Sub-linear, Secure Comparison with Two Non-colluding Parties

Tomas Toft

Dept. of CS, Aarhus University, Denmark
ttoft@cs.au.dk

Abstract. The classic problem in the field of secure computation is Yao's millionaires' problem; we consider two new protocols solving a variation of this: a number of parties, $P_1, \ldots, P_n$, securely hold two ℓ-bit values, x and y – e.g. x and y could be encrypted or secret shared. They wish to obtain a bit stating whether x is greater than y using only secure arithmetic; this should be done without revealing any information, even the output should remain secret. The present setting is special in the sense that it is assumed that two specific parties, referred to as Alice and Bob, are non-colluding. Though this assumption is not satisfied in general, it clearly is for the main example of this work: two-party computation based on Paillier encryption.

The first solution requires $O(\log(\ell)(\kappa + \text{loglog}(\ell)))$ secure arithmetic operations in $O(\log(\ell))$ rounds, where κ is a correctness parameter. The second solution requires only a constant number of rounds, but increases complexity to $O(\sqrt{\ell}(\kappa + \log(\ell)))$ arithmetic operations.

For the motivating setting, each arithmetic operation requires a constant number of Paillier encryptions to be exchanged between Alice and Bob. This implies that both solutions require only a *sub-linear* number of invocations (in the bit-length, ℓ) of the cryptographic primitives. This does not imply sub-linear communication, though, as the size of each encryption transmitted is more than ℓ bits.

Keywords: Secure computation, Yao's Millionaires' problem.

1 Introduction

The concept of secure multiparty computation was introduced by Yao with the now classic millionaires' problem, [Yao82]: two millionaires meet in the street and wish to determine who is richer without revealing their net worths. Since then, this problem – along with variations of it – has received much interest in the research community. Not only is it the original problem, it is also a quite fundamental primitive for secure computation. Auctions are an immediate application, e.g. as considered in [NPS99]. Indeed, they are the motivating examples of many of the papers on secure comparison mentioned below. The problem – and extensions such as determining the minimal of multiple inputs – also plays an important role elsewhere, e.g. in secure data mining such as [LP00] or [JW05] and secure optimization such as [Tof09].

D. Catalano et al. (Eds.): PKC 2011, LNCS 6571, pp. 174–191, 2011.

Secure arithmetic modulo some integer M can be seen as secure integer computation when no overflows modulo M occur. Augmenting such primitives with a protocol for secure comparison provides a setting which allows general integer computation. This approach is often used, when considering applications as those mentioned above. Thus, improving comparison – as the present work – implies improvement of a whole range of applications. These are the first such protocols which invoke the cryptographic primitives $o(\ell)$ times, where ℓ is the bit-length of the inputs.

1.1 Related Work

As the problem of secure comparison has received so much interest, there is a large body of related work, which solves the problem in various settings and with focus on different properties. Some solutions focus on theoretical efficiency – low round complexity (constant or even single round), as well as low communication or computation complexity. Others consider practice, attempting to obtain a fast, real-world solution by balancing the different resources. There are also many variations of the problem. For example: where do the inputs come from; are they known to any of the parties or are they the result of previous computation? Should the result be public, should simple actions be performed based on the outcome, or should the result be private to allow it to be used in arbitrary further computation?

There are essentially two different settings. The two-party case contains Yao circuits, [Yao86], but also includes [Fis01, BK06, DGK07, GSV07, LL07]. The latter are generally not limited to known inputs and public output. The multi-party setting can be split into the cryptographic and the information theoretic (i.t.) settings. [DFK$^+$06, NO07] consider constant rounds solutions in the i.t. setting, while [ST06] considers computationally efficient solutions based on Paillier encryption [Pai99], i.e. the cryptographic setting. Hybrids such as [BCD$^+$09], are also possible: their protocol is based on secret sharing, but the focus is entirely on practice, and primitives guaranteeing only computational security are applied to improve efficiency.

Though radically different, the comparison protocol of Feige et al., [FKN94], is in a sense the closest related work as it uses quadratic residues at its core. However, the basic protocol only allows comparison of integers between zero and two. Though it can be generalized, comparing anything but *very* small values appears highly impractical.

1.2 Contribution

This paper considers a novel approach at secure comparison. The setting consists of n players, $P_1, \ldots, P_n$ that jointly hold two secret, ℓ-bit inputs $x, y \in \mathbb{Z}_M$ to be compared – ordering is obtained by viewing the inputs as ℓ-bit integers. Two of the parties, Alice and Bob, are guaranteed by assumption to be non-colluding. I.e. the adversary cannot corrupt both simultaneously. The desired outcome is for

the parties to hold (in a secure fashion) a bit stating whether or not $x > y$. The motivating setting is the two-party setting; for threshold schemes the protocols are limited to the case when the threshold is 1. When there are few parties (say three or four), this is completely acceptable, however, the protocols are not generally applicable.

This clearly solves the "standard" millionaires' problem: the millionaires simply provide their inputs (e.g. encrypt and send them to the relevant parties). Once the result has been determined, it can be revealed (e.g. decrypted). However, having such a comparison protocol realizes the secure integer computation setting from above – the outcome of a comparison could equally well be used in subsequent computation.

Two protocols are presented here, both based on arbitrary secure arithmetic modulo M, where M is either an RSA-modulus (e.g. Paillier encryption) or a prime (e.g. secret sharing over $\mathbb{F}_M$ where M is prime). The first solution requires $O(\log(\ell)(\kappa + \text{loglog}(\ell)))$ arithmetic operations in $O(\log(\ell))$ rounds, where κ is a correctness parameter. The second solution is constant-rounds, but increases complexity to $O(\sqrt{\ell}(\kappa + \log(\ell)))$. With the exception of [FKN94], all previous solutions known to the author utilize access to the binary representation of the inputs (or the binary representation of random values related to the inputs). Hence, this is the first solution that requires less than a linear number of invocations of the primitives in the bit-length of the inputs.

Security against passive adversaries follows almost entirely from the security of the underlying arithmetic. However, this is not the case with respect to active adversaries. The problem is that the protocol specifies Alice and Bob to provide inputs of bounded size, which cannot be verified efficiently using only arithmetic. Fortunately such a primitive exists in many settings including Paillier based ones as well as based on Shamir sharing over a prime field, $\mathbb{F}_M$.

1.3 An Overview of This Paper

Section 2 introduces the setting and primitives in the form of an ideal functionality; Sect. 3 then explains how this can be realized. Secure equality testing is then presented in Sect. 4; this is needed for both solutions. The two contributions are then presented in Sect. 5 and Sect. 6. Finally, two variations are noted in Sect. 7, before the concluding remarks of Sect. 8.

2 Primitives and Notation

The basic setting will consist of an ideal functionality providing the underlying primitives, the main one being secure $\mathbb{Z}_M$ arithmetic, i.e. it is an arithmetic black-box (ABB), [DN03]. This approach ensures that it is possible to ignore the details of the underlying primitives; focus is on the required properties rather than on specific solutions.

2.1 The Arithmetic Black-Box

The arithmetic black-box allows n parties, $P_1, \ldots, P_n$, to securely store and retrieve elements of a ring $\mathbb{Z}_M$. Here, M will be either a prime or an RSA-modulus, i.e. the product of two odd primes.

The secure storage (input/output) can be thought of as secret sharing, and we borrow that notation, writing stored values in square brackets, $[x]$. However, as noted other solutions are also possible, and the notation could equally well represent encryption: secure storage could be obtained by having one or more parties store encryptions under some public key. Input then means "encrypt and send to these parties," while output means "decrypt and broadcast." Naturally the decryption key itself must not be held by anyone also holding the encryptions, however, it could be secret shared or held by a different party. The ABB can also provide an output to only a *single* party. This is always achievable, e.g. by additively masking the output with a random value provided by the receiving party. For specific primitives, other solutions may be preferable, though.

In addition to secure storage, the ideal functionality allows arithmetic computation on the stored values. Such computation is written using infix notation in the "plaintext" space, e.g.

$$[x \cdot y + z] \leftarrow [x] \cdot [y] + [z] \,.$$

The actual operations to be performed depend on the details of the realization. Presently it does not matter whether an operation is a protocol invocation (e.g. the multiplication protocol of Ben-Or et al. [BGW88]) or simply local computation by one or more parties (e.g. multiplying two Paillier encryptions provides an encryption of the sum of the two plaintexts).

Complexity of a protocol based on the ABB is found by simply counting the number of operations (input/output or arithmetic[1]) performed. It is assumed that the ABB allows operations to be performed concurrently (representing e.g. executing multiple multiplication protocols in parallel), thus, round complexity will be the number of sequential operations performed.

2.2 Required Extensions of the ABB

The ABB-setting considers n parties, $P_1, \ldots, P_n$. The present work requires two of these – denoted Alice and Bob – to be mutually incorruptible, i.e. at least one of them will remain honest. We do not specify the corruption sets further. Indeed, the remaining $n-2$ parties are ignored. The ideas are best explained by focusing on Alice, Bob, and the ABB. Including the "helper parties"[2] would complicate the explanation unnecessarily.

[1] Similarly to other work, we focus on communication complexity and assume that the underlying primitives are additively homomorphic. Thus, only multiplications are counted.

[2] Denoting the remaining $n-2$ parties as "helpers" is not completely fair. They may only be assisting during the present comparison protocols, but could have an equal stake in the overall computation.

As noted above, the present protocols do not provide active security based only on the arithmetic black-box. Even when actively secure arithmetic is given, malicious parties can still deviate. Two extensions to the ABB are therefore needed in order to eliminate these issues. First, given an input, it must be verifiable (using only constant work) that it is less than some public bound. This allows the protocol to specify that an input must come from a small, specified range. Second, it must be verifiable that two held values are indeed equal.

Finally, to improve readability we introduce a bit of syntactic sugar, writing

$$[b] \,?\, [x] : [y]$$

to denote conditional selection between two secret values, $[x]$ and $[y]$, based on an secret bit, $[b]$. This can be achieved using arithmetic only, and the expression is simply shorthand for

$$[b]\,([x] - [y]) + [y]$$

which clearly equals either x or y depending on $b \in \{0, 1\}$.

3 Realizing the Arithmetic Black-Box

The arithmetic black-box can be realized in different settings and be based on various primitives. However, as the present setting requires two of the parties to be mutually incorruptible, our prime example is two-party computation based on Paillier encryption [Pai99]. The realization is sketched in Sect. 3.1. It is stressed that realizations with more than two parties are also relevant. Two possibilities are presented in Sect. 3.2; both provide security against active adversaries. The realizations of the extensions needed to provide active security below are provided in Sect. 3.3.

3.1 Passively Secure, Two-Party Paillier-Based Arithmetic

Paillier's encryption scheme [Pai99] is a semantically secure, additively homomorphic, public-key cryptosystem over $\mathbb{Z}_N$, where $N = pq$ is an RSA-modulus. In the simple, two-party setting, Bob holds a copy of Alice's public key. To provide an input means to hand Bob a fresh encryption of it. Output is realized by sending the relevant encryption to Alice (rerandomizing it first); she then decrypts and returns the plaintext to Bob. At this point both parties know the value in question.

Regarding the realization of the secure arithmetic, the homomorphic property allows Bob to compute linear combinations of encryptions that he holds. However, he is unable to obtain encryptions of products without the help of Alice. For this he uses the standard protocol seen in Fig. 1, where subtraction simply means "invert and add." (In the ciphertext domain this means "multiply by the multiplicative inverse.") Correctness follows from

$$\begin{aligned} xy &= (xy + r_y x + r_x y + r_x r_y) - r_y x - r_x y - r_x r_y \\ &= (x + r_x)(y + r_y) - r_y x - r_x y - r_x r_y \\ &= x'y' - r_y x - r_x y - r_x r_y. \end{aligned}$$

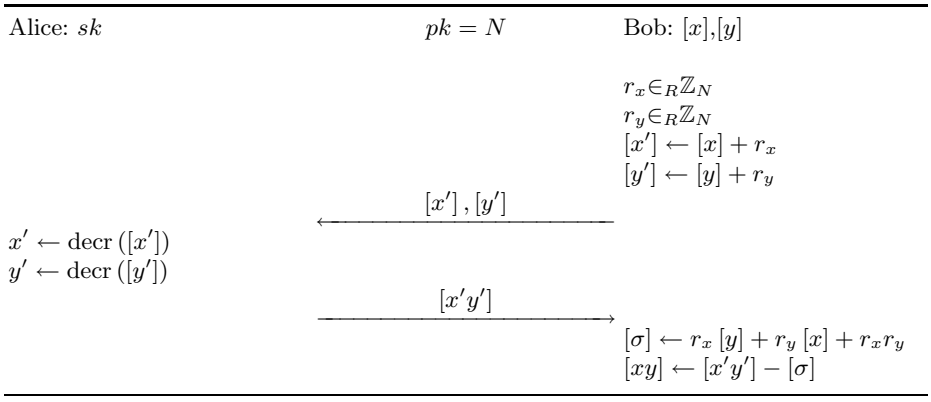

Fig. 1. Multiplication of two encryptions

On the intuitive level, this clearly realizes the arithmetic black-box. For input/output Alice's view consists only of the encryptions of outputs she receives, while Bob only sees the outputs and encryptions under Alice's key. For the multiplication protocol, Alice receives two encryptions of masked (i.e. uniformly random) values, while Bob simply receives two additional encryptions. Neither provides any information.

Formally simulating these views is simple. Sketching the ideas, Alice must be handed fresh encryptions, either of the output to be received (which the simulator knows) or a random value in the case of the multiplication protocol. Bob on the other hand expects to see encryptions of the values in question. The simulator cannot generate these, but from Bob's point of view they are indistinguishable from arbitrary encryptions, as the scheme is semantically secure. Hence any fresh encryptions under Alice's (simulated) key will do.

3.2 The Multiparty Case

The arithmetic black-box can also be realized in a multiparty setting. Note that both of the options mentioned are secure against active adversaries, but may be simplified, if it is assumed that the adversary is honest-but-curious only, i.e. if the corrupt parties follow the protocol as specified.

The first candidate consists of using Shamir's secret sharing scheme over the prime field $\mathbb{F}_M$ along with the protocols of Ben-Or et al. [Sha79, BGW88]. This results in a threshold scheme, which is not applicable in general. However, if the threshold is 1, then at most one party will be corrupt. In that case *any two parties* can take the role of Alice and Bob.

A multiparty solution based on Paillier encryption is also possible using the techniques of Cramer et al. [CDN01]. In that setting, all parties hold all ciphertexts, while the key is secret shared among them. Again, a threshold solution allowing at most one corrupt party is a possible setting. Note that though not

presented so, it is possible to use the protocols of [CDN01] with a threshold $t \geq n/2$; this provides an alternative, two-party setting, which is more suited when considering active adversaries.

3.3 Active Security

Both solutions of the previous section realize the arithmetic black-box in the presence of active adversaries. It remains to provide the required extensions, i.e. to describe how to verify that two values are equal, and how to demonstrate that an input is of bounded size.

Equality of two values is easily verified using only arithmetic by outputting their difference. This is of course not secure in general, however, in the present setting, one of the secret values depends only on inputs from one party (who of course knows the actual value). It can therefore be viewed as coming directly from that party. For an honest party, the output is always 0, while for a corrupt party, the adversary will know the output in advance. Hence no new information can be obtained. It is noted that for specific primitives, more direct solutions may be more efficient.

The second problem is to verify that an input provided by some party is indeed of bounded size. This can be done by taking a detour over the integers, as any non-negative integer can be written as the sum of the squares of four integers. For a Paillier based setting, Schoenmakers and Tuyls [ST06] note that this can be achieved using integer commitments [Bou00, Lip03, DJ02]. For the setting based on Shamir sharing, a similar proof that a value is of bounded size can be obtained using linear integer secret sharing, [Tho09]. Sketching the latter solution, the key idea is to first verify that a value (shared over the integers) is of the desired size (i.e. provide the four integers). This sharing is then converted to a Shamir sharing over $\mathbb{F}_M$.

4 Secure Equality Testing

An additional primitive is needed before the comparison protocols can be presented: securely determining a secret bit stating if two values, $[x]$ and $[y]$, are equal. The protocol is a variation of the secure equality testing of secret shared values, [NO07, Tof07]; the latter work notes that the ideas generalize to the case of multiparty computation based on Paillier encryption. Note that in specific settings, specialized variations of the present protocols will most likely be preferable to the general solution presented here.

The main idea is seen in Fig. 2. To test equality of two values, it suffices to test if their difference, $[d]$, is 0. If this is the case, then adding a random square, $[r]^2$, results in a value with Jacobi symbol $j_{d+r^2} = \left(\frac{d+r^2}{M}\right) = 1$. If $[d] \neq 0$, however, then the Jacobi symbol is -1 with probability roughly $1/2$,

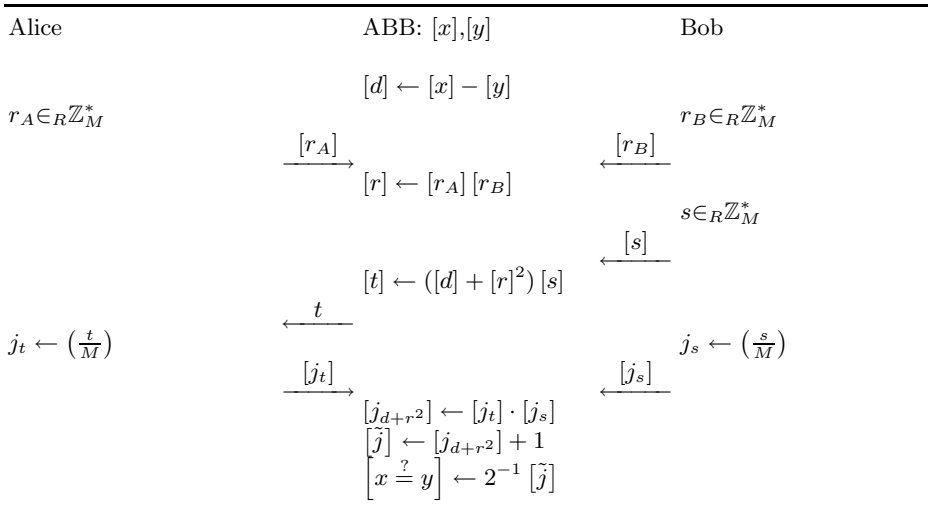

Fig. 2. Testing equality (with error probability $\approx 1/2$)

[Per52, Tof07]. The Jacobi symbol can therefore be used as a test of equality[3]. As the Jacobi symbol of a product is the product of the Jacobi symbols of the factors, and the multiplicative inverse of ± 1 is itself, $[j_{d+r^2}]$ is correctly computed when $[j_s]$ and $[j_t]$ are the Jacobi symbols of $[s]$ and $[t]$ respectively. The final computation simply maps -1 to 0 while preserving a 1.

A false positive – an incorrect 1 – occurs with a probability of roughly 1/2. However, if the protocol is repeated κ times on the same input, the probability that *all* executions provide false positives is negligible. Determining if all invocations have returned 1 can be done by computing the κ-ary fan-in AND of the test results. This can be done in $O(1)$ rounds using $O(\kappa)$ arithmetic operations as described in [DFK+06]. The basic idea is to compute the sum of the bits plus one, $[\sigma]$, and use this as input to the known, $(\kappa+1)$-degree polynomial mapping $1, \ldots, \kappa$ to 0 and $\kappa + 1$ to 1. The powers of $[\sigma]$ – $[\sigma], \left[\sigma^2\right], \ldots, \left[\sigma^{\kappa+1}\right]$ – can be computed in constant rounds using $O(\kappa)$ multiplications by utilizing (a simple variation of) the unbounded fan-in multiplication protocol of Bar-Ilan and Beaver, [BB89].

Correctness. Correctness follows by the intuition above: each test always returns an encryption of 1 when $x = y$, and the logical AND of these bits is therefore 1 as well. When $x \neq y$, then except with negligible probability in κ, at least one of the tests will successfully determine this, i.e. return 0. In this case the logical AND is also 0.

[3] As M is either the product of large primes or a large prime itself, $j_{d+r^2} = 0$ occurs with negligible probability. In some settings, the issue can be eliminated using arithmetic, e.g. by considering $\left(\frac{d^2+r^2}{M}\right)$ when $M \equiv 3 \bmod 4$ is prime. There the additive inverse of a quadratic residue will never be a quadratic residue itself.

Passive security. In Fig. 2, Alice receives t. This is the only potential information leak, as the arithmetic black-box is secure by definition. As M only has large prime factors, $d + r^2$ is in $\mathbb{Z}_M^*$, except with negligible probability. This implies that $[s]$ completely blinds it, hence Alice learns nothing. More generally, Alice and Bob are mutually incorruptible and s and t are known only to them, so no adversary will learn both s and t.

Active security. To ensure security against active adversaries, the actions of the parties must be verified. The only issues are the inputs, as security of the arithmetic is immediate. I.e. the parties must verify that the values provided by Alice and Bob are as specified. There are two issues:

1. Demonstrate that an input is in $\mathbb{Z}_M^*$.
2. Verify that an input is the Jacobi symbol of a stored, inputter-known value.

Note that this suffices; $[r]$ is uniformly random as either Alice or Bob is honest.

For Alice or Bob to demonstrate that an input is invertible, it suffices to provide an additional uniformly random invertible value, and have the product of the two revealed. All parties can then verify that that the product is invertible, which implies the same of both factors. This reveals no other information, as the second element masks the real value.

The second issue is slightly more difficult. For Alice to demonstrate that $[j_t]$ is the Jacobi symbol of $[t]$ she provides uniformly random pairs, $([j_i], [t_i])$ for $1 \leq i \leq k$, such that $j_i = \left(\frac{t_i}{M}\right)$. Bob then flips k coins, $b_1, \ldots, b_k$; the ABB is then asked to compute and reveal either the pair $([j_i] \cdot [j_t], [t_i] \cdot [t])$ if b_i is 1 or $([j_i], [t_i])$ otherwise. All parties then verify that the Jacobi symbol of each revealed pair matches the element. To cheat, Alice would have to guess *all* of Bob's coin flips – this can only occur with probability negligible in k. Note that no information is revealed when Alice is honest: the parties merely see random pairs, $([j_i], [t_i])$, or maskings of $[t]$ and its Jacobi symbol. For Bob to demonstrate the same, the roles are simply reversed.

It is often possible to do better than the general solution. When, for example, M is prime, elements with known Jacobi symbol can be efficiently constructed using only arithmetic:

$$[x] \leftarrow [x']^2 \cdot \left(2^{-1}([j_x] + 1) \,?\, 1 : \omega\right),$$

where ω is a fixed element with Jacobi symbol -1, [NO07, Tof07].

A similar calculation is possible when M is the product of two primes both congruent to 3 modulo 4. In this case -1 is a non-residue with Jacobi symbol 1, hence any element $[x]$ can be written as

$$([b] \,?\, 1 : -1)\,[x']^2 \cdot \left(2^{-1}([j_x] + 1) \,?\, 1 : \omega\right)$$

where $[b]$ is a bit. (If Alice cannot compute b, i.e. distinguish quadratic residue from non-residues, she can instead provide a uniformly random value with the same Jacobi symbol. The parties can then reveal the product of this and the actual value to Bob, who verifies that its Jacobi symbol is 1).

Complexity. For passive security, when M is prime, or when M is the product of two primes both congruent to 3 modulo 4, executing κ copies of Fig. 2 in parallel and performing the κ-ary fan-in AND requires $O(\kappa)$ ABB-operations executed in $O(1)$ rounds. In other settings – i.e. when it is not possible to efficiently verify a Jacobi symbol – arithmetic complexity increases by a factor of k. For simplicity we can view this as being incorporated into κ – say $k = \sqrt{\kappa}$ – though strictly speaking it should be treated separately.

5 The log-Rounds Protocol

Based on the arithmetic black-box presented in Sect. 2 – including extensions such as the equality test – the log-rounds protocol can now be explained. In a sense, this paper follows the same overall strategy as many of the previous solutions: transform the problem to a comparison of "known" values and determine their most significant differing bit-position; this provides the result. What differs are the means to achieve this goal.

On the intuitive level, the approach can be viewed as a binary search, though strictly speaking, this is not the case. The full protocol is seen in Fig. 3 and is explained during the argument of correctness. Assume for simplicity that ℓ is a power of two. This can always be ensured by viewing x and y as numbers of larger (but less than double) size. Further, assume that the modulus of the ABB is much larger than than the input size, $M \gg 2^{\ell+\kappa'}$ for security parameter κ'.

Correctness. Correctness of the protocol is quite simple, once the intuition is understood. Therefore, rather than presenting the protocol from beginning to end, we present the ideas. This requires starting at both the end *and* the beginning at the same time, and working our way towards the middle. It explains *why* a given computation is performed by showing *how* it helps solve the original problem.

Initially the arithmetic black-box is used to compute the value $[z]$. As $x \geq y \Leftrightarrow z \geq 2^\ell$, we find that if $[z_\ell]$ really is an encryption of the ℓ'th bit of z, then the result is correct. This is the case if $\bar{z} = z \bmod 2^\ell$: $z - (z \bmod 2^\ell)$ sets all the ℓ least significant bits of the $(\ell+1)$-bit z to 0 leaving only the top bit. The multiplication by $2^{-\ell}$ (modulo M) simply shifts this down to the desired position.

To perform the modulo reduction of $[z]$, Bob provides a $(2^{\ell+\kappa'})$-bit mask, $[r]$, which is added to $[z]$. The outcome, $[c]$, is then revealed to Alice. As it was assumed that $M \gg 2^{\kappa'+\ell}$, the computation of c can be viewed as occurring over the integers. Therefore

$$z \equiv c - r \mod 2^\ell.$$

Both c and r are easily reduced modulo 2^ℓ. Alice knows the former and Bob the latter, so they may simply supply these values, $[\bar{c}]$ and $[\bar{r}]$. Subtracting the latter from the former provides the desired result, however, this subtraction must occur modulo 2^ℓ, not modulo M.

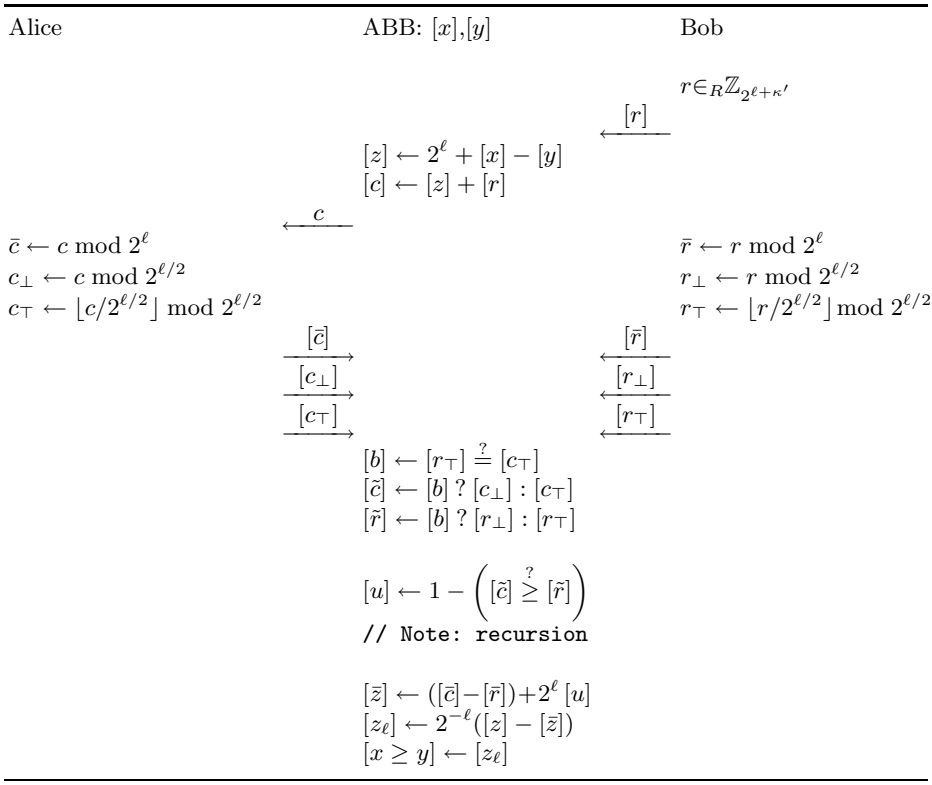

Fig. 3. The $\log(\ell)$-rounds comparison protocol

However, by considering two cases, the desired operation can be simulated with $\mathbb{Z}_M$ arithmetic:

1. If $\bar{c} \geq \bar{r}$, then subtraction modulo M provides the correct result.
2. If $\bar{c} < \bar{r}$, then subtracting $\bar{r}$ results in an underflow modulo M. Adding 2^ℓ at this point provides the correct result.

Unfortunately the parties do not know which case they are in – nor should they learn it – and therefore do not know if they should instruct the ABB to add 2^ℓ.

This leaves us with the problem of comparing $[\bar{c}]$ and $[\bar{r}]$, and while it seems as if we are back at the initial problem, this is not the case, as Alice and Bob each know one of these values. Thus, they can be decomposed into the top and bottom halves – the $\ell/2$ least significant and most significant bits – by having the parties input them. These are denoted $[c_\top]$, $[c_\bot]$, $[r_\top]$, and $[r_\bot]$.

Again there are two cases to consider, and again the parties do not know which is the case:

1. If $c_\top = r_\top$, i.e. if the most significant half of the bits are equal, then it suffices to compare the values representing the least significant bits.

2. If $c_\top \neq r_\top$, i.e. if the top halves differ, then the least significant bits can be ignored, and it suffices to compare $c_\top$ and $r_\top$.

Using the equality test, the parties can determine a secret bit stating which is the case. Based on this, they obliviously choose between the top and bottom halves of $[\bar{c}]$ and $[\bar{r}]$ – i.e. $[r_\top]$, $[c_\top]$ and $[r_\bot]$, $[c_\bot]$ – storing the results as $[\tilde{r}]$ and $[\tilde{c}]$.

The concluding computation consists of determining which of $[\tilde{r}]$ and $[\tilde{c}]$ contains the larger value. This *is* the original problem, however, the bit-length of the numbers is now only $\ell/2$. If $\ell/2 > 1$, then the protocol calls itself recursively. Otherwise – i.e. if the values to be compared are single-bit – it is straightforward to compute an encryption of $\tilde{r} > \tilde{c}$ using only secure arithmetic:

$$(\tilde{c} \oplus \tilde{r})\tilde{r} = \tilde{r} - \tilde{c}\tilde{r}.$$

To see that this is the correct result, note that it is 1 exactly when the bits differ and $\tilde{r}$ is set.

Security. As the arithmetic black-box is secure by definition, an adversary can only obtain information or affect the computation through the input/output. Leakage can only occur through the value, c, received by a corrupt Alice. However, this is statistically indistinguishable (in κ') from a uniformly random $(\ell + \kappa')$-bit value, as $[z]$ is masked by $[r]$ provided by the honest Bob.

Similarly, an adversary can only affect the computation through incorrect inputs. Hence, if it is verified that $[\bar{c}]$, $[c_\top]$, $[c_\bot]$, $[\bar{r}]$, $[r_\top]$, and $[r_\bot]$ are indeed the correct "sub-strings" of $[c]$ and $[r]$ (and that $[r]$ is indeed $\ell + \kappa'$ bits long), then no adversarial behavior is possible.

This verification can be performed by having Alice and Bob provide $[c_I]$ and $[r_I]$, the (ignored) top κ' bits of $[c]$ and $[r]$, as well. Initially it is verified that $[c_\top]$, $[c_\bot]$, $[r_\top]$, and $[r_\bot]$ are all $\ell/2$ bits, and that $[c_I]$ and $[r_I]$ are κ' bits. Then, the parties check that $[\bar{c}] = 2^{\ell/2}\,[c_\top] + [c_\bot]$, $[\bar{r}] = 2^{\ell/2}\,[r_\top] + [r_\bot]$, $[c] = 2^{\ell}\,[c_I] + [\bar{c}]$, and $[r] = 2^{\ell}\,[r_I] + [\bar{r}]$. At this point it has been ensured that the decomposition of $[c]$ and $[r]$ are correct. Note that there is no need to explicitly verify that $[\bar{c}]$ and $[\bar{r}]$ are ℓ bits, nor that $[r]$ is $\ell + \kappa'$ bits. This has already been verified implicitly.

Complexity. For a reduction of the problem to one of half size – Fig. 3 except for the recursive invocation – one invocation of the equality test and a constant amount of input/output and arithmetic is required. Overall this means $O(\kappa)$ arithmetic and input/output operations are needed, where κ is the correctness parameter for the equality test. These can be performed in $O(1)$ rounds. Each iteration reduces the problem to one of half size, thus only $\log(\ell)$ steps are required until the bit-length reaches 1, at which point the remaining work is constant. This implies that the overall complexity is $O(\kappa \log(\ell))$ operations in $O(\log(\ell))$ rounds.

Note that to ensure correctness of the full protocol, *all* invocations of the equality test must succeed. As the number of tests depends on ℓ, so must κ. Adding $\operatorname{loglog}(\ell)$ provides the desired error probability, as $(1 - 2^{-\kappa})^{\log(\ell)} \approx 1 - \log(\ell)2^{-\kappa}$

(since $2^{-\kappa}$ is negligible)[4]. This implies $O(\log(\ell)(\kappa+\operatorname{loglog}(\ell)))$ operations overall, to ensure correctness except with probability negligible in κ.

Theorem 1. *Given two mutually incorruptible parties, Alice and Bob, and two ℓ-bit values, $[x]$ and $[y]$, stored within an arithmetic black-box providing secure arithmetic in $\mathbb{Z}_M$ (where $M > 2^{\ell+\kappa'+\operatorname{loglog}(\ell)}$ is prime or an RSA-modulus and κ' is a security parameter), the parties may obtain a bit, $[b]$, such that b is 1 iff $x \geq y$ (except with probability negligible in the correctness parameter κ) using $O(\log(\ell)(\kappa + \operatorname{loglog}(\ell)))$ ABB-operations in $O(\log(\ell))$ rounds.*

6 The Constant-Rounds Protocol

Decreasing the round complexity to constant without considering the individual bits is achieved by combining the ideas of the previous section with earlier approaches. Rather than going over *all details*, we present the overall protocol in Fig. 4, and only reference existing sub-protocols in the analysis. This avoids muddling the presentation with details of existing protocols.

Correctness. The protocol starts and ends exactly as the log-rounds solution, hence if $[u]$ correctly states if an underflow occurs in the subtraction, then the correct result is determined. In difference to above, $[\bar{c}]$ and $[\bar{r}]$ are split into $\sqrt{\ell}$ blocks rather than two (the bit-length can be padded to ensure that it is a square). This can be viewed as writing the numbers in $2^{\sqrt{\ell}}$-ary notation. As in the previous section, it suffices to only compare the most significant differing block. There are more of them, but the goal remains the same: the parties must obliviously find and select the block in question using only the ABB.

To do this, an equality test is performed for each block: $[b_i]$ states if the i'th block of the numbers are equal. $\left[\tilde{b}_i\right]$ is the logical AND of the top blocks, i.e. from the most significant one down to the i'th. It is 1 exactly when all these blocks are equal. Thus, the most significant differing block will be the first one containing a 0 (starting with the most significant one). The $\left[\tilde{b}_i\right]$ of the less significant blocks will of course also be 0, as there is a more significant, differing position. The value $[b'_i] = \left[\tilde{b}_{i+1}\right] - \left[\tilde{b}_i\right]$ states if the i'th block is the desired one. For the most significant, differing block it will be $1 = 1 - 0$, while the rest have $b'_i = 0$, either from $1 - 1$ or $0 - 0$. Thus, the $[b'_i]$ that is set can be viewed as a pointer to the correct block position, i, and $\left[r^{(i)}\right]$ and $\left[c^{(i)}\right]$ are selected by the sums[5].

Concluding, the $\sqrt{\ell}$-bit $[\tilde{c}]$ and $[\tilde{r}]$ must be compared. This time, we cannot proceed recursively, as this would not result in a constant-rounds protocol. Instead, the parties transform the problem to one where Alice holds one input and

[4] A similar increase is needed for the security parameter, κ'. This does not change complexity though.

[5] If the inputs are equal this is not true, but as we get $\tilde{c} = \tilde{r} = 0$ which provides the same result, it is acceptable.

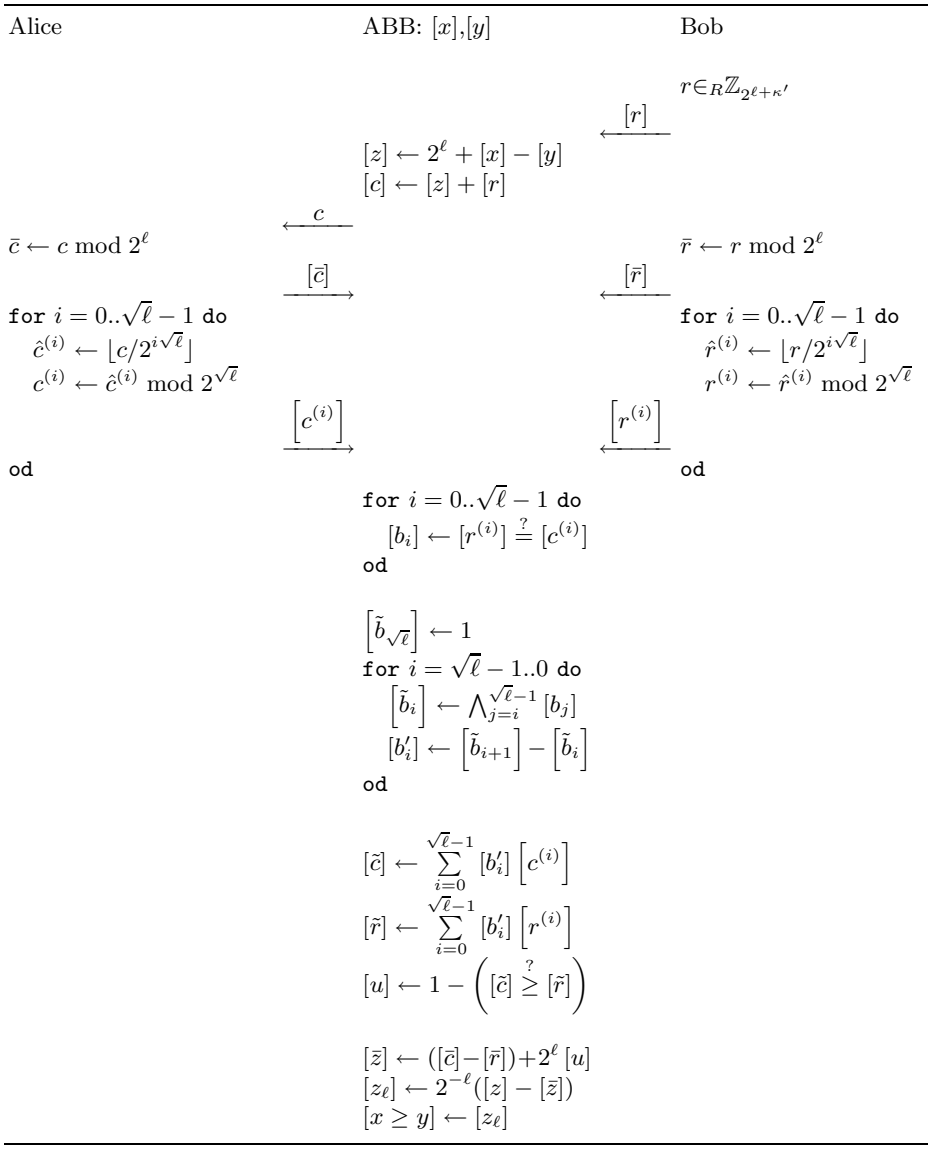

Fig. 4. The constant-rounds comparison protocol

Bob the other – the initial step of the present protocols. Then, these numbers are bit-decomposed – Alice and Bob provide the binary representation of the numbers – and the problem is brute-forced, e.g. using [DFK+06]. As there are only $\sqrt{\ell}$ bits, linear work is acceptable at this point.

Security. Privacy of the protocol follows by the exact same argument as above: leakage can only happen through c, which is statistically indistinguishable from

a uniformly random $(\ell + \kappa')$-bit value. Further, as above, an active adversary can only affect the protocol through inputs. To avoid malicious behavior, it must merely be verified that $[r]$ is indeed of the proper bit-length; that $[r]$ and $[c]$ are decomposed properly into the $\left[c^{(i)}\right]$ and $\left[r^{(i)}\right]$; and that $[\bar{c}]$ and $[\bar{r}]$ are correct. In addition to this, it must be verified that the bit-decomposition needed in the comparison of $[\tilde{c}]$ and $[\tilde{r}]$ is correct. All this can be performed analogously to the constructions of the previous section – the difference is that a $2^{\sqrt{\ell}}$-ary or binary representation is considered rather than a $2^{\ell/2}$-ary one.

Complexity. Initially, Bob provides the mask, $[r]$, after which Alice obtains c, the masking of $[z]$. They decompose these to their $2^{\sqrt{\ell}}$-ary representations and provide these as inputs to the ABB. Complexity of this is clearly $O(\sqrt{\ell})$ input/output operations. Moreover, only three rounds are needed as all elements of the decompositions may be input concurrently.

Regarding the work performed by the ABB, clearly the computation of $[z]$ and $[c]$ is constant. Following this, $\sqrt{\ell}$ equality tests are performed; each of these requires $O(\kappa)$ operations implying $O(\sqrt{\ell} \cdot \kappa)$ operations overall. Round complexity remains constant, though: no test depends on the output of another, so they may be executed in parallel. The next step is to compute the "pointer," $[b'_i]$. The most expensive part of this is the $\sqrt{\ell}$ $\sqrt{\ell}$-ary logical AND's. The naive solution requires $\Omega(\ell)$ operations which is too expensive. However, the AND-gates share the same inputs – overall it is simply a prefix-AND of the b_i. Thus, the computation can be performed with linear work in a constant number of rounds by applying the techniques of [DFK+06]. Concluding the computation are the selection of $[\tilde{r}]$ and $[\tilde{c}]$ each requiring $O(\sqrt{\ell})$ arithmetic operations. This is followed by the brute-force (linear in $\sqrt{\ell}$) comparison and the final, constant-size computation.

The dominating term of all of this is the equality tests, due to their factor of κ. Thus, the overall complexity is $O(\sqrt{\ell}(\kappa+\log(\ell)))$, where the $\log(\ell)$-term is needed to ensure that the error probability remains negligible in κ when performing all $\sqrt{\ell}$ tests.

Theorem 2. *Given two mutually incorruptible parties, Alice and Bob, and two ℓ-bit values, $[x]$ and $[y]$, stored within an arithmetic black-box providing secure arithmetic in $\mathbb{Z}_M$ (where $M > 2^{\ell+\kappa'}$ is prime or an RSA-modulus and κ' is a security parameter), the parties may obtain a bit, $[b]$, such that b is 1 iff $x \geq y$ (except with probability negligible in the correctness parameter κ) using $O(\sqrt{\ell}(\kappa+\log(\ell)))$ ABB-operations in $O(1)$ rounds.*

7 Variations

Handling arbitrary inputs of $\mathbb{Z}_M$. Protocols allowing arbitrary inputs in $\mathbb{Z}_M$ are also possible. This involves comparing both inputs, $[x]$ and $[y]$, as well as $[x - y]$ to $\lfloor M/2 \rfloor$, [NO07]. The answer can be determined using a small arithmetic circuit. Comparison with $\lfloor M/2 \rfloor$ is equivalent to doubling and extracting the

least significant bit, which translates to a comparison of values held by Alice and Bob. At this point the present ideas may be applied.

Improved complexity in the constant-rounds case. Complexity of the constant-rounds protocol can be improved slightly. The idea is to split the numbers into $\sqrt{\ell/\kappa}$ blocks of size $\sqrt{\ell\kappa}$. Determining and selecting the relevant block requires only $O(\sqrt{\ell\kappa})$ arithmetic operations, and while the new comparison problem grows to size $\sqrt{\ell\kappa}$, it is still acceptable to brute-force this.

A second approach allows complexity to be reduced further at the cost of extra rounds. Splitting the numbers into $\sqrt[3]{\ell}$ blocks results in a new problem of size $(\sqrt[3]{\ell})^2$, which may be solved with $O(\sqrt[3]{\ell} \cdot \kappa)$ work. This amount is also needed to reduce the size of the problem, thus, it is also the overall complexity. The idea generalizes to $O(c)$-round $O(\sqrt[c]{\ell} \cdot \kappa)$ solutions for any c.

Hybrid protocols for practice. Due to the blowup of κ in the complexity, theoretically worse solutions may be preferable to the log-rounds protocol for small inputs. This suggests that a hybrid approach will be best in practice: Initially the log-rounds solution can be applied repeatedly to reduce the size of the problem, but at some point (when the problem is small enough) another solution will be better. At this point, that solution may be applied instead of continuing with the recursive approach.

8 Concluding Remarks

The protocols presented demonstrate that in order to perform secure comparison, the explicit binary representation does not have to be considered. Rather, by testing equality of "sub-strings" of the full problem (where this test occurs on elements of the ring or field), the size of the problem can be reduced without having to consider each bit-position individually. This implies an improved theoretic complexity over all previous solution (regarding the number of cryptographic invoked).

A sub-linear comparison protocol for the general multiparty setting is left as an open problem. One obvious possibility would be to generate encryptions or sharings of uniformly random values of bounded size, say k-bit, which are unknown to all. It is not clear how to do this without also generating the binary representation as well, though.

The author would like to thank Ivan Damgård, Martin Geisler, and the anonymous referees for their many remarks and suggestions.

References

[BB89] Bar-Ilan, J., Beaver, D.: Non-cryptographic fault-tolerant computing in a constant number of rounds of interaction. In: Rudnicki, P. (ed.) Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, pp. 201–209. ACM Press, New York (1989)

[BCD+09] 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., Toft, T.: Secure multiparty computation goes live. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 325–343. Springer, Heidelberg (2009)

[BGW88] Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for noncryptographic fault-tolerant distributed computations. In: 20th Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM Press, New York (1988)

[BK06] Blake, I.F., Kolesnikov, V.: Conditional encrypted mapping and comparing encrypted numbers. In: Di Crescenzo, G., Rubin, A. (eds.) FC 2006. LNCS, vol. 4107, pp. 206–220. Springer, Heidelberg (2006)

[Bou00] Boudot, F.: Efficient proofs that a committed number lies in an interval. In: Preneel, B. (ed.) EUROCRYPT 2000. LNCS, vol. 1807, pp. 431–444. Springer, Heidelberg (2000)

[CDN01] Cramer, R., Damgård, I.B., Nielsen, J.B.: Multiparty computation from threshold homomorphic encryption. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 280–300. Springer, Heidelberg (2001)

[DFK+06] 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 (2006)

[DGK07] Damgård, I., Geisler, M., Krøigaard, M.: Efficient and Secure Comparison for On-Line Auctions. In: Pieprzyk, J., Ghodosi, H., Dawson, E. (eds.) ACISP 2007. LNCS, vol. 4586, pp. 416–430. Springer, Heidelberg (2007)

[DJ02] Damgård, I., Jurik, M.: Client/Server tradeoffs for online elections. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 125–140. Springer, Heidelberg (2002)

[DN03] 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 (2003)

[Fis01] Fischlin, M.: A cost-effective pay-per-multiplication comparison method for millionaires. In: Naccache, D. (ed.) CT-RSA 2001. LNCS, vol. 2020, pp. 457–471. Springer, Heidelberg (2001)

[FKN94] Feige, U., Killian, J., Naor, M.: A minimal model for secure computation (extended abstract). In: STOC 1994: Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, pp. 554–563. ACM Press, New York (1994)

[GSV07] Garay, J.A., Schoenmakers, B., Villegas, J.: Practical and secure solutions for integer comparison. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 330–342. Springer, Heidelberg (2007)

[JW05] Jagannathan, G., Wright, R.: Privacy-preserving distributed k-means clustering over arbitrarily partitioned data. In: KDD, pp. 593–599 (2005)

[Lip03] Lipmaa, H.: On diophantine complexity and statistical zero-knowledge arguments. In: Laih, C.-S. (ed.) ASIACRYPT 2003. LNCS, vol. 2894, pp. 398–415. Springer, Heidelberg (2003)

[LL07] Laur, S., Lipmaa, H.: A new protocol for conditional disclosure of secrets and its applications. In: Katz, J., Yung, M. (eds.) ACNS 2007. LNCS, vol. 4521, pp. 207–225. Springer, Heidelberg (2007)

[LP00] Lindell, Y., Pinkas, B.: Privacy preserving data mining. Journal of Cryptology, 36–54 (2000)

[NO07] Nishide, T., Ohta, K.: Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 343–360. Springer, Heidelberg (2007)

[NPS99] Naor, M., Pinkas, B., Sumner, R.: Privacy preserving auctions and mechanism design. In: Proceedings of the 1st ACM Conference on Electronic Commerce, pp. 129–139. ACM Press, New York (1999)

[Pai99] Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999)

[Per52] Perron, O.: Bemerkungen über die Verteilung der quadratischen Reste. Mathematische Zeitschrift 56(2), 122–130 (1952)

[Sha79] Shamir, A.: How to share a secret. Communications of the ACM 22(11), 612–613 (1979)

[ST06] Schoenmakers, B., Tuyls, P.: Efficient binary conversion for paillier encrypted values. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 522–537. Springer, Heidelberg (2006)

[Tho09] Thorbek, R.: Linear Integer Secret Sharing. PhD thesis, Aarhus University (2009)

[Tof07] Toft, T.: Primitives and Applications for Multi-party Computation. PhD thesis, Aarhus University (March 2007), `http://www.daimi.au.dk/~ttoft/publications/dissertation.pdf`

[Tof09] Toft, T.: Solving linear programs using multiparty computation. In: Dingledine, R., Golle, P. (eds.) FC 2009. LNCS, vol. 5628, pp. 90–107. Springer, Heidelberg (2009)

[Yao82] Yao, A.: Protocols for secure computations (extended abstract). In: 23th Annual Symposium on Foundations of Computer Science (FOCS 1982), pp. 160–164. IEEE Computer Society Press, Los Alamitos (1982)

[Yao86] Yao, A.: How to generate and exchange secrets (extended abstract). In: 27th Annual Symposium on Foundations of Computer Science, pp. 162–167. IEEE Computer Society Press, Los Alamitos (1986)