{\ell_\nu} [a_i][\mathrm{lt}_i].$$

Fig. 9: Protocol to compute a (high degree) sharing of $\varphi_\nu^{c,z}$

5.1.2 Extracting the least significant bit

When m is even, a sharing of the least significant bit can be obtained easily (which is well-known) by computing the residue modulo 2 of the shares.

When m is odd the situation is more complicated. In order to avoid a circular situation (extracting a least-significant-bit to extract a least-significant-bit, etc.), a bound on $\varphi_\nu^{c,z}$ is used to extract its least significant bit with a simple method. [45] requires the bound $\varphi_\nu^{c,z} < \sqrt{4m}$, which roughly doubles the bandwidth required, whereas we merely require that $2^{\ell-1} + 2^{\ell_\nu} \leq m \leq 2^\ell$, a fairly innocent requirement on m when $\nu > 1$. We will use

Proposition 3. *We have* $0 \leq \varphi_\nu^{c,z} \leq 2^{\ell_\nu} - 1$.

Proof. By definition

$$\varphi_\nu^{c,z} = \sum_{i=1}^{\ell_\nu} (c \overset{i}{\underset{\nu}{<}} z)\, 2^{\sum_{j=i+1}^{\ell_\nu} c \overset{j}{\underset{\nu}{\neq}} z} \leq \sum_{i=1}^{\ell_\nu} 2^{\sum_{j=i+1}^{\ell_\nu} 1} = \sum_{i=1}^{\ell_\nu} 2^{i-1} = 2^{\ell_\nu} - 1. \qquad \square$$

Remark 3. (i) If $\nu = 1$ and $m \neq 2^\ell$ then this range does not fit inside of $\mathbb{Z}/m\mathbb{Z}$ so an overflow might occur, implying that this protocol only has statistical correctness.
(ii) If ν is large a small improvement in bandwidth can potentially be gained by generating all bits in a ring containing 2^{ℓ_ν}.

Proposition 4. *Pick a pair of natural numbers $\ell'' < \ell'$ and let $2^{\ell'-1} + 2^{\ell''} \leq m \leq 2^{\ell'}$. Pick any numbers $0 \leq \varphi < 2^{\ell''}$ and $0 \leq r < m$, and let $0 \leq c < m$ be the remainder of $\varphi + r$ modulo m. Then there is an identity of Boolean values*
$$(c < r) = r_{\ell'}\big(1 - c_{\ell'}\big).$$

Proof. The identity says that the Boolean $c < r$ is equal to 1 if and only if $r_{\ell'} = 1$ and $c_{\ell'} = 0$ both hold. If the latter conditions hold then indeed $c < 2^{\ell'-1} \leq r$. In the converse direction, since $\varphi + r > r$ it follows from $c < r$ that $\varphi + r > m$. If $r_{\ell'} = 0$ then $\varphi + r < 2^{\ell''} + 2^{\ell'-1} < m$ which is a contradiction, so $r_{\ell'} = 1$. But also $\varphi + r < 2^{\ell'-1} + m$ which yields $c_{\ell'} = 0$. $\square$

Corollary 2. *Suppose m is odd. Then the least significant bit φ_1 of such φ is given by*

$$\varphi_1 = c_{\ell'}(c_1 \oplus r_1) + (1 - c_{\ell'})(c_1 \oplus r_1 \oplus r_{\ell'}),$$

and given another bit r'_1 we have

$$r'_1 \oplus \varphi_1 = c_{\ell'}(c_1 \oplus r'_1 \oplus r_1) + (1 - c_{\ell'})(c_1 \oplus r'_1 \oplus r_1 \oplus r_{\ell'}),$$

Proof. For arbitrary m, the least significant bit is given by

$$\varphi_1 = c_1 \oplus r_1 \oplus \delta_m(c < r) = c_1 \oplus r_1 \oplus \delta_m r_{\ell'} \big(1 - c_{\ell'}\big),$$

where δ_m is 1 if m is odd and 0 otherwise. For odd m, both claims now follows from the bit identity $a \oplus bc = a + bc - 2abc = a(1-c) + (a \oplus b)c$. □

In the next protocol, we assume that m is odd and that a sharing $[r_1 \oplus r_{\ell'}]$ was already computed during preprocessing. In [45] an additional multiplication is preprocessed, but we won't have to.

Protocol `LSBBounded`

$$[\varphi_1] \longleftarrow \mathcal{P}_{\texttt{LSBBounded}}\big([\![\varphi]\!]\big)$$

1. The nodes compute a shifted decomposition $(\varphi + r, [r]_{\text{bit}}) \leftarrow \mathcal{P}_{\texttt{ShiftedBitDecomp}}(\varphi, 1)$, and set $c := \varphi + r$.
2. They locally compute
$$[\varphi_1] = c_{\ell'}(c_1 \oplus [r_1]) + \big(1 - c_{\ell'}\big)(c_1 \oplus [r_1 \oplus r_{\ell'}]).$$

Fig. 10: Protocol to compute a sharing of the least significant bit of a bounded shared value

This yields

Protocol `BitLessThan`

$$[c < z] \longleftarrow \mathcal{P}^{\texttt{2R},\nu}_{\texttt{BitLessThan}}\big(c, [z]_{\nu\text{-bit}}\big)$$

1. The nodes compute $[\![\varphi^{c,z}_\nu]\!] \leftarrow \mathcal{P}_{\texttt{CompPhi}}\big(c, [z]_{\nu\text{-bit}}\big)$.
2. The nodes then compute $[c < z] \leftarrow \mathcal{P}_{\texttt{LSBBounded}}\big([\![\varphi^{c,z}_\nu]\!]\big)$.

Fig. 11: Protocol to compute a sharing of the comparison of a public value and bitwise shared value

If we were to directly use this bitwise comparison protocol, we would end up with

$$[z < 0] = c_1 \oplus [r_1] \oplus [c < r]$$

which would cost another round of communication to compute. This will be avoided by doing a bit of preprocessing: again assume that m is odd and now suppose that sharings $[r'_1 \oplus r_1]$ and $[r'_1 \oplus r_1 \oplus r_{\ell'}]$ are already computed during preprocessing, where $[r'_1]$ is the first bit of another bitwise shared random value $[r']_{\text{bit}}$.

Protocol LSB

$$[z_1] \longleftarrow \mathcal{P}^{\mathtt{3R},\nu}_{\mathtt{LSB}}(\llbracket z \rrbracket)$$

1. The nodes compute a shifted decomposition $(z+r, [r]_{\nu\text{-bit}}) \leftarrow \mathcal{P}_{\mathtt{ShiftedBitDecomp}}(z,\nu)$, and set $c := z+r$.
2. The nodes compute $\llbracket \varphi \rrbracket \leftarrow \mathcal{P}_{\mathtt{CompPhi}}(c, [r]_{\nu\text{-bit}})$.
3. The nodes compute another shifted decomposition $(\varphi + r', [r']_{\text{bit}}) \leftarrow \mathcal{P}_{\mathtt{ShiftedBitDecomp}}(\varphi, 1)$ and set $c' := \varphi + r'$.
4. They locally compute

$$[z_1] = c_{\ell'}(c_1 \oplus [r'_1 \oplus r_1]) + \big(1 - c_{\ell'}\big)(c_1 \oplus [r'_1 \oplus r_1 \oplus r_{\ell'}]).$$

Fig. 12: Protocol to compute a sharing of the least significant bit

Following, for example, the second approach in §3.2, the protocol $\mathcal{P}^{\mathtt{3R},\nu}_{\mathtt{LessThanZero}}$ can now be implemented as follows:

5.2 A logarithmic-round protocol

Finally, we turn towards the remaining bit comparison protocols [18, `LTZL`] and [22,41,27]. As explained in §3, their protocols share the following properties:

- The number of rounds is $\log(\ell)$,
- Their offline and online bandwidth are linear in ℓ,
- The bit computations can be done over $\mathbb{Z}/2\mathbb{Z}$ if desired.

The protocol [22, `BitLT`] is identical to [18, `BitLTL`], whereas the protocol [27, `BitComp`] comes from [29, §3.1]. We use the latter circuit but obtain the following improvements (when setting $\nu = 2$):

- The number of rounds decreases by 1.
- Total bandwidth drops by more than 25%.
- Online bandwidth drops by more than 50%.

The resulting *bitwise* comparison protocol compares to its brethren as follows:[7]

Bitwise comparison	[41, `LTBits`]	[18, `BitLTL`]	[29, §3.1]	$\mathcal{P}^{\mathtt{logR},2}_{\mathtt{LessThan}}$
Offline $\mathcal{P}_{\mathtt{Mult}}$	0	0	0	0.5ℓ
Online $\mathcal{P}_{\mathtt{Mult}}$	$0.5L\ell$	2ℓ	2ℓ	0.93ℓ
Number of rounds	L	L	L	$L-1$

Table 4: Approximate decomposition of various bit comparison protocols into smaller primitives. For the meaning of ℓ and L see §2.

As these protocols consist only of addition and multiplication gates, the number of communication rounds can be halved with [32] and the bandwidth decreased by performing bit computations over $\mathbb{Z}/2\mathbb{Z}$. But since the effects of these changes on these protocols will be more-or-less "proportional", it should not change the comparison between these protocols. See also Remark 1.

[7] [41] writes that they are using a prefix OR protocol from [18] to compute the circuit of [23]. Since their computation takes place over $\mathbb{Z}/2\mathbb{Z}$ this prefix OR protocol is not [18, `PreOrC`], so we presume it is a specialisation of [18, `PreOpL`]. It is explicitly stated on [18, p. 10] however that this would lead to a less efficient algorithm than [18, `BitLTL`]; Table 4 suggests that this is true when the bit-length (which we assume is a power of 2) is at least 16.

5.2.1 Implementing higher arity bit operations

The most efficient of the circuits in §3.3 requires $2\ell - \log(\ell) - 2$ multiplications. One of these is the circuit of [29], which can be interpreted as being composed of various bit operations; modifying those to act on multiple bits simultaneously just as in §4 yields:

Theorem 3. *Pick a natural number $\nu \in \mathbb{N}_1$. The secure comparison $[c < x]$ for a public value c and bitwise-shared value $[x]_{\mathrm{bit}}$ of bit-length ℓ can be computed (in the worst-case) with the following communication costs:*

- *The number of online rounds is $\lceil \log(\ell_\nu) \rceil$.*
- *The number of offline multiplications is $(2^\nu - 1)\ell_\nu - \ell$.*
- *The number of online multiplications is $2\ell_\nu - \log(\ell_\nu) - 2$.*

So (in the worst-case) the total number of multiplications is: $(2^\nu + 1)\ell_\nu - \ell - \log(\ell_\nu) - 2$.

Proof. For simplicity we will estimate the complexity of comparing integers of bit-length $\nu\ell_\nu \geq \ell$. The number of offline multiplications comes from Corollary 1. The goal is to compute

$$[c < z] = [c \overset{\ell_\nu}{\underset{\nu}{<}} z] + [c \overset{\ell_\nu - 1}{\underset{\nu}{<}} z][c \overset{\ell_\nu}{\underset{\nu}{=}} z] + \cdots + [c \overset{1}{\underset{\nu}{<}} z] \prod_{k=2}^{\ell_\nu} [c \overset{k}{\underset{\nu}{=}} z].$$

Following [29], we can compute the right-hand-side in $\lceil \log \ell_\nu \rceil$ rounds with $2\ell_\nu - \log(\ell_\nu) - 2$ multiplications through the following recursive algorithm: for each i in $\lceil \ell_\nu/2 \rceil$, set

$$a_i := [c \overset{2i}{\underset{\nu}{<}} z] + [c \overset{2i-1}{\underset{\nu}{<}} z] \cdot [c \overset{2i}{\underset{\nu}{=}} z] \qquad \text{and} \qquad b_i := [c \overset{2i}{\underset{\nu}{=}} z] \cdot [c \overset{2i-1}{\underset{\nu}{=}} z]. \tag{5}$$

Then

$$[c < z] = a_{\ell_\nu/2} + a_{\ell_\nu/2-1} b_{\ell_\nu/2} + \cdots + a_1 \prod_{k=2}^{\ell_\nu/2} b_k.$$

This halves the size of the circuit and inductively leads to a total of $2(\ell_\nu - 1)$ multiplications, which we can reduce by $\log(\ell_\nu)$ if we skip the computation of the final b_i in each round. $\square$

We illustrate these values for low ν in the following table, where for simplicity we assume that $\ell = 2^L$ is a power of 2 and assume that it is not too small so that we can ignore lower-order terms:

Bit arity ν	Number of rounds	Offline $\mathcal{P}_{\texttt{Mult}}$	Online $\mathcal{P}_{\texttt{Mult}}$	Total $\mathcal{P}_{\texttt{Mult}}$
1	L	0	2ℓ	2ℓ
2	$L-1$	0.5ℓ	ℓ	1.5ℓ
3	$L-1$	1.33ℓ	0.66ℓ	2ℓ
4	$L-2$	2.75ℓ	0.5ℓ	3.25ℓ
5	$L-2$	5.2ℓ	0.4ℓ	5.6ℓ

Table 5: Worst-case communication complexity of computing the secure comparison of a public value and a bitwise shared value for various ν. Here ν denotes the number of bits to be compared simultaneously and the last three columns denote the number of invocations of the subprotocol $\mathcal{P}_{\texttt{Mult}}$ which multiplies two secret values.

5.2.2 Redundant multiplications

Now let us briefly describe the second optimisation. If c and x are two positive integers of bit-length ν and c turns out to be equal to the maximum value $2^\nu - 1$, then the Boolean value $c < x$ is zero, regardless of what the value of x is. If c is uniformly random, the probability of this happening is $2^{-\nu}$. The same is then true for the integers $c_{\nu,i}$ constructed from c of bit-length ℓ, if ℓ is divisible by ν and c is uniformly random. The latter holds in every in every[8] subprotocol from section §3 for the public input value c of the bitwise comparisons. We will use this observation to reduce the number of multiplications required to compute the protocol (5).

Definition 2. *If given such a public value c it follows that a multiplication in (5) can be skipped, we call it* redundant*.*

Now let us turn to the online phase of the bitwise comparison protocol. The following quantity will arise:

Definition 3. *Assume that ℓ is a power of 2 and for $1 \leq \nu \leq \ell$ consider the real number*

$$\gamma_{\nu,\ell} := \frac{1}{\nu} \sum_{i=1}^{\lceil \log(\ell_\nu) \rceil} 2^{-2^{i-1}\nu - i}.$$

For fixed ν, the sequence $\gamma_{\nu,1}, \gamma_{\nu,2}, \ldots$ converges as it is dominated by the geometric series with ratio 1/2.

Example 5. Suppose that $\nu = 1$. Then

$$0.25 \leq \gamma_{1,\ell} < 0.321.$$

Example 6. Suppose that $\nu = 2$. Then

$$0.0625 \leq \gamma_{2,\ell} < 0.0706.$$

Lemma 1. *The a_i in layer $j \geq 0$ of the circuit of Theorem 3 is zero when the $2^j\nu$ bits of c at the sequential coordinates $(\!(j, \nu, i)\!) := (2^j\nu i + 1 - 2^j\nu, \ldots, 2^j\nu i)$ are equal to 1. Here in layer 0 we set $a_i := (c \overset{i}{\underset{\nu}{<}} x)$.*

Proof. We may split the sequence $(\!(j, \nu, i)\!)$ into two disjoint components

$$(\!(j, \nu, i)\!) = (\!(j-1, \nu, 2i-1)\!) \oplus (\!(j-1, \nu, 2i)\!),$$

which follows from the identities

$$\begin{aligned} 2^{j-1}\nu(2i-1) + 1 - 2^{j-1}\nu &= 2^j\nu i + 1 - 2^j\nu, \\ 2^{j-1}\nu(2i-1) + 1 &= 2^{j-1}\nu(2i) + 1 - 2^{j-1}\nu, \\ 2^{j-1}\nu(2i) &= 2^j\nu i. \end{aligned}$$

We prove the claim by induction. Specialising to $j = 1$ in this decomposition yields that the set of bits of c corresponding to $(\!(1, \nu, i)\!)$ is $\{c_{\nu,2i-1}, c_{\nu,2i}\}$, which implies that $c \overset{2i-1}{\underset{\nu}{<}} x$ and $c \overset{2i}{\underset{\nu}{<}} x$ are zero, and hence so is $a_i := [c \overset{2i}{\underset{\nu}{<}} x] + [c \overset{2i-1}{\underset{\nu}{<}} x] \cdot [c \overset{2i}{\underset{\nu}{=}} x]$. For $j > 1$ the induction hypothesis implies that the a_{2i} and a_{2i-1} of layer $j - 1$ are zero, proving the claim. □

Lemma 2. *Let c be a uniformly random public value and $[x]_{\mathrm{bit}}$ be a bitwise-shared value of bit-length ℓ, pick a natural number $\nu \in \mathbb{N}_1$ and consider the circuit of (5). If ℓ_ν is a power of 2 then the expected number of redundant multiplications is $\gamma_{\nu,\ell}\ell$.*

[8] The only exception is [18], due to statistical slack, but even there the bits of c are approximately uniformly random so the remainder of our analysis still holds.

If ℓ_ν is not a power of 2, the precise result is slightly more complex but quite similar.

Proof. First let us count the number of multiplication gates in each layer of (5): it has $\log \ell_\nu$ layers, and the number of multiplication gates in the j-th layer is $\ell/2^{j-1}\nu$. Only half of them can become redundant, namely those corresponding to the a_i computations, so that leaves $\ell/2^j\nu$ gates in layer j. For convenience let the 0-th layer denote the input $c_1, \ldots, c_\nu$, which we also denote by a_i.

Now consider the computation of a_i in layer $j \geq 1$. Its multiplication gate is redundant precisely when the a_{2i-1} of the previous layer is zero, which by the previous lemma happens precisely when $2^{j-1}\nu$ bits of c are equal to 1. This leaves $\ell - 2^{j-1}\nu$ bits freely. We can now compute the expected number of redundant multiplications as

$$\mathbb{E}(\text{redundant mults}) = 2^{-\ell} \sum_{i=1}^{\log \ell_\nu} \frac{\ell}{\nu} 2^{-i} 2^{\ell - 2^{i-1}\nu} = \frac{\ell}{\nu} \sum_{i=1}^{\log \ell_\nu} 2^{-2^{i-1}\nu - i} = \gamma_{\nu,\ell}\ell. \qquad \square$$

Thus we can improve the online complexity of the protocol described in Theorem 3 to:

Theorem 4. *Let ℓ denote the bit-length and pick a natural number $\nu \in \mathbb{N}_1$, and assume for simplicity that ℓ/ν is a power of 2. The secure comparison $[c < x]$ for a public value c and bitwise-shared value $[x]_{\text{bit}}$ can be computed with the following average communication costs:*

- *The number of online rounds is* $\log(\ell/\nu)$.
- *The number of offline multiplications is* $(2^\nu - 1)\ell/\nu - \ell$.
- *The number of online multiplications is* $2\ell/\nu - \gamma_{\nu,\ell}\ell - \log(\ell_\nu) - 2$.

So on average the total number of multiplications is: $(2^\nu + 1)\ell/\nu - (\gamma_{\nu,\ell} + 1)\ell - \log(\ell_\nu) - 2$.

As the value of $\gamma_{\nu,\ell}$ drops rapidly when ν increases, Table 6 shows that for $\nu = 5$ the improvement in online communication complexity is becoming negligible:

Bit arity ν	Number of rounds	Offline $\mathcal{P}_{\texttt{Mult}}$	Online $\mathcal{P}_{\texttt{Mult}}$	Total $\mathcal{P}_{\texttt{Mult}}$
1	L	0	1.68ℓ	1.68ℓ
2	$L-1$	0.5ℓ	0.93ℓ	1.43ℓ
3	$L-1$	1.33ℓ	0.64ℓ	1.98ℓ
4	$L-2$	2.75ℓ	0.49ℓ	3.24ℓ
5	$L-2$	5.2ℓ	0.4ℓ	5.6ℓ

Table 6: Average communication complexity of computing the secure comparison of a public value and a bitwise shared value for various ν. Here ν denotes the number of bits to be compared simultaneously and the last three columns denote the number of invocations of the ideal functionality $\mathcal{F}_{\texttt{Mult}}$ of multiplication.

Protocol `BitLessThan`

$$[c < x] \longleftarrow \mathcal{P}^{\mathtt{logR},\nu}_{\mathtt{BitLessThan}}(c, [x]_{\nu\text{-bit}})$$

1. The nodes locally compute: $([\mathbf{eq}^{c,z}_{\nu}], [\mathbf{lt}^{c,z}_{\nu}]) \leftarrow \mathcal{P}_{\mathtt{nAryComps}}(c, [x]_{\nu\text{-bit}})$.
2. Let `Zeroes` $:= \{i \in \{1, \ldots, \lceil \ell/\nu \rceil\} : c_{\nu,i} = 2^{\nu} - 1\}$ and define $\ell' := \lceil \lceil \ell/\nu \rceil /2 \rceil = \lceil \ell/2\nu \rceil$. While $\ell' > 1$:
   (a) In parallel, for j in $1, \ldots, \lfloor \ell'/2 \rfloor$ the nodes compute $[\widetilde{\mathrm{eq}}_j] \leftarrow \mathcal{P}_{\mathtt{Mult}}([\mathrm{eq}_{2j}], [\mathrm{eq}_{2j-1}])$ and for i in $1, \ldots, \lfloor \ell'/2 \rfloor$:
   - If $2i - 1$ is not in `Zeroes`, they also compute
   $$[\widetilde{\mathrm{lt}}_i] \leftarrow \mathcal{P}_{\mathtt{Mult}}([\mathrm{eq}_{2i}], [\mathrm{lt}_{2i-1}]).$$
   - Otherwise, set $[\widetilde{\mathrm{lt}}_i] := 0$.

   (b) For i in $1, \ldots, \lfloor \ell'/2 \rfloor$, the nodes rename:
   $$[\mathrm{lt}_i] := [\mathrm{lt}_{2i}] + [\widetilde{\mathrm{lt}}_i] \qquad \text{and} \qquad [\mathrm{eq}_i] := [\widetilde{\mathrm{eq}}_i].$$
   (c) – If ℓ' is even, the nodes set $\ell' := \ell'/2$.
   – If ℓ' is odd, the nodes set $\ell' := \lceil \ell'/2 \rceil$ and $[\mathrm{eq}_{\ell'}] := [\mathrm{eq}_{2\ell'-1}]$ and $[\mathrm{lt}_{\ell'}] := [\mathrm{lt}_{2\ell'-1}]$.
3. The nodes set $[c < x] := [\mathrm{lt}_1]$.

Fig. 13: Protocol to compute a sharing of the comparison of a public value and bitwise shared value

Protocol `LSB`

$$[z_1] \longleftarrow \mathcal{P}^{\mathtt{logR},\nu}_{\mathtt{LSB}}([\![z]\!])$$

1. The nodes compute a shifted decomposition $(z + r, [r]_{\nu\text{-bit}}) \leftarrow \mathcal{P}_{\mathtt{ShiftedBitDecomp}}(z, \nu)$, and set $c := z + r$.
2. The nodes compute $[w] \leftarrow \mathcal{P}^{\mathtt{logR},\nu}_{\mathtt{BitLessThan}}(c, [r]_{\nu\text{-bit}})$.
3. The nodes locally compute the bit XOR $[c_1 \oplus r_1]$ and then compute $[v] := [w] \cdot [c_1 \oplus r_1]$.
4. They locally compute
$$[z_1] = [w] + [c_1 \oplus r_1] - 2[v].$$

Fig. 14: Protocol to compute a sharing of the least significant bit

References

1. Biometric authentication & cryptography. `https://keyless.io/technology/authentication-cryptography`, accessed: 2023-07-17
2. More Randomness Extraction for Information-Theoretic MPC, forthcoming (draft available on request).
3. Spectrum auctions. `https://partisia.com/better-market-solutions/spectrum-auctions/`, accessed: 2023-07-17
4. 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). `https://doi.org/10.1007/978-3-030-64840-4_6`
5. Abspoel, M., Cramer, R., Damgård, I., Escudero, D., Yuan, C.: Efficient information-theoretic secure multiparty computation over $\mathbb{Z}/p^k\mathbb{Z}$ via galois rings. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019, Part I. LNCS, vol. 11891, pp. 471–501. Springer, Heidelberg (Dec 2019). `https://doi.org/10.1007/978-3-030-36030-6_19`
6. Aliasgari, M., Blanton, M., Zhang, Y., Steele, A.: Secure computation on floating point numbers. In: NDSS 2013. The Internet Society (Feb 2013)
7. Aly, A., Nawaz, K., Salazar, E., Sucasas, V.: Through the looking-glass: Benchmarking secure multi-party computation comparisons for ReLU 's. In: Beresford, A.R., Patra, A., Bellini, E. (eds.) CANS 22. LNCS, vol. 13641, pp. 44–67. Springer, Heidelberg (Nov 2022). `https://doi.org/10.1007/978-3-031-20974-1_3`

8. Archer, D.W., Bogdanov, D., Lindell, Y., Kamm, L., Nielsen, K., Pagter, J.I., Smart, N.P., Wright, R.N.: From keys to databases–real-world applications of secure multi-party computation. The Computer Journal **61**(12), 1749–1771 (2018)
9. Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in constant number of rounds of interaction. In: Rudnicki, P. (ed.) 8th ACM PODC. pp. 201–209. ACM (Aug 1989). https://doi.org/10.1145/72981.72995
10. Baum, C., Cozzo, D., Smart, N.P.: Using TopGear in overdrive: A more efficient ZKPoK for SPDZ. In: Paterson, K.G., Stebila, D. (eds.) SAC 2019. LNCS, vol. 11959, pp. 274–302. Springer, Heidelberg (Aug 2019). https://doi.org/10.1007/978-3-030-38471-5_12
11. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC. pp. 1–10. ACM Press (May 1988). https://doi.org/10.1145/62212.62213
12. Bogdanov, D., Jõemets, M., Siim, S., Vaht, M.: Privacy-preserving tax fraud detection in the cloud with realistic data volumes. T-4-24, Cybernetica AS (2016)
13. Bogdanov, D., Kamm, L., Kubo, B., Rebane, R., Sokk, V., Talviste, R.: Students and taxes: a privacy-preserving study using secure computation. PoPETs **2016**(3), 117–135 (Jul 2016)
14. Bogdanov, D., Talviste, R., Willemson, J.: Deploying secure multi-party computation for financial data analysis - (short paper). In: Keromytis, A.D. (ed.) FC 2012. LNCS, vol. 7397, pp. 57–64. Springer, Heidelberg (Feb / Mar 2012)
15. Bogetoft, P., Christensen, D.L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J.D., Nielsen, J.B., Nielsen, K., Pagter, J., Schwartzbach, M.I., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (Feb 2009)
16. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: 42nd FOCS. pp. 136–145. IEEE Computer Society Press (Oct 2001). https://doi.org/10.1109/SFCS.2001.959888
17. Catrina, O.: Complexity and performance of secure floating-point polynomial evaluation protocols. In: Bertino, E., Shulman, H., Waidner, M. (eds.) ESORICS 2021, Part II. LNCS, vol. 12973, pp. 352–369. Springer, Heidelberg (Oct 2021). https://doi.org/10.1007/978-3-030-88428-4_18
18. Catrina, O., de Hoogh, S.: Improved primitives for secure multiparty integer computation. In: Garay, J.A., Prisco, R.D. (eds.) SCN 10. LNCS, vol. 6280, pp. 182–199. Springer, Heidelberg (Sep 2010). https://doi.org/10.1007/978-3-642-15317-4_13
19. Catrina, O., Saxena, A.: Secure computation with fixed-point numbers. In: Sion, R. (ed.) FC 2010. LNCS, vol. 6052, pp. 35–50. Springer, Heidelberg (Jan 2010)
20. Chokparova, Z., Becher, K., Klose, A., Strufe, T., Urbas, L.: Cryptographic protocol for privacy-preserving integration of HAZOPs in modular process plants. Computers & Chemical Engineering **176**, 108295 (2023). https://doi.org/https://doi.org/10.1016/j.compchemeng.2023.108295
21. Damgård, I., Damgård, K., Nielsen, K., Nordholt, P.S., Toft, T.: Confidential benchmarking based on multiparty computation. In: Grossklags, J., Preneel, B. (eds.) FC 2016. LNCS, vol. 9603, pp. 169–187. Springer, Heidelberg (Feb 2016)
22. Damgård, I., Escudero, D., Frederiksen, T.K., Keller, M., Scholl, P., Volgushev, N.: New primitives for actively-secure MPC over rings with applications to private machine learning. In: 2019 IEEE Symposium on Security and Privacy. pp. 1102–1120. IEEE Computer Society Press (May 2019). https://doi.org/10.1109/SP.2019.00078
23. Damgård, I., Fitzi, M., Kiltz, E., Nielsen, J.B., Toft, T.: Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 285–304. Springer, Heidelberg (Mar 2006). https://doi.org/10.1007/11681878_15
24. Damgård, I., Nielsen, J.B.: Universally composable efficient multiparty computation from threshold homomorphic encryption. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 247–264. Springer, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_15
25. 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). https://doi.org/10.1007/978-3-540-74143-5_32
26. 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). https://doi.org/10.1007/978-3-642-32009-5_38
27. Duan, X., Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ACCO: Algebraic computation with comparison. In: Proceedings of the 2021 on Cloud Computing Security Workshop. pp. 21–38 (2021)
28. Escudero, D., Ghosh, S., Keller, M., Rachuri, R., Scholl, P.: Improved primitives for MPC over mixed arithmetic-binary circuits. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part II. LNCS, vol. 12171, pp. 823–852. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_29

29. Garay, J.A., Schoenmakers, B., Villegas, J.: Practical and secure solutions for integer comparison. In: Okamoto and Wang [43], pp. 330–342. https://doi.org/10.1007/978-3-540-71677-8_22
30. Gennaro, R., Rabin, M.O., Rabin, T.: Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In: Coan, B.A., Afek, Y. (eds.) 17th ACM PODC. pp. 101–111. ACM (Jun / Jul 1998). https://doi.org/10.1145/277697.277716
31. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or A completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC. pp. 218–229. ACM Press (May 1987). https://doi.org/10.1145/28395.28420
32. Goyal, V., Li, H., Ostrovsky, R., Polychroniadou, A., Song, Y.: ATLAS: Efficient and scalable MPC in the honest majority setting. In: Malkin, T., Peikert, C. (eds.) CRYPTO 2021, Part II. LNCS, vol. 12826, pp. 244–274. Springer, Heidelberg, Virtual Event (Aug 2021). https://doi.org/10.1007/978-3-030-84245-1_9
33. Hillis, W.D., Steele Jr., G.L.: Data parallel algorithms. Communications of the ACM **29**(12), 1170–1183 (1986)
34. Hirt, M., Nielsen, J.B.: Robust multiparty computation with linear communication complexity. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 463–482. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_28
35. Kamm, L., Willemson, J.: Secure floating point arithmetic and private satellite collision analysis. International Journal of Information Security **14**(6), 531–548 (2015)
36. Keller, M.: MP-SPDZ: A versatile framework for multi-party computation. In: Ligatti, J., Ou, X., Katz, J., Vigna, G. (eds.) ACM CCS 2020. pp. 1575–1590. ACM Press (Nov 2020). https://doi.org/10.1145/3372297.3417872
37. Keller, M., Pastro, V., Rotaru, D.: Overdrive: Making SPDZ great again. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part III. LNCS, vol. 10822, pp. 158–189. Springer, Heidelberg (Apr / May 2018). https://doi.org/10.1007/978-3-319-78372-7_6
38. Kerschbaum, F., Biswas, D., de Hoogh, S.: Performance comparison of secure comparison protocols. In: 2009 20th International Workshop on Database and Expert Systems Application. pp. 133–136. IEEE (2009)
39. Lapets, A., Jansen, F., Albab, K.D., Issa, R., Qin, L., Varia, M., Bestavros, A.: Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In: Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies. pp. 1–5 (2018). https://doi.org/10.1145/3209811.3212701
40. Lipmaa, H., Toft, T.: Secure equality and greater-than tests with sublinear online complexity. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) ICALP 2013, Part II. LNCS, vol. 7966, pp. 645–656. Springer, Heidelberg (Jul 2013). https://doi.org/10.1007/978-3-642-39212-2_56
41. Makri, E., Rotaru, D., Vercauteren, F., Wagh, S.: Rabbit: Efficient comparison for secure multi-party computation. In: Borisov, N., Díaz, C. (eds.) FC 2021, Part I. LNCS, vol. 12674, pp. 249–270. Springer, Heidelberg (Mar 2021). https://doi.org/10.1007/978-3-662-64322-8_12
42. Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto and Wang [43], pp. 343–360. https://doi.org/10.1007/978-3-540-71677-8_23
43. Okamoto, T., Wang, X. (eds.): PKC 2007, LNCS, vol. 4450. Springer, Heidelberg (Apr 2007)
44. Reistad, T., Toft, T.: Linear, constant-rounds bit-decomposition. In: Lee, D., Hong, S. (eds.) ICISC 09. LNCS, vol. 5984, pp. 245–257. Springer, Heidelberg (Dec 2010)
45. Reistad, T.I.: Multiparty comparison-an improved multiparty protocol for comparison of secret-shared values. In: SECRYPT. pp. 325–330 (2009)
46. Reistad, T.I., Toft, T.: Secret sharing comparison by transformation and rotation. In: Information Theoretic Security: Second International Conference, ICITS 2007, Madrid, Spain, May 25-29, 2007, Revised Selected Papers 2. pp. 169–180. Springer (2009)
47. Shamir, A.: How to share a secret. Communications of the Association for Computing Machinery **22**(11), 612–613 (Nov 1979)
48. Yao, A.C.C.: Protocols for secure computations (extended abstract). In: 23rd FOCS. pp. 160–164. IEEE Computer Society Press (Nov 1982). https://doi.org/10.1109/SFCS.1982.38

A Information-theoretic MPC protocols

We briefly describe and summarise the performance of [2]. This honest majority MPC protocol presents a minor improvement over [32]:

Traditionally, information-theoretically secure LSSS-based MPC protocols for an arbitrary number of nodes were based on the so-called BGW protocol [11], which was later improved by [30]. For each multiplication gate, this protocol requires each node to send a share to all other nodes. When the number of nodes is not too small, the protocol of [25] is faster by instead requiring a node to send their shares only to a single "king" node, which subsequently sends shares back. This protocol thus ends up making each node send approximately 2 elements online for each multiplication gate, and an additional 4 elements offline to construct the necessary secret shared random numbers (when using superinvertible matrices [34]). By cleverly reusing this randomness, the MPC protocol [32] brings the preprocessing cost down from 4 elements to only 2.

As is evident in this paper, efficient comparison protocols do not merely consist of addition and multiplication gates, but also rely heavily on reveal and public multiplication gates. The MPC protocol [2] also uses "king" nodes for these gates, again resulting in 2 elements per node sent online per gate, and reuses the trick of [32] of t-wise independent sharings so that public multiplications require only 1 element per node preprocessed offline.

B One-round comparison protocol

The comparison protocol of [38, §II.B] (see also [20]) can be summarised as follows: given a secret shared value $[y]$, the nodes jointly compute and reveal some value $yc_0 - c_1$ for large but bounded secret integers c_0, c_1 with $|c_1| \ll |c_0| \neq 0$, and then use the sign of this public value to locally compute a sharing of the Boolean $y < 0$.

- They state their protocol has round complexity $O(\log n)$, but after moving (without affecting efficiency) most of its components to preprocessing their protocol can be done with only 1 online public multiplication.

This public value $yc_0 - c_1$ leaks some information on y and thus one needs to estimate its statistical privacy precisely. However, if one requires perfect correctness, then in order to determine whether y is strictly positive or strictly negative the value $|c_1|$ needs to be less than the minimum value c_0 can attain. If furthermore the value zero is not ruled for y, then $|c_1|$ must be less than half this minimum. In conclusion, the effect of c_1 on masking y is negligible, and the masking of y is essentially multiplicative. But it is well-known that such masks are not very effective because the number of divisors of a random number are relatively low, hence the additional bandwidth required would be inefficient.

C Comparison protocol with logarithmic complexity

As far as we know, there is only one protocol in the literature which computes a bitwise comparison with complexity logarithmic in the number of bits ℓ:

- [40, $\texttt{gt}_{(\ell),\log}$]: The basic idea of this protocol is to recursively check whether the $\ell/2$ most significant bits of two secret shared values of bit-length ℓ are different, until only the highest pair of bits in which they differ, is left; comparing those then yields the result. Each of these $\lceil \log \ell \rceil$ invocations has 5 communication rounds, costing a shifted bit decomposition[9] plus reveal, a bit equality protocol (which uses 2 reveals, a multiplication and an invertible secret random value whose powers have been precomputed to the power ℓ), and a multiplication.

Although the online complexity of this protocol is only logarithmic in the number of bits, its preprocessing cost and high round complexity make it inefficient in most practical settings. Moreover the privacy of their protocol is only statistical, thus using extra bandwidth. Only when the bit-length of the values to be compared is very high (at least 128) and online bandwidth is more of a concern than latency and offline bandwidth, and perfect security is required, does this protocol seem to become potentially competitive, after making the following improvements:

- Use a more efficient bit equality protocol $\mathcal{P}_{\texttt{BitEq}}$, see [2].

[9] An advantage here of the non-ITS setting is that fewer random bits have to be generated to preserve statistical privacy.

– The last multiplication in the protocol is only computed locally (in the honest majority setting), saving an additional round in each iteration.

The number of rounds in the protocol is then almost halved from $5\lceil \log \ell \rceil + 1$ to $3\lceil \log \ell \rceil + 1$, and as a consequence the online bandwidth is also reduced by 40% when using [2].