Efficient Secure Comparison Protocols

Geoffroy Couteau

KIT, Karlsruhe, Germany

Abstract. A secure comparison protocol allows players to evaluate the greater-than predicate on hidden values; it addresses a problem that belongs to the field of multiparty computation, in which players wish to jointly and privately evaluate a function on secret inputs. Introduced by Yao under the name *millionaires' problem*, secure comparison has received a great deal of attention. It has proven to be a fundamental building block in a variety of multiparty computation protocols. However, due to their inherent non-arithmetic structure, existing constructions often remain a major efficiency bottleneck in multiparty computation.
In this work, we design new two-party protocols for the greater-than functionality, improving over the state of the art. We prove the security of our protocols in the UC model with respect to passive corruption, assuming only oblivious transfers. Our constructions can readily be used in a variety of protocols in which secure comparisons constitute the main efficiency bottleneck. We construct our protocols in the preprocessing model, with an extremely efficient information-theoretically secure online phase. We use oblivious transfer extensions to get rid of all but a constant amount of expensive computations. Toward our goal of secure comparison, we also design protocols for testing equality between private inputs, which improve similarly over the state of the art. The latter contribution is of independent interest.

Keywords. Two-party computation, Secure comparison, Equality test, Oblivious transfer.

1 Introduction

Multiparty Computation (MPC) addresses the challenge of performing computation over sensitive data without compromising its privacy. In the past decades, several general-purpose solutions to this problem have been designed, starting with the seminal works of Yao [Yao86] and Golwasser, Micali, and Widgerson [GMW87]. Among the large variety of problems related to MPC that have been considered, the *secure comparison* problem, in which the players wish to find out whether $x \geq y$ for given x, y without disclosing them, is probably the one that received the most attention. Indeed, in addition to being the first MPC problem ever considered (introduced in [Yao86] under the name of millionaire's problem), it has proven to be a fundamental primitive in a considerable number of important applications of multiparty computation. Examples include auctions, signal processing, database queries, machine learning and statistical analysis, biometric authentication, combinatorial problems, or computation on rational numbers. Secure comparison is at the heart of any task involving sorting data, finding a minimum value, solving any optimization problem, or even in tasks as basic as evaluating the predicate of a while loop, among countless other examples.

Two-party and multiparty computation seem now at the edge of becoming practical, with increasing evidence that they are no more beyond the reach of the computational power of today's computers. However, secure comparisons appear to be a major bottleneck in secure algorithms. Various implementations of secure algorithms unanimously lead to the conclusion that secure comparison is the most computationally involved primitive, being up to two orders of magnitude slower than, e.g., secure multiplication. Hence, we believe that improving secure comparison protocols is one of the major roads toward making multiparty computation truly practical.

In this work, we consider secure comparison on inputs privately held by each player, so that the output is shared between the players. This setting captures every possible applications, as simple folklore methods allow to reduce the problem of comparing shared or encrypted inputs to the problem of comparing inputs held by the parties. We construct new two-party protocols for securely comparing private inputs which compare very favorably to state-of-the-art solutions. In particular, our protocols are well suited for large scale secure computation protocols using secure comparison as a basic routine. We prove security in the universal composability framework of Canetti [Can01], which ensures that security is preserved under general composition. As this is the model used in most practical applications, we focus on the passive adversarial model, in which players are assumed to follow the specifications of the protocol. We leave as open the interesting question of extending our protocols to handle malicious adversaries, while preserving (as much as possible) their efficiency.

1.1 State of the Art for Secure Comparison

To avoid unnecessary details in the presentation, we assume some basic knowledge on classical cryptographic primitives, such as garbled circuits, oblivious transfers and cryptosystems. Preliminaries on oblivious transfers are given Section 2. In the following, we let ℓ denote an input length, and κ denote a security parameter.

From Garbled Circuits. The first category regroups protocols following the garbled circuit approach of Yao [Yao86]. The protocol of [KS08], which was later improved in [KSS09] and [ZRE15], is, to our knowledge, the most communication-efficient secure comparison protocol. The protocol of [KSS09] proceeds by letting the first player garble a circuit containing ℓ comparison gates, which amounts to ℓ AND gates with the free-xor trick. In a setting where several instances of the comparison protocol will be invoked, oblivious transfer extensions [IKNP03] can be used to execute an arbitrary number of secure comparisons using a only constant number of public key operations and cheap symmetric operations for each invocation of the secure protocol, making it computationally very efficient.

From Homomorphic Encryption. Solutions to the millionaire problem from homomorphic-encryption originated in [BK04]. The most efficient method in this category, to our knowledge, is [DGK07], which uses an ad hoc cryptosystem. This protocol was corrected in [DGK09], and improved in [Veu12]. The protocol communicates 4ℓ ciphertexts (in the version that outputs shares of the result) and is often regarded as one of the most computationally efficient. The more recent construction of [GHJR15] relies on the flexibility of lattice based cryptosystems to design a secure comparison protocol. Using a degree-8 somewhat homomorphic encryption scheme and ciphertext packing techniques, the (amortized) bit complexity of their protocol is $\tilde{O}(\ell + \kappa)$. Although asymptotically efficient, this method is expected to remain less efficient than alternative methods using simpler primitives for any realistic parameters.

From the Arithmetic Black Box Model. The third category consists of protocols built on top of an arithmetic black box [CDN01] (ABB), which is an ideal reactive functionality for performing securely basic operations (such as additions and multiplications) over secret values loaded in the ABB. The ABB itself can be implemented from various primitives, such as oblivious transfer [Rab81, EGL82] or additively homomorphic encryption (most articles advocate the Paillier scheme [Pai99]). Protocols in this category vary greatly in structure. Most protocols [DFK$^+$06, GSV07, NO07, Cd10a] involve $\tilde{O}(\ell)$ private multiplications, each typically requiring $O(1)$ operations over a field of size $O(\ell + \kappa)$, resulting in an overall $\tilde{O}(\ell(\ell + \kappa))$ bit complexity. The protocols of Toft [Tof11], and Toft and Lipmaa [LT13], use only a sublinear (in ℓ) number of invocations to the cryptographic primitive; however, the total bit complexity remains superlinear in ℓ. For large values of ℓ ($\kappa^2/\ell = o(1)$), the protocol of [YY12] enjoys an optimal $O(\ell)$ communication complexity; however, the constants involved are quite large: it reduces to $84\lambda + 96$ bit oblivious transfer and 6 ℓ-bit secure multiplications for a $1/2^\lambda$ error probability, and becomes competitive with e.g. [KSS09] only for inputs of at least 500 bits (assuming a $1/2^{40}$ error probability).

1.2 Our Contribution

In this work, we construct new protocols for secure comparison, which improve over the best state-of-the-art protocols. Our protocols are secure in the universal composability framework, assuming only an oblivious transfer. Using oblivious transfer extensions allows to confine all public-key operations to a one-time setup phase. The online phase of our protocols enjoys information theoretic security, and is optimal regarding both communication and computation: $O(\ell)$ bits are communicated, and $O(\ell)$ binary operations are performed, with small constants. Regarding overall complexity, our protocols match the best existing constructions in terms of asymptotic efficiency (and have in particular an optimal $O(\ell)$ complexity for large values of ℓ, see Table 1), and outperform the most efficient for practical parameters (which is the protocol of [KSS09], according to our estimations) by about 40%. This comes at the cost of a somewhat high $O(\log \ell)$ interactivity (2 to 12 online rounds in practice, or 2 to 7 using a cheap tradeoff between communication and interactivity, see Subsection 4.5).

Additional contributions resulting from the methods we use include a new efficient equality test (outputting shares of a bit equal to 1 iff the input strings are equal) which improves over the state of art by up to 80% regarding communication and computation (see Subsection 3.5), and a simple method which reduces by 25% the communication of the Naor-Pinkas oblivious transfer protocol [NP01], when the size of the transmitted strings is lower than $\kappa/2$ (see Section 3). These contributions are of independent interest: equality test protocols enjoy independent applications as building blocks in various multiparty computation protocols. Examples include, but are not limited to, protocols for switching between homomorphic encryption schemes [CPP16], secure linear algebra [CKP07], secure pattern matching [HT14], and secure evaluation of linear programs [Tof09]. In addition, we provide as a supplementary material a variant of our equality test protocol in a batch settings (where many equality tests are performed "by blocks"), which uses additively homomorphic encryption to further improve the communication of our equality test protocol by up to 50%. We postpone its description to the supplementary material.

1.3 Universal Composability

We prove the security of our protocols in the universal composability framework (UC), and assume that the reader has some familiarity with it. The UC framework has been introduced by Canetti in [Can01]. It defines protocols by the mean of systems of interactive Turing machines. The expected behavior of the protocol is captured by an ideal functionality F. This functionality is a simple interactive machine, connected to a set of dummy parties, some of whom might be corrupted by an ideal adversary $\mathscr{Sim}$ through perfectly secure authenticated channels. In the real execution of a protocol π, probabilistic polynomial time players, some of whom might be corrupted by a real adversary Adv, interact with each other through some channels. The *environment* refers to an interactive machine Z that oversees the execution of the protocol in one of the two worlds (the ideal world with the functionality F, or the real world with the protocol π). We refer to [Can01], for the definitions of the real world ensembles $\mathsf{EXEC}_{\pi,\mathsf{Adv},Z}$ and the ideal world ensemble $\mathsf{EXEC}_{F,\mathscr{Sim},Z}$. A protocol UC securely implements a functionality F if for any adversary Adv, there is a simulator $\mathscr{Sim}$ so that the real world ensemble and the ideal world ensemble are indistinguishable for any environment Z.

1.4 Our Method

The high level intuition of our approach is an observation that was already made in previous works [Tof11, LT13]: to compare two strings, it suffices to divide them in equal length blocks, and compare the first block on which they differ. Therefore, a protocol for (obliviously) finding this block can be used to reduce the secure comparison problem on large strings to the secure comparison problem on smaller strings. One can then recursively apply this size-reduction protocol, until the strings to be compared are small enough, and compute the final result using a second protocol tailored to secure comparison on small strings. However, this intuition was typically implemented in previous work using heavy public-key primitives, such as homomorphic encryption. In this work, we show how this strategy can be implemented using exclusively oblivious transfers on small strings.

To implement the size-reduction protocol, we rely on a protocol to obliviously determine whether two strings are equal. Therefore, a first step toward realizing a secure comparison protocol is to design a protocol for testing equality between two strings, which outputs shares (modulo 2) of a bit which is 1 if and only if the strings are equal. Keeping this approach in mind, we start by designing an equality test protocol which is based solely on oblivious transfer. Recall that in an oblivious transfer protocol, one party (the sender) inputs a pair (s_0, s_1), while the other party (the receiver) inputs a bit b; the receiver receives s_b as output and learns nothing about s_{1-b}, while the sender learns nothing about b. Our protocol relies on a classical observation: two strings are equal if and only if their Hamming distance is zero. More specifically, our protocols proceed as follows:

Equality Test. Consider two inputs (x, y), of length ℓ. We denote $(x_i, y_i)_{i\leq\ell}$ their bits. The parties execute ℓ parallel oblivious transfers over $\mathbb{Z}_{\ell+1}$, where the first player input pairs $(a_i+x_i \bmod \ell+1, a_i+1-x_i \bmod \ell+1)$ (a_i is a random mask over $\mathbb{Z}_{\ell+1}$), and the second party input his secret bits y_i; let b_i be

his output ($b_i = a_i + x_i \oplus y_i \bmod \ell+1$, where $\oplus$ is the exclusive or). Observe that $x' \leftarrow \sum_i a_i \bmod \ell+1$ and $y' \leftarrow \sum_i b_i \bmod \ell+1$ are equal if and only if the Hamming distance between x and y is 0, if and only if $x = y$. Note that (x', y') are of length $\log(\ell+1)$.

The players repeatedly invoke the above method, starting from (x', y'), to shrink the input size while preserving equality, until they end up with string of length at most (say) 3 bits (it takes about $O(\log^* \ell)$ invocations of the protocol, where the first invocation dominates the communication cost). The players then perform a straightforward equality test on these small strings, using oblivious transfers to evaluate an explicit exponential-size formula for equality checking on the small entries.

The core feature of this compression method is that it can be almost entirely preprocessed: by executing the compression protocol on random inputs (r, s) in a preprocessing phase (and storing the masks generated), the players can reconstruct the output of the protocol on input (x, y) simply by exchanging $x \oplus r$ and $s \oplus y$ in the online phase. Therefore, the communication of the entire equality test protocol can be made as low as a few dozens to a few hundreds of bits in the online phase. Furthermore, in the preprocessing phase, the protocol involves only oblivious transfers on very small entries (each entry has size at most $\log \ell$ bits), for which particularly efficient constructions exist [KK13].

Secure Comparison. We now describe our solution to the secure comparison problem. This protocol has a structure somewhat comparable to the previous one, but is more involved. The parties break their inputs (x, y) in $\sqrt{\ell}$ blocks of length $\sqrt{\ell}$ each. In the first part of the protocol, the parties will construct $\sqrt{\ell}$ shares of bits, which are all equal to 0 except for the ith bit, where i is the index of the first block on which x differs from y. This step relies on parallel invocations to the equality test functionality, and on oblivious transfers. Then, using these bit-shares and oblivious transfers, the players compute shares of the first block on which x differs from y.

At this point, we cannot directly repeat the above method recursively, as this method takes inputs *known to the parties*, while the output values are only shared between the parties. However, under a condition on the size of the group on which the shares are computed, we prove a lemma which shows that the parties can *non-interactively* reduce the problem of securely comparing shared value to the problem of securely comparing known values, using only local computations on their shares. From that point, the parties can apply the compression protocol again (for $O(\log \ell)$ rounds), until they obtain very small values, and use (similarly as before) a straightforward protocol based on an explicit exponential-size formula for comparison. Alternatively, to reduce the interactivity, the compression protocol can be executed a fixed (constant) number of times, before applying, e.g., a garbled-circuit-based protocol on the reduced-size inputs.

This protocol involves $O(\sqrt{\ell})$ equality tests and oblivious transfers on small strings, which can both be efficiently preprocessed. This leads to a secure comparison protocol that communicates about a thousand bits in the online phase, for 64-bit inputs.

1.5 Comparison with Existing Works

We provide Table 1 a detailed comparison between the state of the art, our logarithmic-round protocol SC_1, and its constant-round variant SC_2. We evaluate efficiency in an amortized setting and ignore one-time setup costs. We considered two methods based on garbled circuit, the protocol of [KSS09] and the same protocol enhanced with the method of [AIKW13] to optimize the online communication. We also considered the solution based on the DGK cryptosystem [DGK07,DGK09,Veu12], the protocol of [LT13], and the probabilistically correct protocol of [YY12]. Note that [LT13, YY12] are described with respect to an arithmetic black box, hence their cost depends on how the ABB is implemented. For [LT13], which require an ABB over large order fields, we considered a Paillier based instantiation, as advocated by the authors. For [YY12], which involves (mainly) an ABB over $\mathbb{F}_2$, we considered the same optimizations than in our protocols, implementing the ABB with oblivious transfers on bits.

As illustrated in Table 1, our protocols improve over existing protocols regarding both communication and computation. This comes at the cost of a somewhat high $O(\log \ell)$ interactivity (or $O(c \cdot \log^* \kappa)$ in the constant-round setting). In particular, for large values of ℓ (and for any value of ℓ in the online phase), our protocols enjoy an optimal $O(\ell)$ communication and computation complexity. The hidden constants are small, making our protocols more efficient than the state of the art for any practical

Table 1: Amortized Costs of State of The Art Secure Comparison

Protocol	[KSS09]	[DGK07, DGK09, Veu12][1]	[LT13][1]	[KSS09]+ [AIKW13][1]	[YY12]
			Preprocessing Phase		
Communication	$O(\kappa\ell)$	–	$O(n\kappa \log \ell)$	$O(n\ell)$	$O(\frac{\kappa^2}{\log \kappa} + \ell)$
Computation	$O(\kappa\ell)$	$O(\ell(\kappa+\ell) \cdot C_n)$	$O(n\kappa \log \ell \cdot C_n)$	$O(n\ell \cdot C_n)$	$O(\frac{\kappa^2}{\log \kappa} + \ell)$
Rounds	1	–	$O(1)$	1	$O(1)$
Assumption	OT	–	ABB	RSA	ABB
			Online Phase		
Communication	$O(\kappa\ell)$	$O(n\ell)$	$O(n \log \ell)$	$O(\ell + n)$	$O(\kappa + \ell)$
Computation	$O(\kappa\ell)$	$O(\ell \log \ell \cdot C_n)$	$O(n \log \ell \cdot C_n)$	$O(\kappa\ell + n \cdot C_n)$	$O(\kappa + \ell)$
Rounds	2	2	$O(\log \ell)$	2	$O(\log \kappa)$
Assumption	OWF	DGK	ABB	RSA	None
Protocol	SC_1		SC_2 (c is some fixed constant)		
			Preprocessing Phase		
Communication	$O(\frac{\kappa\ell}{\log \kappa})$ if $\ell/\kappa^2 = o(1)$ $O(\ell)$ else		$O(\frac{\kappa\ell}{\log \kappa})$ if $\ell^{1-1/c}/\kappa^2 = o(1)$ $O(\ell)$ else		
Computation	$O(\frac{\kappa\ell}{\log \kappa})$ if $\ell/\kappa^2 = o(1)$ $O(\ell)$ else		$O(\frac{\kappa\ell}{\log \kappa})$ if $\ell^{1-1/c}/\kappa^2 = o(1)$ $O(\ell)$ else		
Rounds	$O(\log \ell)$		$O(c \log^* \kappa)$		
Assumption	OT		OT		
			Online Phase		
Communication	$O(\ell)$		$O(\ell)$		
Computation	$O(\ell)$		$O(\ell)$		
Rounds	$O(\log \ell)$		$O(c \log^* \kappa)$		
Assumption	Information theoretic		OWF		

[1] $n > \ell + \kappa$ is the length of an RSA modulus. C_n denotes the cost of a modular multiplication modulo n. Note that [AIKW13] can also be instantiated from the DDH or the LWE assumption.

parameter. For values of ℓ between 4 and 128, the protocol of [KSS09] (which enjoys tiny constants) outperforms the other existing protocols regarding communication, computation, and interactivity. We therefore focus on this protocol as a basis for comparison with our protocols in our concrete efficiency estimations, Subsection 4.5.

Equality Tests. The state of the art given Table 1 remains essentially the same for equality tests. Indeed, all the papers listed in the table (at the exception of [DGK07], but including the present paper) do also construct equality tests protocols, with the same (asymptotic) complexity and from the same assumptions. Note that we consider only equality tests whose output is shared between the players (as this is necessary for our secure comparison protocol); if the players get to learn the output in the clear (this is known as the socialist millionaires problem), more efficient solutions exist, but they is no simple way of designing equality tests with shared outputs from these solutions.

1.6 Applications

Secure comparisons have found a tremendous number of applications in cryptography; we provide thereafter a non-exhaustive list of applications for which our protocols lead to increased efficiency. We note that in applications for which implementations have been described, the communication of secure comparisons was generally pointed out as the main efficiency bottleneck.

– *Obliviously sorting data* [Goo10,Goo14] has proven useful in contexts such as private auctions [NIIO14], oblivious RAM [Gol87], or private set intersection [HEK12], but it remains to date quite slow

(in [HICT14], sorting over a million 32-bit words takes between 5 and 20 minutes). All existing methods crucially rely on secure comparisons and require at least $O(m \log m)$ secure comparisons in $O(\log m)$ rounds to sort lists of size m.
- *Biometric authentication*, while solving issues related to the use of passwords, raises concerns regarding the privacy of individuals, and received a lot of attention from the cryptographic community. Protocols for tasks such as secure face recognition [SSW10] require finding the minimum value in a database, which reduces to $O(m)$ secure comparisons in $O(\log m)$ rounds.
- Secure protocols for *machine learning* employ secure comparisons as a basic routine for tasks such as classification [BPTG14], generating private recommendations [EVTL12], spam classification [WFNL15], multimedia analysis [CC15], clinical decisions [RPV+14], evaluation of disease risk [ARL+13], or image feature extraction [LLY+16].
- Secure algorithms for *combinatorial problems*, such as finding the flow of maximum capacity in a weighted graph, or searching for the shortest path between two nodes, have been investigated in several works, e.g. [Lau15], and have applications in protocols such as private fingerprint matching [BS15], privacy-preserving GPS guidance, or privacy-preserving determination of topological features in social networks [ACM+13]. They typically involve a very large number of secure comparisons (e.g. n^2 comparisons for Dijkstra's shortest path algorithm on an n-node graph [ACM+13]).
- Other applications that heavily rely on comparisons include computing on non integer values [ABZS13], various types of secure auctions [DGK07], range queries over encrypted databases [SJB14], or algorithms for optimization problems [Tof09, Cd10b].

1.7 Organization

In Section 2, we recall definitions and classical results on oblivious transfers, as well as on oblivious transfer extensions. Section 3 introduces our new equality test protocol. Section 4 focus on the construction of secure comparison protocols. Eventually, we describes a variant of our equality test in the supplementary material, Section A.

1.8 Notations

Given a finite set S, the notation $x \leftarrow_R S$ means that x is picked uniformly at random from S. For an integer n, $\mathbb{Z}_n$ denotes the set of integers modulo n. Throughout this paper, $+$ will always denote addition over the integers, and not modular additions. We use bold letters to denote vectors. For a vector $\boldsymbol{x}$, we denote by $\boldsymbol{x}[i]$ its i'th coordinate; we identify k-bit-strings to vectors of $\mathbb{Z}_2^k$ (but do not use bold notations for them). We denote by $\boldsymbol{x} * \boldsymbol{y}$ the Hadamard product $(\boldsymbol{x}[i] \cdot \boldsymbol{y}[i])_i$ between $\boldsymbol{x}$ and $\boldsymbol{y}$. Let $\oplus$ denote the xor operation (when applied on bit-strings, it denotes the bitwise xor). For integers (x, y), $[x = y]$, $[x < y]$, and $[x \leq y]$ denote a bit which is 1 if the equality/inequality holds, and 0 otherwise. The notation $(x \bmod k)$, between parenthesis, indicates that $x \bmod t$ is seen as an integer between 0 and $t-1$, not as an element of $\mathbb{Z}_t$. For an integer k, let $\langle \cdot \rangle_k$ denote the randomized function that, on input x, returns two uniformly random shares of x over $\mathbb{Z}_k$ (i.e., a random pair $(a, b) \in \mathbb{Z}_k$ such that $a + b = x \bmod k$). We extend this notation to vectors in a natural way: for an integer vector $\boldsymbol{x}$, $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow_R \langle \boldsymbol{x} \rangle_k$ denote the two vectors obtained by applying $\langle \cdot \rangle_k$ to the coordinates of $\boldsymbol{x}$. Finally, for an integer x, we denote by $|x|$ the bit-size of x.

2 Oblivious Transfer

Oblivious transfers (OT) were introduced in [Rab81]. An oblivious transfer is a two-party protocol between a sender and a receiver, where the sender obliviously transfers one of two string to the receiver, according to the selection bit of the latter. The ideal functionality for k oblivious transfers on l-bit strings is specified as follows:

$$\mathscr{F}_{\mathsf{OT}}^{k,l} : ((\boldsymbol{s}_0, \boldsymbol{s}_1), x) \mapsto \left(\perp, \left(\boldsymbol{s}_{x[i]}[i]\right)_{i \leq k}\right)$$

where $(\boldsymbol{s}_0, \boldsymbol{s}_1) \in (\mathbb{F}_2^l)^k \times (\mathbb{F}_2^l)^k$ is the input of the sender, and $x \in \mathbb{F}_2^k$ is the input of the receiver. In a *random oblivious transfer* (ROT), the input of the sender is picked at random:

$$\mathscr{F}_{\mathsf{ROT}}^{k,l} : (\perp, x) \mapsto \left((\boldsymbol{s}_0, \boldsymbol{s}_1), \left(\boldsymbol{s}_{x[i]}[i]\right)_{i \leq k} \right)$$

The primitive can be extended naturally to k-out-of-n oblivious transfers; we let $\binom{n}{k}$-OT_ℓ^t denotes t invocations of a k-out-of-n OT on strings of length ℓ. Oblivious transfer is a fundamental primitive in MPC as it implies general multiparty computation [Kil88] and can be made very efficient.

2.1 Oblivious Transfer Extension

Although oblivious transfer requires public-key cryptographic primitives, which can be expensive, *oblivious transfer extension* allows to execute an arbitrary number of oblivious transfers, using only cheap, symmetric operations, and a small number of base OTs. OT extensions were introduced in [Bea96]. The first truly practical OT extension protocol was introduced in [IKNP03], assuming the random oracle model.[1] We briefly recall the intuition of the OT extension protocol of [IKNP03]. A $\binom{2}{1}$-OT_t^κ can be directly obtained from a $\binom{2}{1}$-$\mathsf{OT}_\kappa^\kappa$: the sender associates two κ-bit keys to each pair of messages and obliviously transfer one key of each pair to the receiver. Then, the receiver stretches two t-bit strings from the two keys of each pair, using a pseudo-random generator, and sends the xor of each of these strings and the corresponding message to the receiver. The $\binom{2}{1}$-OT_ℓ^t itself can be implemented with a single call to a $\binom{2}{1}$-OT_t^κ functionality, in which the receiver plays the role of the sender (and reciprocally). The total communication of the reduction from $\binom{2}{1}$-OT_ℓ^t to $\binom{2}{1}$-$\mathsf{OT}_\kappa^\kappa$ is $2t\ell + 2t\kappa$ bits. Regarding the computational complexity, once the base OTs have been performed, each OT essentially consists in three evaluations of a hash function. An optimization to the protocol of [IKNP03] was proposed in [ALSZ13] (and discovered independently in [KK13]). It reduces the communication of the OT extension protocol from $2t\ell + 2t\kappa$ bits to $2t\ell + t\kappa$ bits, and allows to perform the base OTs without an a-priori bound on the number of OTs to be performed later (the OTs can be continuously extended).

Oblivious Transfer of Short Strings. An optimized OT extension protocol for short strings was introduced in [KK13], where the authors describe a reduction of $\binom{2}{1}$-OT_ℓ^t to $\binom{2}{1}$-$\mathsf{OT}_\kappa^\kappa$ with $t(2\kappa/\log n + n \cdot \ell)$ bits of communication, n being a parameter that can be chosen arbitrarily so as to minimize this cost. Intuitively, this is done by reducing $\log n$ invocations of $\binom{2}{1}$-OT to one invocation of $\binom{n}{1}$-OT; the result is then obtained by combining this reduction with a new $\binom{n}{1}$-OT extension protocol introduced in [KK13]. In our concrete efficiency estimations, we will heavily rely on this result as our equality test protocol involves only OTs on very short strings.

Correlated and Random Oblivious Transfers. The authors of [ALSZ13] described several OT extension protocols, tailored to OTs on inputs satisfying some particular conditions. In particular, the communication of the OT extension protocol can be reduced from $2t\ell + t\kappa$ bits to $t\ell + t\kappa$ bits when the inputs to each OT are *correlated*, i.e. when each input pair is of the form $(r, f(r))$ for a uniformly random r and a function f known by the sender (which can be different for each OT). For random oblivious transfer extension, the bit-communication can be further reduced to $t\kappa$. We note that the optimizations of [KK13] and [ALSZ13] can be combined: $\log n$ correlated $\binom{2}{1}$-OT can be reduced to one correlated $\binom{n}{1}$-OT (defined by input pairs of the form $(r, f_1(r), \cdots f_{n-1}(r))$ for a random r and functions $f_1 \cdots f_{n-1}$ known by the sender). This gives a correlated short-string oblivious transfer extension protocol which transmits $t(2\kappa/\log n + (n-1) \cdot \ell)$ bits.

3 Equality Test

3.1 Equality Test Protocol

In this section, we design an equality-test (ET) protocol to securely compute shares over $\mathbb{Z}_2$ of the equality predicate.

[1] The random oracle model can be avoided by assuming that the hash function is a correlation-robust function, see [KK13], Appendix A.2.

Ideal Functionalities. The ideal functionality for our ET protocol is represented Figure 1. Following the common standard for multiparty computation, we design our protocol in the preprocessing model, where the players have access to a preprocessing functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. The preprocessing functionality is used in an initialization phase to generate material for the protocol; it does not require the inputs of the players. Our ideal preprocessing functionality is also represented Figure 1.

Functionality $\mathscr{F}_{\mathsf{ET}}$

The functionality runs with two parties, Alice and Bob. Upon receiving (ET, x) from Alice and (ET, y) from Bob, set $\beta \leftarrow 1$ if $x = y$, and $\beta \leftarrow 0$ else. Set $(a, b) \leftarrow_R \langle \beta \rangle_2$. Return a to Alice and b to Bob.

Functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$

The functionality runs with two parties, Alice and Bob.

Size Reduction: Upon receiving (SR, j) from both players, the functionality picks $(x, y) \leftarrow_R (\mathbb{Z}_2^j)^2$ and sets $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow_R \langle x \oplus y \rangle_{j+1}$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs $(x, \boldsymbol{a})$ to Alice and $(y, \boldsymbol{b})$ to Bob.

Product Sharing: Upon receiving (PS, n) from both players, the functionality picks $(x, y) \leftarrow_R (\mathbb{Z}_2^{2^n-2})^2$ and sets $(a, b) \leftarrow_R \langle x * y \rangle_2$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs (x, a) to Alice and (y, b) to Bob.

Fig. 1: Ideal Functionalities for Equality Test and Preprocessing

Protocol. We now describe our implementation of $\mathscr{F}_{\mathsf{ET}}$ in the $\mathscr{F}_{\mathsf{ET\text{-}prep}}$-hybrid model, with respect to passive corruption. The protocol runs with two players, Alice and Bob. It is parametrized by two integers (ℓ, n), where n is called the *threshold* of the protocol. The players recursively perform size reduction steps using the material produced by the size reduction procedure of $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. Each step reduces inputs of size ℓ to inputs of size $|\ell + 1|$ while preserving the equality predicate. The players stop the reduction when the bitsize of their inputs becomes smaller than the threshold n (taken equal to 3 or 4 in our concrete estimations). The equality predicate is computed on the small inputs with the material produced by the product sharing procedure of $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. The protocol is represented Figure 2.

Theorem 1. *The protocol Π_{ET} securely implements $\mathscr{F}_{\mathsf{ET}}$ in the $\mathscr{F}_{\mathsf{ET\text{-}prep}}$-hybrid model, with respect to passive corruption.*

3.2 Security Analysis

Let Adv be an adversary that interacts with Alice and Bob, running the protocol Π_{ET}. We will construct a simulator $\mathscr{S}im$ which interacts with $\mathscr{F}_{\mathsf{ET}}$, so that no environment Z can distinguish an interaction with Adv in Π_{ET} from an interaction with $\mathscr{S}im$ in the ideal world. $\mathscr{S}im$ starts by invoking a copy of Adv. Each time $\mathscr{S}im$ received from Z an input value, he writes it on Adv's input tape as if coming from Z. Each time Adv writes on its output tape, $\mathscr{S}im$ writes the same thing on his output tape.

One Player is Corrupted. We focus here on the case of a corrupted Bob; as the protocol is essentially symmetrical, the simulation is similar for a corrupted Alice.

Initialize: $\mathscr{S}im$ runs a local copy of $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. He honestly answers to each call to the SR and PS commands, and stores the outputs of each call (This step does not require the input of Alice).

Equality Test:

1. When $\mathscr{S}im$ receives $y'_1 = s_1 \oplus y$ from Adv, he retrieves s_1 from his memory and computes $y = y'_1 \oplus s_1$. $\mathscr{S}im$ sends (ET, y) to $\mathscr{F}_{\mathsf{ET}}$ on behalf of the corrupted party, and receives a bit T. During each round of the size reduction protocol, $\mathscr{S}im$ does only send uniformly random values x'_i of the appropriate size on behalf of Alice. Moreover, for $i \geq 1$ and while $j > n$, $\mathscr{S}im$ stores $y_{i+1} \leftarrow \sum_{l=1}^{j} (-1)^{z_i[l]} \boldsymbol{b_i}[l] + z_i[l] \bmod j + 1$, using the vector $\boldsymbol{b_i}$ stored in $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ and the string $z_i = x'_i \oplus y'_i$.

Protocol Π_{ET}

Initialize: Let $i \leftarrow 1$ and $j \leftarrow \ell$. The players perform the following operations:

- (size reduction) While $j > n$, both players call $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ on input (SR, j) to get outputs $(r_i, \boldsymbol{a}_i)$ and $(s_i, \boldsymbol{b}_i)$. The players set $i \leftarrow i+1$ and $j \leftarrow |j+1|$.
- (product sharing) Both players call $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ on input (PS, n) to get outputs (r, a) and (s, b).

Equality Test: On input two ℓ-bit integers, x from Alice and y from Bob, let $x_1 \leftarrow x$ and $y_1 \leftarrow y$. Let $i \leftarrow 1$ and $j \leftarrow \ell$. The players perform the following operations:

1. While $j > n$, Alice sends $x'_i \leftarrow r_i \oplus x_i$ to Bob, and Bob sends $y'_i \leftarrow s_i \oplus y_i$ to Alice. Let $z_i \leftarrow x'_i \oplus y'_i$. Alice sets $x_{i+1} \leftarrow -\sum_{l=1}^{j}(-1)^{z_i[l]}\boldsymbol{a}_i[l] \bmod j+1$, and Bob sets $y_{i+1} \leftarrow \sum_{l=1}^{j}(-1)^{z_i[l]}\boldsymbol{b}_i[l] + z_i[l] \bmod j+1$. The players set $i \leftarrow i+1$ and $j \leftarrow |j+1|$. Note that $(x_i, y_i) \in \mathbb{Z}_j^2$.
2. Once $j \leq n$, let $(I_k)_{1 \leq k \leq 2^n-2}$ denote the list of non-empty strict subsets of $\{1, \cdots, n\}$ (in any arbitrary fixed order). For $k = 1$ to $2^n - 2$, Alice, sets $X_k \leftarrow \prod_{l \in I_k}(1 \oplus x_i[l])$ and $\alpha_k \leftarrow r[k] \oplus X_k$. Then, Bob sets $Y_k \leftarrow \prod_{l \notin I_k} y_i[l]$, and $\beta_k \leftarrow s[k] \oplus Y_k$. Alice picks $\alpha \leftarrow_R \{0,1\}$ and sends $(\alpha, (\alpha_k)_{k \leq 2^n-2})$, and Bob picks $\beta \leftarrow_R \{0,1\}$ and sends $(\beta, (\beta_k)_{k \leq 2^n-2})$.
3. Alice outputs
$$\bigoplus_{k \leq 2^n-2}(a[k] \oplus \beta_k X_k) \oplus \prod_{l \leq n}(1 \oplus x_i[l]) \oplus \alpha \oplus \beta$$
Bob outputs
$$\bigoplus_{k \leq 2^n-2}(b[k] \oplus \alpha_k s[k]) \oplus \prod_{l \leq n} y_i[l] \oplus \alpha \oplus \beta$$

Fig. 2: Protocol for Equality Test

2. When $\mathscr{S}im$ receives $(\beta, (\beta_k)_{k \leq 2^n-2})$, he retrieves Bob's output (b, s) to the PS command, and picks uniformly random bits $(\alpha_k)_{k \leq 2^n-2}$. $\mathscr{S}im$ sets
$$\alpha \leftarrow \bigoplus_{k \leq 2^n-2}(b[k] \oplus \alpha_k s[k]) \oplus \prod_{l \leq n} y_i[l] \oplus T \oplus \beta$$
and sends $(\alpha, (\alpha_k)_{k \leq 2^n-2})$ to Bob.

Remaining Cases. When both parties are corrupted, $\mathscr{S}im$ simply runs Adv internally. When neither party is corrupted, $\mathscr{S}im$ internally runs Alice and Bob honestly, with inputs $(0,0)$, and forwards the messages exchanged to Adv.

Indistinguishability. We focus on the case where Bob is corrupted; the argument follows symmetrically for a corrupted Alice, and is straightforward when both players are corrupted, or no player is corrupted. We show that the joint view of Z and Adv in the real world is indistinguishable from the view of Z and the simulated Adv in the ideal world; as the simulator perfectly (and honestly) simulates the initialization phase (which does not require the inputs of the parties), we focus on the online phase (and implicitly include the preprocessing material in the view of all the parties). Let t be the number of repetitions of step 1 during the execution of the protocol. As the corrupted player is semi-honest, it honestly follows the specifications of the protocol. Let $(s, b, (x'_i)_{i \leq t}, \alpha, (\alpha_j)_{j \leq 2^n-2})$ be the view of Adv in the online phase during a run of Π_{ET} with inputs (x, y). Let (o_A, o_B) denote the outputs of Alice and Bob, that are sent to Z.

Claim (Correctness of the Size Reduction). For every $i \leq t$, $[x = y] = [x_i = y_i]$.

We show that for every $i \leq t-1$, $[x_i = y_i] = [x_{i+1} = y_{i+1}]$. As $(x, y) = (x_1, y_1)$, the claim follows. Let $i \leq t-1$ be an integer. As the players follow the protocol, it holds that $y_{i+1} - x_{i+1} = \sum_{l=1}^{j}(-1)^{z_i[l]}(\boldsymbol{b}_i[l] + \boldsymbol{a}_i[l]) + z_i[l] \bmod j+1$, with $z_i = x_i \oplus y_i \oplus r_i \oplus s_i$. Furthermore, for any $l \leq j$, $\boldsymbol{b}_i[l] + \boldsymbol{a}_i[l] = r_i[l] \oplus s_i[l] \bmod j+1$. Observe that when $z_i[l] = 0$, it holds that $x_i[l] \oplus y_i[l] = r_i[l] \oplus s_i[l]$, whereas when $z_i[l] = 1$, it holds that $x_i[l] \oplus y_i[l] = 1 - (r_i[l] \oplus s_i[l])$. Overall, it holds that $(-1)^{z_i[l]}(r_i[l] \oplus s_i[l]) + z_i[l] = x_i[l] \oplus y_i[l]$. Therefore, $y_{i+1} - x_{i+1} = \sum_{l=1}^{j}(x_i[l] \oplus y_i[l]) \bmod j+1$. But the right hand term is exactly the Hamming distance between x_i and y_i, which is bounded by the bitsize j of the strings (hence no overflow occurs with the modulus $j+1$). Observe that the Hamming distance between two strings is 0 if and only if the two strings are equal. Therefore, $y_{i+1} - x_{i+1} = 0$ if and only if $x_i = y_i$. The claim follows.

Claim (Correctness of Π_{ET}). The outputs of Alice and Bob in Π_{ET} form shares over $\mathbb{Z}_2$ of $[x = y]$.

By the previous claim, it suffices to show that the output of Alice and Bob in Π_{ET} form shares over $\mathbb{Z}_2$ of $[x_t = y_t]$. It holds that $o_A \oplus o_B = \bigoplus_{k \leq 2^n-2}(a[k] \oplus b[k] \oplus \beta_k X_k \oplus \alpha_k s[k]) \oplus \prod_{l \leq n}(1 - x_t[l]) \oplus \prod_{l \leq n} y_t[l]$. As for $k \leq 2^n - 2$, $\alpha_k = r[k] \oplus X_k$, $\beta_k = s[k] \oplus Y_k$, and $a[k] \oplus b[k] = r[k]s[k]$, this simplifies to $o_A \oplus o_B = \bigoplus_{k \leq 2^n-2} X_k Y_k \oplus \prod_{l \leq n}(1 - x_t[l]) \oplus \prod_{l \leq n} y_t[l]$.

Observe that the right hand term is exactly the product $\prod_{l=1}^{n}((x_t[l] \oplus 1) \oplus y_t[l])$, in developed form. Moreover, this product evaluates to 1 if and only if it holds that for any $l \leq n$, $(x_t[l] \oplus 1) \oplus y_t[l] = 1$, which happens exactly when $x_t[l] = y_t[l]$, and to 0 otherwise. Therefore, $\prod_{l=1}^{n}((x_t[l] \oplus 1) \oplus y_t[l]) = [x_t = y_t]$.

Claim (Indistinguishability). When Bob is corrupted, the joint distribution $(s, b, (x'_i)_{i \leq t}, \alpha, (\alpha_j)_{j \leq 2^n-2}, o_A, o_B)$ is equal to the distribution of transcripts of an interaction with $\mathscr{S}im$ together with the output of $\mathscr{F}_{\mathsf{ET}}$.

Recall that $\mathscr{S}im$ honestly picks (s, b). Moreover, for each $i \leq t$, $x'_i = x_i \oplus r_i$ is perfectly masked by the random value r_i, and for each $j \leq 2^n - 2$, $\alpha_j = r[j] \oplus X_k$ is perfectly masked by the random bit $r[k]$. The value α is simulated so that the output computed by Bob is exactly the output T of $\mathscr{F}_{\mathsf{ET}}$ for the ideal version of Bob. As α is masked by T, which is a uniformly random bit from the viewpoint of Bob (by definition of $\mathscr{F}_{\mathsf{ET}}$), α is also uniform. The outputs of $\mathscr{F}_{\mathsf{ET}}$ form shares of $[x = y]$, as do (o_A, o_B) (from our above analysis). The claim follows. □

3.3 Implementing the Preprocessing Functionality

We now describe the implementation of the functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$, in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model. The protocol is represented Figure 3.

Protocol $\Pi_{\mathsf{ET\text{-}prep}}$

Size-Reduction(ℓ)**:** Alice picks $(x, \boldsymbol{a}) \leftarrow_R \mathbb{Z}_2^\ell \times \mathbb{Z}_{\ell+1}^\ell$, and Bob picks $y \leftarrow_R \mathbb{Z}_2^\ell$. The players call $\mathscr{F}_{\mathsf{OT}}^{\ell,|\ell+1|}$, on input $(x[i] - \boldsymbol{a}[i] \bmod \ell + 1, 1 - \boldsymbol{a}[i] - x[i] \bmod \ell + 1)_{i \leq \ell}$ for Alice and y for Bob. Let $\boldsymbol{b}$ denote Bob's output. Alice outputs $(x, \boldsymbol{a})$ and Bob outputs $(y, \boldsymbol{b})$.

Product-Sharing(n)**:** Alice picks $(x, a) \leftarrow_R (\mathbb{Z}_2^{2^n-2})^2$, and Bob picks $y \leftarrow_R \mathbb{Z}_2^{2^n-2}$. The players call $\mathscr{F}_{\mathsf{OT}}^{2^n-2,2}$ on input $(a[i], a[i] \oplus x[i])_{i \leq 2^n-2}$ for Alice and y for Bob. Let b denote Bob's output. Alice outputs (x, a) and Bob outputs (y, b).

Fig. 3: Preprocessing Protocol for Equality Test

Theorem 2. *The protocol Π_{ET} securely implements $\mathscr{F}_{\mathsf{ET}}$ when calls to $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ in Π_{ET} are replaced by executions of $\Pi_{\mathsf{ET\text{-}prep}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model, with respect to passive corruption.*

Due to lack of space, we postpone the proof of Theorem 6 to Appendix B of the supplementary material. While the proof is rather straightforward, observe that we do not claim that $\Pi_{\mathsf{ET\text{-}prep}}$ UC-securely implements $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ with respect to passive corruption, but rather that the entire protocol remain secure when calls to $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ are replaced by executions of $\Pi_{\mathsf{ET\text{-}prep}}$. The reason for this distinction is that $\Pi_{\mathsf{ET\text{-}prep}}$ does in fact not UC-securely implement $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. Intuitively, this comes from the fact that in $\Pi_{\mathsf{ET\text{-}prep}}$, the parties choose (part of) their outputs themselves; hence, no simulator can possibly force the parties to set their outputs to being equal to the outputs of $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. While this can be solved by adding a resharing step at the end of the protocol, this would add some unecessary interaction and communication to the protocol. Instead, we rely on an approach of [BLLP14], which was developped exactly for this purpose: we prove that the protocol is *input-private* (meaning that there is a simulator producing a view indistinguishable from an execution of the protocol for any environment that ignores the output of the protocol), which, as shown in [BLLP14], suffices to argue the security of the composed protocol as soon as some rules on ordered composition are respected.

3.4 Communication Complexity

We first analyse the complexity of our protocol using any standard oblivious transfer protocol, which transmits $O(\kappa)$ bits. The size reduction procedure transmits $O(\ell\kappa)$ bits on input ℓ, and the product sharing procedure transmits $O(2^n\kappa)$ bits on input n. Setting n to a small constant value (e.g. $n = 3$), the bit communication of Π_{ET} is dominated by its first size reduction procedure, which transmits $O(\kappa\ell)$ bits in the initialization phase. In the online phase, where the equality test is performed using the preprocessed material, only $O(\ell)$ bits are transmitted.

We now analyse the complexity of our protocol in an amortized setting, where many equality test protocols are likely to be invoked (not necessarily in parallel). By a classical observation (see e.g. [LT13]), we can always assume that the inputs of the players are less than κ-bit long: if this is not the case, each party can hash its input first, preserving the correctness of the protocol with overwhelming probability. Therefore, as the largest strings obliviously transferred during the protocol Π_{ET} are $|\ell + 1| \leq |\kappa + 1|$ bit long (for $\kappa = 128$, this corresponds to 8-bit strings), we can benefit from the short-string oblivious transfer extension protocol of [KK13]. Ignoring the computation of the base OTs, which is performed a single time for an arbitrary number of equality tests, k size reduction procedures on ℓ-bit inputs transmit $O(k\ell(\kappa/\log x + x \cdot |\ell|))$ bits, where x is a parameter that can be arbitrarily set so as to minimize this cost. This minimizes to $O(k\ell\kappa/\log\kappa)$, up to some $\log\log$ term. As a consequence, when performing many equality tests, the (amortized) cost of a single equality test is $O(\kappa\ell/\log\kappa)$ bits in the preprocessing phase (and still $O(\ell)$ bits in the online phase). For inputs of size $\ell > \kappa$, where the players can hash their input first, the complexity becomes $O(\kappa^2/\log\kappa)$ in the preprocessing phase, and $O(\kappa)$ in the online phase.

3.5 Concrete Efficiency

We now analyze the efficiency of our protocol for various input-lengths. In all our numerical applications, we set the security parameter κ to 128. We estimate both the efficiency in a single run setting, and in an amortized setting, where we can use oblivious transfer extension.

Comparison with Equality Test from Garbled Circuit. We compare our protocol to the garbled-circuit-based protocol of [KSS09], which is to our knowledge the most efficient state-of-the-art protocol for equality test with shared output. Let us provide an intuition of this protocol; details on garbled circuits and the free-xor trick can be found in [KS08]. First, the equality test function is represented as a circuit with $\ell - 1$ equality gates (with carry), each gate being implemented with three xor gates (which are for free in the construction, in the sense that they do not require communicating anything) and a AND gate. During the preprocessing phase, Alice starts by assigning two keys to each wire of the circuit (corresponding to the two possible values 0 and 1), and for each gate g, she computes a *garbled gate*, which encrypts the output-wire keys of the gate so that given two input-wire keys corresponding to inputs (b, b'), only the output-wire key corresponding to $g(b, b')$ can be recovered. Using the recent result of [ZRE15], each AND gate can be garbled using two $(\kappa + 1)$-bit strings. In the online phase, Alice sends to Bob the ℓ keys corresponding to the bits of her entry, and acts as sender in ℓ parallel oblivious transfers, using as input each pair of keys corresponding to the possible values for an input of Bob. Bob's selection bits are his input bits. Once he has recovered the keys, Bob can evaluate the circuit securely.

Note that the pairs of keys in this scheme satisfy some correlation, hence the optimization of [ALSZ13] for correlated oblivious transfer extensions can be applied. However, this prevents Alice from constructing and sending the garbled circuit during the preprocessing phase, as the values of the keys will be determined by the correlated oblivious transfers (in which one of the outputs is a random value). In our estimations, we chose not to use this optimization, which results in a slight loss in overall communication, but almost cuts in half the communication during the online phase. We use random OTs [ALSZ13] to push the cost of executing oblivious transfers to the offline phase, at the (mild) cost of adding ℓ bits to the total communication. Observe also that the method of [AIKW13] could be applied to reduce the communication of online phase of [KSS09]; however, this would require sending $O(\ell^2)$ group elements in the preprocessing phase (or $O(\ell)$ using the RSA based instantiation) and computing $O(\ell^2)$ (resp. $O(\ell)$)

modular multiplications online. In an amortized setting where [KSS09] requires only cheap symmetric operations, this would strongly affect both the computation and the communication of the protocol.

Non-Amortized Setting. We now evaluate the concrete efficiency of our protocol. We first focus on the simpler setting, the non-amortized setting, in which a single ET will be performed. We stop the size reduction protocol as soon as $n \leq 4$ (stopping at $n \leq 3$ or $n \leq 2$ saves a few hundreds of bits for some sizes of ℓ, at the cost of additional rounds). For the oblivious transfer, we use the scheme of Naor and Pinkas [NP01], whose security in the random oracle model relies on the decisional Diffie-Hellman (DDH) assumption (we use an elliptic curve of prime order p, which can be taken of bit-size $\log p = 2\kappa$ according to recommended parameters). The protocols of [NP01] have the following communication:

- t executions of a $\binom{2}{1}$-OT_ℓ on strings of size $\ell \leq \kappa$ transmit $\kappa 4t$ bits. The initialization phase consists of two group elements sent by the sender which amounts to 4κ bits.
- t executions of a $\binom{N}{1}$-OT_ℓ on strings of size $\ell \leq \kappa$ transmit $\kappa(N+2)t$ bits. The initialization phase consists of $N+1$ group elements sent by the sender which amounts to $2(N+1)\kappa$ bits.

We will use this $\binom{N}{1}$-OT_ℓ in a crucial way. One of the constructions described in [NP01] gives a trade-off between communication (which is increased) and computation (which is decreased). The idea is that to perform $\log N$ oblivious transfers on ℓ-bit strings, it suffices to perform a single $\binom{N}{1}$-$\mathsf{OT}_{\ell \log N}$, in which the N inputs are all the 2^{logN} possible concatenations of one input from each of the $\log N$ input pairs. But recall that the communication of the protocol of [NP01] is always the same for any $\ell \leq \kappa$ (for larger values, the oblivious transfer are performed on keys, which are used to encrypt the values, adding 2ℓ bits of overhead to the protocol). Our equality test protocol involves only oblivious transfers on very small strings, of size smaller than $|\kappa + 1|$. Hence, by picking a sufficiently small N so that $\ell \log N \leq \kappa$, the trade-off protocol of Naor and Pinkas does in fact *reduce* the communication. Indeed, performing $\log N$ oblivious transfers on short strings transmits $4\kappa \log N$ bits, while using instead a single $\binom{N}{1}$-$\mathsf{OT}_{\ell \log N}$ transmits $(N+2)\kappa$ bits if $\ell \log N \leq \kappa$. This amounts to $(N+2)\kappa/\log N$ bits per $\binom{2}{1}$-OT, which is minimized for $N = 4$ and transmits 3κ bits. Hence, performing the $\binom{2}{1}$-OT by pairs, as a single $\binom{4}{1}$-OT, reduces the communication by 25% when the transmitted strings are of size $\ell \leq \kappa/2$. We note that, to our knowledge, this simple observation was not made before and is of independant interest.

Table 2 sums up the costs of our equality test protocol for various values of ℓ, and compares it to the garbled-circuit-based protocol of [KSS09]. Note that we use the oblivious transfer of [NP01] in both our ET and the protocol of [KSS09], but while we can use the optimization described above to reduce the communication in our protocol (as it transmits short strings), this does not hold for garbled circuits, in which the transmitted values are κ-bit keys (and our optimization does not result in any improvement in this case). Hence, t $\binom{2}{1}$-OT transmit $3\kappa t$ bits in our ET, but $4\kappa t$ bits in [KSS09]. As one can see from Table 2, our protocol transmits more bits than [KSS09] in the preprocessing phase, but has a communication two orders of magnitudes smaller in the online phase. Overall, our protocol is approximately 50% more efficient than [KSS09].

Amortized Setting. We now provide a concrete efficiency analysis of the protocol in an amortized setting, using oblivious transfer extensions. We do not take into account the cost of the base oblivious transfers for the OT extension scheme, as this is a constant independent of the number of equality tests performed, which is the same for both our protocol and the protocol of [KSS09]. Adapting the construction of [KK13] to the case of correlated short inputs, the exact cost of reducing m oblivious transfers of t-bit strings to κ oblivious transfers of κ-bit strings is $m(2\kappa/\log x + (x-1)t)$ (this takes into account an optimization described in [KK13, Appendix A] and the optimization for correlated inputs of [ALSZ13]).Therefore, the amortized cost of a size reduction protocol on input k is $k(2\kappa/\log x + (x-1)k)$, where x can be chosen so as to minimize this cost. Table 2 sums up the amortized costs of our equality test protocol for various values of ℓ, and compares it again with [KSS09]; oblivious transfers for the garbled circuit approach of [KSS09] are performed using the OT extension protocol of [ALSZ13] on κ-bit inputs, which transmits 3κ bits per OT. As shown in Table 2, our protocol improves over the communication of [KSS09] by up to 80% overall. During the online phase, our protocol is extremely efficient, two orders of magnitude faster than [KSS09].

Table 2: Communication of ℓ-bit ET

	Single Run Setting				Amortized Setting[1]			
	Our ET		[KSS09]		Our ET		[KSS09]	
ℓ	comm.[2]	rounds	comm.	rounds	comm.	rounds	comm.	rounds
Preprocessing Phase								
4	5376	2	1810	1	1106	2	1288	1
8	8448	3	3856	1	2018	3	2832	1
16	10365	4	7968	1	2945	4	5920	1
32	16896	4	16192	1	5212	4	12096	1
64	29568	4	32640	1	9863	4	24448	1
128	57600	4	65536	1	20194	4	49152	1
Online Phase								
4	28	1	1540	2	28	1	1540	2
8	44	2	3080	2	44	2	3080	2
16	54	3	6160	2	54	3	6160	2
32	88	3	12320	2	88	3	12320	2
64	154	3	24640	2	154	3	24640	2
128	300	3	49280	2	300	3	49280	2

[1] The one-time cost of the base OTs is ignored in the amortized setting.
[2] Comm. denotes the number of bits exchanged during a protocol run.

Amortized Computational Complexity. The computational complexity of both [KSS09] and our protocol are directly proportional to their communication in the amortized setting (and it is dominated by the evaluation of hash functions in both, which are required for (extended) OTs and garbled gates), hence our constructions improve upon [KSS09] regarding computation by factors similar to those listed in Table 2.

4 Secure Comparison from Equality Test

In this section, we design a secure comparison protocol (SC) to securely evaluate shares (over $\mathbb{Z}_2$) of the greater-than predicate on inputs held by Alice and Bob. The high level intuition of our construction is close in spirit to the construction of equality test in the previous section: the two parties will play several rounds of a size-reduction procedure, that will shrink the size of their inputs while preserving their ordering. When the input size has been skrinked below some threshold, the parties compute the output using a protocol tailored to secure comparison on small inputs. However, the actual construction differs significantly from the protocol of the previous section, and is considerably more involved. We detail below the main steps of our construction.

- The size reduction protocol takes as input two bitstrings (x, y) of length k, on known to Alice and one known to Bob. It divides the string into $\sqrt{k}$ blocks of $\sqrt{k}$ bits. Then, it relies on equality tests to obliviously identify the first block of both x and y on which they differ, and on oblivious transfers to obliviously extract these two blocks. At the end of the protocol, the parties hold shares of two $\sqrt{k}$-bit blocks (x', y') satisfying $[x' \leq y'] = [x \leq y]$.
- From there, the two parties would like to recursively invoke the above procedure, until they end up with small strings to be compared. However, this cannot be done right away: the size reduction protocol takes as inputs values *known to Alice and Bob*, but outputs *shares of values to be compared*. Nevertheless, we show that Alice and Bob can *non-interactively* reduce the task of comparing shared values to the task of comparing values that they know. More specifically, we show an equality of the form

$$[x' \leq y'] = b_A \oplus b_B \oplus [v_A \leq v_B] \tag{1}$$

Functionality $\mathscr{F}_{\mathsf{SC}}$

The functionality runs with two parties, Alice and Bob. Upon receiving (SC, x) from Alice and (SC, y) from Bob, set $\beta \leftarrow 1$ if $x \leq y$, and $\beta \leftarrow 0$ else. Set $(a, b) \leftarrow_R \langle \beta \rangle_2$. Return a to Alice and b to Bob.

Functionality $\mathscr{F}_{\mathsf{shrink}}$

The functionality runs with two parties, Alice and Bob. Upon receiving $(\mathsf{shrink}, \ell, \lambda, x)$ from Alice and $(\mathsf{shrink}, \ell, \lambda, y)$ from Bob, where λ divides ℓ and (x, y) are ℓ-bit long, it divides the bits of x and y into ℓ/λ consecutive blocks of λ bit, and computes $\hat{x} \in \{0,1\}^\lambda$ (resp. $\hat{y} \in \{0,1\}^\lambda$) as the first block of x (resp. y) on which x and y differ. Then, it computes $(\hat{x}_A, \hat{x}_B) \leftarrow_R \langle \hat{x} \rangle_{2^{\lambda+1}}$, $(\hat{y}_A, \hat{y}_B) \leftarrow_R \langle \hat{y} \rangle_{2^{\lambda+1}}$, and returns $(\hat{x}_A, \hat{y}_A)$ to Alice and $(\hat{x}_B, \hat{y}_B)$ to Bob.

Functionality $\mathscr{F}_{\mathsf{SC\text{-}prep}}$

The functionality runs with two parties, Alice and Bob.

Size Reduction: Upon receiving $(\mathsf{SR}, \lambda, \mu)$ from both players, the functionality picks $(\boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u_0}, \boldsymbol{u_1}) \leftarrow_R (\mathbb{Z}_{\mu+1}^{\mu})^2 \times (\mathbb{Z}_{2^{\lambda+1}}^{\mu})^4$ and $(c, d, e) \leftarrow_R (\mathbb{Z}_2^{\mu})^3$. $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ sets $(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u}) \leftarrow \left((\boldsymbol{s_{c[i]}}[i])_{i \leq \mu}, (\boldsymbol{t_{d[i]}}[i])_{i \leq \mu}, (\boldsymbol{u_{e[i]}}[i])_{i \leq \mu} \right)$

It outputs $(e, \boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u})$ to Alice and $(c, d, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u_0}, \boldsymbol{u_1})$ to Bob.

Product Sharing: Upon receiving (PS, n) from both players, the functionality picks $(\rho, \sigma) \leftarrow_R (\mathbb{Z}_2^{2^n-1})^2$ and sets $(a, b) \leftarrow_R \langle \rho * \sigma \rangle_2$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs (ρ, a) to Alice and (σ, b) to Bob.

Fig. 4: Ideal Functionalities for Secure Comparison and Preprocessing

where (b_A, v_A) (resp. (b_B, v_B)) are values that Alice (resp. Bob) can compute *locally* from her shares of (x', y') ((b_A, b_B) ar bits). Therefore, the parties will invoke the size reduction procedure on the values (v_A, v_B) of the above equality, and store the values (b_A, b_B).

- The equality 1 will crucially relies on the assumption that $x \neq y$. Observe that if the initial inputs (x, y) are not equal, then the reduced values (x', y') are also different (they are the first block of x and y on which the inputs differ). From the proof of 1, it will follow that in this case, it also holds that $v_A \neq v_B$. Therefore, to ensure that the inputs to be compared remain different throughout the entire computation, it suffices to ensure that the initial inputs are not equal. This can be ensure very easily as follows: Alice will append a 0 to her initial input, and Bob will append a 1 to his initial input. This ensures that $x \neq y$, while preserving the value of the predicate $[x \leq y]$.
- After $\log \ell$ such size reduction and non-interactive conversion steps (for initial inputs of length ℓ), the parties end up with small inputs to be compared, and bits locally computed during the non-interactive conversions. They invoke an OT-based small-input secure comparison protocol (similar to the OT-based small-input equality test protocol used in the previous section), and xor their outputs with all their locally computed bits.

Ideal Functionalities. The ideal functionality $\mathscr{F}_{\mathsf{SC}}$ for our SC protocol is represented on Figure 4. To simplify the exposition, we describe an intermediate functionality $\mathscr{F}_{\mathsf{shrink}}$, which performs the size reduction procedure. We will implement this functionality in the $(\mathscr{F}_{\mathsf{ET}}, \mathscr{F}_{\mathsf{SC\text{-}prep}})$-hybrid model afterward. Both $\mathscr{F}_{\mathsf{shrink}}$ and $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ are represented on Figure 4.

Reduction Lemma. Let (x, y) be two bitstrings of length k. Let $t \leftarrow 2^{k+1}$. Let $(x_A, x_B) \leftarrow_R \langle x \rangle_t$ and $(y_A, y_B) \leftarrow_R \langle y \rangle_t$; that is, (x_A, x_B) (resp. (y_1, y_B)) are random shares of x (resp. y) over $\mathbb{Z}_t$. Let $(z_A, z_B) \leftarrow (y_A - x_A \bmod t, y_B - x_B \bmod t)$. We prove the following lemma:

Lemma 3. *If $x \neq y$, then it holds that $[x \leq y] = b_A \oplus b_B \oplus [w_A \leq t/2 - w_B]$, where $b_A = [z_A < t/2]$, $b_B = [z_B < t/2]$, $w_A = z_A \bmod t/2$, and $w_B = z_B \bmod t/2$.*

Suppose that Alice and Bob are given *shares* of two k-bit values (x, y) to be compared; that is, Alice knows (x_A, y_A), Bob knows (x_B, y_B), and the parties want to compute shares of $[x \leq y]$. Lemma 3 shows that (if $x \neq y$) the parties can *non-interactively* reduce this task to the task of comparing two values, one known to Alice and one known to Bob. Indeed, given (x_A, y_A), Alice (resp. Bob) can locally

Protocol Π_{SC}

Initialize: Both players call $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ on input (PS, n) to get (ρ, a) and (σ, b).

Secure Comparison: On input two ℓ-bit integers, x from Alice and y from Bob, let $i \leftarrow 1$, $x'_1 \leftarrow x$, and $y'_1 \leftarrow y$.

1. The players agree on an integer λ_i which divides ℓ (the optimal parameters for this block decomposition will depend on the actual implementation of $\mathscr{F}_{\mathsf{shrink}}$). Alice calls $\mathscr{F}_{\mathsf{shrink}}$ on input $(\mathsf{shrink}, \ell, \lambda_i, x_i)$, and Bob on input $(\mathsf{shrink}, \ell, \lambda_i, y_i)$. Let (x_{i+1}, y_{i+1}) denote the first blocks where x_i differs from y_i. We denote (x^A_{i+1}, y^A_{i+1}) and (x^B_{i+1}, y^B_{i+1}) the outputs of $\mathscr{F}_{\mathsf{shrink}}$ to Alice and Bob, which form shares of (x_{i+1}, y_{i+1}).
2. From these shares, the parties locally compute bits (b^A_{i+1}, b^B_{i+1}) and values (x'_{i+1}, y'_{i+1}) satisfying
$$[x'_{i+1} \leq y'_{i+1}] \oplus b^A_{i+1} \oplus b^B_{i+1} = [x_{i+1} \leq y_{i+1}],$$
where (b^A_{i+1}, x'_{i+1}) is known to Alice and (b^B_{i+1}, y'_{i+1}) is known to Bob.
3. The players set $\ell \leftarrow \lambda_i$ and $i \leftarrow i+1$. If $|\ell| > n-1$, the players iterate the steps 1 to 3. Otherwise, they go to step 4.
4. Let $f : (j,l) \mapsto l - 1 + 2^{j-1}$. For $j = 1$ to $n-1$, let $(I^j_l)_{1 \leq l \leq 2^j}$ denote the list of subsets of $\{1, \cdots, j\}$ (in any arbitrary fixed order). For $j = 1$ to n, for $l = 1$ to 2^{j-1}, Alice picks $\alpha \leftarrow_R \{0,1\}$ and sets $\alpha_{jl} \leftarrow \rho[f(j,l)] \oplus x_i[j] \cdot \prod_{k \in I^{j-1}_l} (1 \oplus x_i[k])$, and Bob picks $\beta \leftarrow_R \{0,1\}$ and sets $\beta_{jl} \leftarrow \sigma[f(j,l)] \oplus (1 \oplus y_i[j]) \cdot \prod_{k \notin I^{j-1}_l} y_i[k]$. Alice sends $(\alpha, (\alpha_{jl})_{jl})$ and Bob sends $(\beta, (\beta_{jl})_{jl})$ (this amounts to 2^{n+1} bits exchanged).
5. Alice outputs
$$\left(\bigoplus_{k=1}^{i} b^A_i\right) \oplus \left(\bigoplus_{\substack{j \leq n \\ l \leq 2^j - 1}} \rho[f(j,l)]\beta_{jl} \oplus a[f(j,l)]\right) \oplus \alpha \oplus \beta$$
Bob outputs
$$\left(\bigoplus_{k=1}^{i} b^B_k\right) \oplus \left(\bigoplus_{\substack{j \leq n \\ l \leq 2^j - 1}} (\sigma[f(j,l)] \oplus \beta_{jl})\alpha_{jl} \oplus b[f(j,l)]\right) \oplus \alpha \oplus \beta$$

Fig. 5: Protocol for Secure Comparison

compute b_A and w_A (resp. b_B and w_B) as in Lemma 3. Hence, to compute shares of $[x \leq y]$, Alice and Bob can simply compute shares of $[w_A \leq t/2 - w_B]$ (where Alice knows w_A and Bob knows $t/2 - w_B$), and locally xor their outputs with b_A and b_B. Due to lack of space, we formally prove the reduction lemma in Appendix B of the supplementary material.

4.1 Main Protocol

We now describe our implementation of $\mathscr{F}_{\mathsf{SC}}$, in the $(\mathscr{F}_{\mathsf{SC\text{-}prep}}, \mathscr{F}_{\mathsf{shrink}})$-hybrid model, with respect to passive corruption. The protocol runs with two players, Alice and Bob. It is parametrized by two integers (ℓ, n), where n is the threshold of the protocol. The players recursively perform size reduction steps $\mathscr{F}_{\mathsf{shrink}}$ and the reduction lemma. Each step reduces inputs of size ℓ to inputs of size roughly $\sqrt{\ell}$ while preserving the comparison predicate. The players stop the reduction when the bitsize of their inputs becomes smaller than the threshold n (taken equal to 3 or 4 in our concrete estimations). The comparison predicate is computed on the small inputs with the material produced by the product sharing procedure (PS) of $\mathscr{F}_{\mathsf{SC\text{-}prep}}$. The protocol is represented Figure 5. For simplicity, we assume from now that the inputs (x, y) satisfy $x \neq y$; as previously outlined, this can be ensured by letting Alice append a 0 and Bob a 1 to their initial inputs.

Theorem 4. *The protocol Π_{SC} securely implements $\mathscr{F}_{\mathsf{SC}}$ in the $(\mathscr{F}_{\mathsf{SC\text{-}prep}}, \mathscr{F}_{\mathsf{shrink}})$-hybrid model, with respect to passive corruption.*

4.2 Security Analysis

Let Adv be an adversary that interacts with Alice and Bob, running the protocol Π_{SC}. We will construct a simulator $\mathscr{S}im$ which interacts with $\mathscr{F}_{\mathsf{SC}}$, so that no environment Z can distinguish an interaction with Adv in Π_{SC} from an interaction with $\mathscr{S}im$ in the ideal world. $\mathscr{S}im$ starts by invoking a copy of Adv. Each time $\mathscr{S}im$ receives from Z an input value, he writes it on Adv's input tape as if coming from Z. Each time Adv writes on its output tape, $\mathscr{S}im$ writes the same thing on his output tape.

One Player is Corrupted. We focus here on the case of a corrupted Bob; as the protocol is essentially symmetrical, the simulation is similar for a corrupted Alice.

Initialize: $\mathscr{S}im$ runs local copies of $(\mathscr{F}_{\mathsf{shrink}}, \mathscr{F}_{\mathsf{SC\text{-}prep}})$. He honestly answers to the call to the PS command, and stores the output. (This step does not require the input of Alice)

Secure Comparison:

1. When $\mathscr{S}im$ receives $(\mathsf{shrink}, \ell, \lambda_1, y_1)$ from Bob, he stores $y = y_1$ and sends (SC, y) to $\mathscr{F}_{\mathsf{SC}}$ on behalf of Bob in the ideal world. $\mathscr{S}im$ receives an output bit T. For each compression round i, $\mathscr{S}im$ simulates $\mathscr{F}_{\mathsf{shrink}}$ by returning two random shares of the appropriate size to Bob, and computes and stores the corresponding value b^B_{i+1}.
2. When $\mathscr{S}im$ receives $(\beta, (\beta_{jl})_{jl})$, he retrieves Bob's output (σ, b) to the PS command, and picks uniformly random bits $(\alpha_{jl})_{j\leq n-1, l\leq 2^{j-1}}$. $\mathscr{S}im$ sets
$$\alpha \leftarrow \left(\bigoplus_{k=1}^{i} b^B_k\right) \oplus \left(\bigoplus_{\substack{j\leq n \\ l\leq 2^j-1}} (\sigma[f(j,l)] \oplus \beta_{jl})\alpha_{jl} \oplus b[f(j,l)]\right) \oplus T \oplus \beta$$
and sends $(\alpha, (\alpha_{jl})_{jl})$ to Bob.

Remaining Cases. When both parties are corrupted, $\mathscr{S}im$ simply runs Adv internally. When neither party is corrupted, $\mathscr{S}im$ internally runs Alice and Bob honestly, with inputs $(0,0)$, and forwards the messages exchanged to Adv.

Indistinguishability. We focus on the case where Bob is corrupted; the argument follows symmetrically for a corrupted Alice, and is straightforward when both players are corrupted, or no player is corrupted. We show that the joint view of Z and Adv in the real world is indistinguishable from the view of Z and the simulated Adv in the ideal world. In steps 1 to 3, $\mathscr{S}im$ perfectly simulates the answers of the functionality $\mathscr{F}_{\mathsf{shrink}}$, and stores all the corresponding bits b^B_{i+1} computed by Bob. During an execution of the real protocol, in addition to Bob's interaction with $\mathscr{F}_{\mathsf{shrink}}$, the environment sees $(\alpha, (\alpha_{jl})_{jl})$, as well as the outputs (o_A, o_B) of the players. In the ideal world, on input (x, y), the outputs of $\mathscr{F}_{\mathsf{SC}}$ are random shares of $[x \leq y]$; let us first show that this also holds for the outputs (o_A, o_B) in the real world.

Claim (Correctness of Π_{SC}). $o_A \oplus o_B = [x \leq y]$.

Let t be the number of repetitions of step 1. By definition of $\mathscr{F}_{\mathsf{shrink}}$, and by using the recursion lemma (Lemma 3) at each step, it holds that $[x_t < y_t] = [x_t \leq y_t] = [x \leq y] \oplus \bigoplus_{k=1}^{t} b^B_k$ (recall that $x_t \neq y_t$). We now show that $o_A \oplus o_B \oplus \bigoplus_{k=1}^{t} b^B_k = [x_t < y_t]$. Replacing $(\alpha_{jl}, \beta_{jl})_{jl}$ by their corresponding expression, and using the fact that $\rho[l]\sigma[l] = a[l] \oplus b[l]$, we get

$$o_A \oplus o_B \oplus \bigoplus_{k=1}^{t} b^B_k = \bigoplus_{j\leq n} x_t[j](1 \oplus y_t[j]) \left(\bigoplus_{l\leq 2^j-1} \prod_{k\in I_l^{j-1}} (1 \oplus x_t[k]) \cdot \prod_{k\notin I_l^{j-1}} y_t[k]\right)$$

The term between the parenthesis is simply the product $\prod_{k\leq j-1}(1 \oplus x_t[k] \oplus y_t[t])$ developed. This product evaluates to 1 if and only if it holds for each $k \leq j-1$ that $x_t[k] = y_t[k]$ (which is equivalent to $1 \oplus x_t[k] \oplus y_t[k] = 1$). Observe now that $[x_t < y_t]$ can be computed recursively using the following formula:

$$[x_t < y_t] = [x_t[1] < y_t[1]] \oplus [x_t[1] = y_t[1]] \cdot [x_t[2] \cdots x_t[n] < y_t[2] \cdots y_t[n]]$$

As for any $j \leq n$, $[x_t[j] < y_t[j]] = (1 \oplus x_t[j])y_t[j]$ and $[x_t[j] = y_t[j]] = 1 \oplus x_t[j] \oplus y_t[j]$, recursively applying the above formula gives

$$[x_t < y_t] = \bigoplus_{j\leq n} x_t[j](1 \oplus y_t[j]) \left(\prod_{k=1}^{j-1}(1 \oplus x_t[k] \oplus y_t[k])\right)$$

Protocol Π_{shrink}

Let (ℓ, λ) be two integers such that $\lambda \leq \ell$. Let μ be the smallest integer such that $\lambda\mu \geq \ell$. On input x from Alice and y from Bob, both of size ℓ-bit,

Initialize: The players call $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ on input $(\mathsf{SR}, \lambda, \mu)$ to get outputs $(e, \boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u})$ for Alice and $(c, d, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u_0}, \boldsymbol{u_1})$ for Bob.

Compression: Let $(x_j)_{j\leq\mu} \in \mathbb{Z}_{2^\lambda}$ (resp. $(y_j)_{j\leq\mu} \in \mathbb{Z}_{2^\lambda}$) be the decomposition of x (resp. y) into μ blocks of size λ (i.e., $x = \sum_{j=1}^{\mu} x_j 2^{\lambda(j-1)}$ and $y = \sum_{j=1}^{\mu} y_j 2^{\lambda(j-1)}$). The players perform the following operations:

1. (Equality Tests.) For $j = 1$ to μ, the players call $\mathscr{F}_{\mathsf{ET}}$ on inputs (ET, x_j) and (ET, y_j). Let $(\alpha_j, 1 \oplus \beta_j)_{j\leq\mu} \in \mathbb{Z}_2^{2\mu}$ denote their respective outputs (that is, (α_j, β_j) form shares of 0 if $x_j = y_j$, and of 1 otherwise). This step allows Alice and Bob to (obliviously) identify the blocks of x and y which are not equal.
2. (Modulus Change.) Alice picks $\boldsymbol{r} \leftarrow_R \mathbb{Z}_{\mu+1}^{\mu}$. For $j = 1$ to μ, Bob sends $d_j \leftarrow \beta_j \oplus c[j]$ to Alice. For $j = 1$ to μ, Alice sends the pair

$$(a_{j,0}, a_{j,1}) \leftarrow (\boldsymbol{s_{d_j}}[j] + \alpha_j + \boldsymbol{r}[j], \boldsymbol{s_{1-d_j}}[j] + (1 - \alpha_j) + \boldsymbol{r}[j]) \bmod \mu + 1$$

   to Bob. For $j = 1$ to μ, Bob computes $y'_j \leftarrow \sum_{k=1}^{j} a_{k,c[k]} - \boldsymbol{s}[k] \bmod \mu + 1$ and Alice computes $x'_j \leftarrow \sum_{k=1}^{j} \boldsymbol{r}[k] \bmod \mu + 1$. This steps converts Alice and Bob's shares of the $[x_j \neq y_j]$ from shares modulo 2 to shares modulo $\mu + 1$, and computes all partial sums (from 1 to j) of these shares.
3. (Identiying the First Different Block.) For $j = 1$ to μ, the players call $\mathscr{F}_{\mathsf{ET}}$ on inputs (ET, x'_j) and (ET, y'_j). Let $(\alpha'_j, 1 \oplus \beta'_j)_{j\leq\mu} \in \mathbb{Z}_2^{2\mu}$ denote their respective outputs (that is, (α'_j, β'_j) form shares of 0 if $x'_j = y'_j$, and of 1 otherwise) and $(\alpha'_0, \beta'_0) \leftarrow (0, 0)$. For $j = 1$ to μ_i, Alice sets $\gamma_j \leftarrow \alpha_{j-1} \oplus \alpha_j$ and Bob sets $\delta_j \leftarrow \beta_{j-1} \oplus \beta_j$. The following steps 4 and 5 are executed in parrallel:
4. (Selecting the First Different Block – Alice's Step.) Alice picks $\boldsymbol{r_A} \leftarrow_R \mathbb{Z}_{2^{\lambda+1}}^{\mu}$. For $j = 1$ to μ, Bob sends $d_j^B \leftarrow \delta_j \oplus d[j]$ to Alice, and Alice sends the pair

$$(a'_{j,0}, a'_{j,1}) \leftarrow (\boldsymbol{t_{d_j^B}}[j] + \gamma_j x_j + \boldsymbol{r_A}[j], \boldsymbol{t_{1-d_j^B}}[j] + (1 - \gamma_j)x_j + \boldsymbol{r_A}[j]) \bmod 2^{\lambda+1}$$

   to Bob. Bob computes $\hat{y}_B \leftarrow \sum_{j=1}^{\mu} a'_{j,c[j]} - \boldsymbol{t}[j] \bmod 2^{\lambda+1}$, and Alice sets $\hat{y}_A \leftarrow -\sum_{j=1}^{\mu} \boldsymbol{r_A}[j] \bmod 2^{\lambda+1}$.
5. (Selecting the First Different Block – Bob's Step.) Bob picks $\boldsymbol{r_B} \leftarrow_R \mathbb{Z}_{2^{\lambda+1}}^{\mu}$. For $j = 1$ to μ, Alice sends $d_j^A \leftarrow \gamma_j \oplus e[j]$ to Bob, and Bob sends the pair

$$(b_{j,0}, b_{j,1}) \leftarrow (\boldsymbol{u_{d_j^A}}[j] + \delta_j y_j + \boldsymbol{r_B}[j], \boldsymbol{u_{1-d_j^A}}[j] + (1 - \delta_j)y_j + \boldsymbol{r_B}[j]) \bmod 2^{\lambda+1}$$

   to Alice. Alice computes $\hat{x}_A \leftarrow \sum_{j=1}^{\mu} b_{j,c[j]} - \boldsymbol{u}[j] \bmod 2^{\lambda+1}$, and Bob sets $\hat{y}_A \leftarrow -\sum_{j=1}^{\mu} \boldsymbol{r_B}[j] \bmod 2^{\lambda+1}$.

Output: Alice outputs $(\hat{x}_A, \hat{y}_A)$ and Bob outputs $(\hat{x}_B, \hat{y}_B)$.

Fig. 6: Compression Protocol for Secure Comparison

Which concludes the proof of the claim. Moreover, each value α_{jl} sent during the protocol is perfectly masked by $\rho[f(j, l)]$, hence all the α_{jl} are perfectly indistinguishable from uniformly random values, and all the simulated α_{jl} are uniformly random bits. α is random in the real protocol, and is masked by the output T of $\mathscr{F}_{\mathsf{SC}}$ in the simulated protocol, which is a uniformly random bit by definition of $\mathscr{F}_{\mathsf{SC}}$. It is straightforward to see that the semi-honest Bob will indeed obtain the bit T as output in the simulated protocol. Therefore, the joint view of Z and Adv in the real protocol is perfectly indistinguishable from their joint view in the simulated protocol. □

4.3 Compression Functionality

We now implement the functionality $\mathscr{F}_{\mathsf{shrink}}$, in the $(\mathscr{F}_{\mathsf{SC\text{-}prep}}, \mathscr{F}_{\mathsf{ET}})$-hybrid model. The protocol is represented Figure 6.

Theorem 5. ***Theorem 6.*** *The protocol Π_{ET} securely implements $\mathscr{F}_{\mathsf{ET}}$ when calls to $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ in Π_{ET} are replaced by executions of $\Pi_{\mathsf{ET\text{-}prep}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model, with respect to passive corruption.*

The protocol Π_{SC} securely implements the functionality $\mathscr{F}_{\mathsf{SC}}$ when calls to $\mathscr{F}_{\mathsf{shrink}}$ in Π_{SC} are replaced by executions of Π_{shrink} in the $(\mathscr{F}_{\mathsf{ET}}, \mathscr{F}_{\mathsf{SC\text{-}prep}})$-hybrid model, with respect to passive corruption.

We postpone the proof of this theorem to the supplementary material, Appendix B, noting that the proof is rather straightforward. Below, we provide an explanation of its correctness. The general idea of

the protocol is that to compare two strings, it suffices to divide these strings in blocks, and to compare the first block on which they differ. The purpose of the compression step is for the players to obliviously select this block. The inputs (x, y) are first divided into μ blocks of size λ. At the end of step 1, the players obtain shares (α_j, β_j) (over $\mathbb{Z}_2$) of all the bits $[x_j \neq y_j]$. During step 2, the players compute values (x'_j, y'_j) whose difference modulo $\mu+1$ is $\sum_{k=1}^{j}[x_k \neq y_k]$. This requires to use some preprocessed material. Let j^* be the first block on which x differs from y. Observe that $\sum_{k=1}^{j}[x_k \neq y_k] = 0$ for $j < j^*$, and $\sum_{k=1}^{j}[x_k \neq y_k] > 0$ afterward. The players perform in step 3 equality tests on the values (x'_j, y'_j). Therefore, they obtain shares of the bits $[x'_i - y'_j = 0]$. Observe that these bits are 1 for $j \leq j^*$, and 0 afterward. From these shares, the players can locally compute shares (γ_j, δ_j) of bits which are 0 for every $j \neq j^*$, and 1 only for $j = j^*$, by xoring pairs of consecutive bits. In step 4 and 5, using preprocessed material again, the players compute shares of $\sum_{j=1}^{\mu}(\gamma_j \oplus \delta_j)x_j = x_{j^*}$ and $\sum_{j=1}^{\mu}(\gamma_j \oplus \delta_j)y_j = y_{j^*}$, which correspond to the target outputs.

4.4 Implementing the Preprocessing Functionality

The implementation of the functionality $\mathscr{F}_{\mathsf{SC\text{-}prep}}$, in the $(\mathscr{F}_{\mathsf{OT}}, \mathscr{F}_{\mathsf{ROT}})$-hybrid model, is represented Figure 7, and the security claim is given in Theorem 7.. The proof of this theorem is essentially identical to the proof of Theorem 6.

Protocol $\Pi_{\mathsf{SC\text{-}prep}}$

Size-Reduction(ℓ)**:** The players perform the following operations:

1. The players call $\mathscr{F}_{\mathsf{ROT}}^{\mu,\mu+1}$, with Alice acting as sender and Bob as receiver. Let $(\boldsymbol{s_0}, \boldsymbol{s_1})$ denote Alice's output, and let $e \in \mathbb{Z}_2^{\mu+1}$ and $\boldsymbol{s} \leftarrow \left(\boldsymbol{s_{e[i]}}[i]\right)_{i\leq\mu}$ denote Bob's output.
2. The players call $\mathscr{F}_{\mathsf{ROT}}^{\mu,2^{\lambda+1}}$, with Alice acting as sender and Bob as receiver. Let $(\boldsymbol{t_0}, \boldsymbol{t_1})$ denote Alice's output, and let $d \in \mathbb{Z}_2^{2^{\lambda+1}}$ and $\boldsymbol{t} \leftarrow \left(\boldsymbol{t_{d[i]}}[i]\right)_{i\leq\mu}$ denote Bob's output.
3. The players call $\mathscr{F}_{\mathsf{ROT}}^{\mu,2^{\lambda+1}}$, with Alice acting as receiver and Bob as sender. Let $(\boldsymbol{u_0}, \boldsymbol{u_1})$ denote Bob's output, and let $e \in \mathbb{Z}_2^{2^{\lambda+1}}$ and $\boldsymbol{u} \leftarrow \left(\boldsymbol{u_{e[i]}}[i]\right)_{i\leq\mu}$ denote Alice's output.

Product-Sharing(n)**:** Alice picks $(x, a) \leftarrow_R (\mathbb{Z}_2^{2^n-2})^2$, and Bob picks $y \leftarrow_R \mathbb{Z}_2^{2^n-1}$. The players call $\mathscr{F}_{\mathsf{OT}}^{2^n-2,1}$ on input $(a[i], a[i] \oplus x[i])_{i\leq n}$ for Alice and y for Bob. Let b denote Bob's output. Alice outputs (x, a) and Bob outputs (y, b).

Fig. 7: Preprocessing Protocol for Secure Comparison

Theorem 7. *The protocol Π_{SC} securely implements $\mathscr{F}_{\mathsf{SC}}$ when calls to $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ in Π_{SC} are replaced by executions of $\Pi_{\mathsf{SC\text{-}prep}}$ in the $(\mathscr{F}_{\mathsf{ROT}}, \mathscr{F}_{\mathsf{OT}})$-hybrid model.*

4.5 Efficiency Analysis

We estimate both the asymptotic complexity and the concrete efficiency of our protocols; however, we focus only on the amortized setting here, which is more meaningful in most applications. In all our numerical applications, we set the security parameter κ to 128. We consider two protocols in our estimations: the protocol Π_{SC} described above, which performs a logarithmic number of size reduction steps, and constant-round variant of Π_{SC} where a constant number c of size reduction steps are performed, and the final comparison is done with [KSS09].

Communication Complexity. For any integer t, let Π_{ET}^t be the protocol Π_{ET} for t-bit inputs. The full protocol involves μ executions of $\Pi_{\mathsf{ET}}^{\lambda}$, $\Pi_{\mathsf{ET}}^{|\mu+1|}$, $\mathsf{OT}_{|\mu+1|}$, 2μ executions of $\mathsf{OT}_{\lambda+1}$, and a secure comparison on $(\lambda+1)$-bit inputs. For small values $\ell(\kappa) = o(\kappa^2)$ of ℓ, this transmits $O(\frac{\kappa\ell}{\log\kappa})$ bits. Asymptotically, when $\ell \gg \kappa^2$, an equality test transmits $O(\kappa^2/\log\kappa)$ bits independently of the size of ℓ, as the size of the strings to be compared can be reduced to κ bits while statistically preserving their

Table 3: Amortized communication of ℓ-bit SC

	SC_1		SC_2		[KSS09]	
ℓ	comm.	rounds	comm.	round	comm.	round
			Preprocessing Phase			
4	1544 bits	2 rounds	-		1032 bits	1 round
8[1]	3572 bits	2 rounds	-		3088 bits	1 round
16[2]	8396 bits	2 rounds	-		6176 bits	1 round
32[3]	15120 bits	3 rounds	12568 bits	3 rounds	12352 bits	1 round
64[4]	31388 bits	3 rounds	28872 bits	3 rounds	24704 bits	1 round
128[5]	52121 bits	3 rounds	48031 bits	3 rounds	49408 bits	1 round
			Online Phase			
4	30 bits	2 rounds	-		1540 bits	2 rounds
8	162 bits	6 rounds	-		3080 bits	2 rounds
16	308 bits	6 rounds	-		6160 bits	2 rounds
32	530 bits	12 rounds	3125 bits	7 rounds	12320 bits	2 rounds
64	1120 bits	12 rounds	4138 bits	7 rounds	24640 bits	2 rounds
128	2101 bits	12 rounds	5801 bits	7 rounds	49280 bits	2 rounds

[1] $\mu_1 = 4, \lambda_1 = 2$
[2] $\mu_1 = 6, \lambda_1 = 3$ reduces the input size to $\ell = 5$, then $\mu_2 = 3, \lambda_2 = 2$
[3] $\mu_1 = 6, \lambda_1 = 6$ reduces the input size to $\ell = 7$, then $\mu_2 = 4, \lambda_2 = 2$
[4] $\mu_1 = 10, \lambda_1 = 7$ reduces the input size to $\ell = 8$, then $\mu_2 = 4, \lambda_2 = 2$
[5] $\mu_1 = 15, \lambda_1 = 9$ reduces the input size to $\ell = 10$, then $\mu_2 = 4, \lambda_2 = 2$

equality. In the constant-round setting, this gives a $O(c \cdot \log^* \kappa)$-round protocol, and in the logarithmic-round setting, setting $c = O(\log \ell / \log\log \kappa)$, the round complexity becomes $O(\log \ell \cdot \log^* \kappa / \log\log \kappa)$. This leads to protocols with respective asymptotic communication

$$O\left(c\left(\frac{\ell \log \kappa}{\kappa}\right)^{\frac{1}{c+1}} \frac{\kappa^2}{\log \kappa} + \ell\right) = O(\ell), \qquad O\left(\frac{\kappa^2 \log \ell}{\log \kappa \log\log \kappa} + \ell\right) = O(\ell).$$

Concrete Efficiency. We now estimate the efficiency of our secure comparison protocol, in an amortized setting (using oblivious transfer extension). We use the equality test of the previous section, with short-string correlated oblivious transfer extension [KK13, ALSZ13]. The results are given in Table 3; they correspond to the results obtained using the optimal block-decomposition of the inputs. The notes in Table 3 indicate the optimal values of λ_i, μ_i for each value of ℓ. SC_1 denotes the protocol obtained by recursively applying the reduction protocol, until the inputs are small enough so that the small-string secure comparison protocol becomes efficient. We set the thresholds of both the secure comparison protocol and the equality-tests subprotocol to 4. If one is willing to reduce the round complexity of the protocol at the cost of transmitting more bits, the threshold can be increased. SC_2 denotes the protocol obtained by performing a single reduction step, then using the garbled circuit approach of [KSS09] to complete the protocol. This approach is interesting only for $\ell > 16$, as for $\ell \leq 32$, the optimal values for λ are equal to 4 or less, hence applying the small-string equality test directly is more efficient than using garbled circuits (and has the same round complexity). As one can see from Table 3, the communication in SC_1 is reduced by 30 to 45% compared to the garbled circuit approach overall, and by $97 - 99\%$ during the online phase. SC_2 has comparable overall efficiency, but offers slightly less communication improvements during the online phase, in exchange for a better round complexity. Note that as for equality tests, the computational complexity of both [KSS09] and our protocols are proportional to their communication, hence our constructions improve upon [KSS09] regarding computation by factors similar to those listed in Table 3.

Acknowledgements. We would like to thank Thomas Schneider, for pointing out inaccuracies in our cost estimations for the garbled circuit-based constructions of equality tests and secure comparison.

References

ABZS13. M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *NDSS 2013*, February 2013.

ACM+13. A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. V. Vyve. Securely solving simple combinatorial graph problems. In *FC 2013*, *LNCS* 7859, pages 239–257. Springer, April 2013.

AIKW13. B. Applebaum, Y. Ishai, E. Kushilevitz, and B. Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 166–184. Springer, August 2013.

ALSZ13. G. Asharov, Y. Lindell, T. Schneider, and M. Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM CCS 13*, pages 535–548. ACM Press, November 2013.

ARL+13. E. Ayday, J. L. Raisaro, M. Laren, P. Jack, J. Fellay, and J.-P. Hubaux. Privacy-preserving computation of disease risk by using genomic, clinical, and environmental data. In *Proceedings of USENIX Security Workshop on Health Information Technologies (HealthTech" 13)*, number EPFL-CONF-187118, 2013.

Bea96. D. Beaver. Correlated pseudorandomness and the complexity of private computations. In *28th ACM STOC*, pages 479–488. ACM Press, May 1996.

BK04. I. F. Blake and V. Kolesnikov. Strong conditional oblivious transfer and computing on intervals. In *ASIACRYPT 2004*, *LNCS* 3329, pages 515–529. Springer, December 2004.

BLLP14. D. Bogdanov, P. Laud, S. Laur, and P. Pullonen. From input private to universally composable secure multiparty computation primitives. Cryptology ePrint Archive, Report 2014/201, 2014. `http://eprint.iacr.org/2014/201`.

BPTG14. R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. Cryptology ePrint Archive, Report 2014/331, 2014. `http://eprint.iacr.org/2014/331`.

BS15. M. Blanton and S. Saraph. Oblivious maximum bipartite matching size algorithm with applications to secure fingerprint identification. LNCS, pages 384–406. Springer, 2015.

Can01. R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

CC15. W.-T. Chu and F.-C. Chang. A privacy-preserving bipartite graph matching framework for multimedia analysis and retrieval. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 243–250. ACM, 2015.

Cd10a. O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In *SCN 10*, *LNCS* 6280, pages 182–199. Springer, September 2010.

Cd10b. O. Catrina and S. de Hoogh. Secure multiparty linear programming using fixed-point arithmetic. In *ESORICS 2010*, *LNCS* 6345, pages 134–150. Springer, September 2010.

CDN01. R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *EUROCRYPT 2001*, *LNCS* 2045, pages 280–299. Springer, May 2001.

CKP07. R. Cramer, E. Kiltz, and C. Padró. A note on secure computation of the Moore-Penrose pseudoinverse and its application to secure linear algebra. In *CRYPTO 2007*, *LNCS* 4622, pages 613–630. Springer, August 2007.

CPP16. G. Couteau, T. Peters, and D. Pointcheval. Encryption switching protocols. to appear in the proceedings of CRYPTO, 2016. `http://eprint.iacr.org/2015/990`.

DFK+06. I. Damgård, M. Fitzi, E. Kiltz, J. B. Nielsen, and T. Toft. Unconditionally secure constant-rounds multi-party computation for equality, comparison, bits and exponentiation. In *TCC 2006*, *LNCS* 3876, pages 285–304. Springer, March 2006.

DGK07. I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP 07*, *LNCS* 4586, pages 416–430. Springer, July 2007.

DGK09. I. Damgard, M. Geisler, and M. Kroigard. A correction to'efficient and secure comparison for on-line auctions'. *International Journal of Applied Cryptography*, 1(4):323–324, 2009.

DJ01. I. Damgård and M. Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In *PKC 2001*, *LNCS* 1992, pages 119–136. Springer, February 2001.

EGL82. S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *CRYPTO'82*, pages 205–210. Plenum Press, New York, USA, 1982.

EVTL12. Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *Information Forensics and Security, IEEE Transactions on*, 7(3):1053–1066, 2012.

GHJR15. C. Gentry, S. Halevi, C. S. Jutla, and M. Raykova. Private database access with HE-over-ORAM architecture. In *ACNS 15*, LNCS, pages 172–191. Springer, 2015.

GMW87. O. Goldreich, S. Micali, and A. Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *CRYPTO'86*, *LNCS* 263, pages 171–185. Springer, August 1987.

Gol87. O. Goldreich. Towards a theory of software protection and simulation by oblivious RAMs. In *19th ACM STOC*, pages 182–194. ACM Press, May 1987.

Goo10. M. T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *21st SODA*, pages 1262–1277. ACM-SIAM, January 2010.

Goo14. M. T. Goodrich. Zig-zag sort: a simple deterministic data-oblivious sorting algorithm running in $O(n \log n)$ time. In *46th ACM STOC*, pages 684–693. ACM Press, 2014.

GSV07. J. A. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *PKC 2007*, *LNCS* 4450, pages 330–342. Springer, April 2007.

HEK12. Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS 2012*, February 2012.

HICT14. K. Hamada, D. Ikarashi, K. Chida, and K. Takahashi. Oblivious radix sort: An efficient sorting algorithm for practical secure multi-party computation. Cryptology ePrint Archive, Report 2014/121, 2014. `http://eprint.iacr.org/2014/121`.

HT14. C. Hazay and T. Toft. Computationally secure pattern matching in the presence of malicious adversaries. *Journal of Cryptology*, 27(2):358–395, April 2014.

IKNP03. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, *LNCS* 2729, pages 145–161. Springer, August 2003.

Kil88. J. Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

KK13. V. Kolesnikov and R. Kumaresan. Improved OT extension for transferring short secrets. In *CRYPTO 2013, Part II*, *LNCS* 8043, pages 54–70. Springer, August 2013.

KS08. V. Kolesnikov and T. Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP 2008, Part II*, *LNCS* 5126, pages 486–498. Springer, July 2008.

KSS09. V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS 09*, *LNCS* 5888, pages 1–20. Springer, December 2009.

Lau15. P. Laud. A private lookup protocol with low online complexity for secure multiparty computation. In *ICICS 14*, LNCS, pages 143–157. Springer, 2015.

LLY+16. P. Li, T. Li, Z.-A. Yao, C.-M. Tang, and J. Li. Privacy-preserving outsourcing of image feature extraction in cloud computing. *Soft Computing*, pages 1–11, 2016.

LT13. H. Lipmaa and T. Toft. Secure equality and greater-than tests with sublinear online complexity. In *ICALP 2013, Part II*, *LNCS* 7966, pages 645–656. Springer, July 2013.

NIIO14. T. Nishide, M. Iwamoto, A. Iwasaki, and K. Ohta. Secure (m+ 1) st-price auction with automatic tie-break. In *Trusted Systems*, pages 422–437. Springer, 2014.

NO07. T. Nishide and K. Ohta. Multiparty computation for interval, equality, and comparison without bit-decomposition protocol. In *PKC 2007*, *LNCS* 4450, pages 343–360. Springer, April 2007.

NP01. M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *12th SODA*, pages 448–457. ACM-SIAM, January 2001.

Pai99. P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT'99*, *LNCS* 1592, pages 223–238. Springer, May 1999.

Rab81. M. Rabin. How to exchange secrets by oblivious transfer. *Technical Report TR-81, Harvard University,*, 1981.

RPV+14. Y. Rahulamathavan, R. C.-W. Phan, S. Veluru, K. Cumanan, and M. Rajarajan. Privacy-preserving multi-class support vector machine for outsourcing the data classification in cloud. *IEEE Transactions on Dependable and Secure Computing*, 11(5):467–479, 2014.

SJB14. B. K. Samanthula, W. Jiang, and E. Bertino. Lightweight and secure two-party range queries over outsourced encrypted databases. *arXiv:1401.3768*, 2014.

SSW10. A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC 09*, *LNCS* 5984, pages 229–244. Springer, December 2010.

Tof09. T. Toft. Solving linear programs using multiparty computation. In *FC 2009*, *LNCS* 5628, pages 90–107. Springer, February 2009.

Tof11. T. Toft. Sub-linear, secure comparison with two non-colluding parties. In *PKC 2011*, *LNCS* 6571, pages 174–191. Springer, March 2011.

Veu12. T. Veugen. Improving the dgk comparison protocol. In *Information Forensics and Security (WIFS), 2012 IEEE International Workshop on*, pages 49–54. IEEE, 2012.

WFNL15. D. J. Wu, T. Feng, M. Naehrig, and K. Lauter. Privately evaluating decision trees and random forests. Cryptology ePrint Archive, Report 2015/386, 2015. `http://eprint.iacr.org/2015/386`.

Yao86. A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.

YY12. C.-H. Yu and B.-Y. Yang. Probabilistically correct secure arithmetic computation for modular conversion, zero test, comparison, MOD and exponentiation. In *SCN 12*, *LNCS* 7485, pages 426–444. Springer, September 2012.

ZRE15. S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. LNCS, pages 220–250. Springer, 2015.

A Batch ET from Additively Homomorphic Encryption

In this section, we present a batch protocol to efficiently perform simultaneous equality tests. Unlike the other protocols of this article, this construction assumes an additively homomorphic encryption scheme, with a few additional properties. Our protocol share some similarities with the equality test protocol of [GHJR15] (which relies on ciphertext packing to amortize the communication of equality tests), and in fact matches the communication complexity of [GHJR15], which has to our knowledge the best communication complexity among existing works. However, contrary to [GHJR15], we do

not need somewhat homomorphic encryption; our protocol can be instantiated with e.g. factorization-based additively homomorphic cryptosystems such as the Paillier scheme [Pai99] or the Damgard-Jurik scheme [DJ01]. For concrete parameters, the amortized communication strongly improves upon every prior ET protocol we know of, including the protocol described Section 3.

A.1 Encryption Scheme

Definition 8. *(Encryption Scheme) An IND-CPA encryption scheme is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ *such that:*

- $\mathsf{Setup}(1^\kappa)$ *outputs a key-pair* $(\mathsf{pk}, \mathsf{sk})$*;* pk *implicitly defines a plaintext space* $\mathscr{M}$ *and a ciphertext space* $\mathscr{C}$*.*
- $\mathsf{Enc}(\mathsf{pk}, m)$*, on input* pk *and a plaintext* $m \in \mathscr{M}$*, outputs a ciphertext* $c \in \mathscr{C}$*.*
- $\mathsf{Dec}(\mathsf{sk}, c)$*, on input* sk *and a ciphertext* $c \in \mathscr{C}$*, deterministically outputs a plaintext* $m' \in \mathscr{M}$*.*

In addition, an IND-CPA encryption scheme satisfies the properties of correctness *and* IND-CPA security*, defined below.*

The *correctness* states that decryption is the reverse operation of encryption: for any $(\mathsf{pk}, \mathsf{sk}) \leftarrow_R \mathsf{Setup}(1^\kappa)$, for any $m \in \mathscr{M}$ and any $c \leftarrow_R \mathsf{Enc}(\mathsf{pk}, m)$, $\mathsf{Dec}(\mathsf{sk}, c) = m$. The *IND-CPA security* is defined by considering the following game between an adversary and a challenger:

- The challenger picks $(\mathsf{pk}, \mathsf{sk}) \leftarrow_R \mathsf{Setup}(1^\kappa)$ and sends pk to the adversary.
- The adversary sends $(m_0, m_1) \leftarrow_R \mathscr{M}^2$ to the challenger.
- The challenger picks $b \leftarrow_R \{0, 1\}$ and sends $c \leftarrow_R \mathsf{Enc}(\mathsf{pk}, m_b)$ to the challenger.
- The challenger outputs a guess b' and wins the game if $b' = b$.

An encryption scheme is IND-CPA secure if no polynomial-time adversary can win the game with non-negligible advantage over the random guess.

Additively Homomorphic Encryption Scheme. An encryption scheme is additively homomorpic if there is a law $\boxplus : \mathscr{C}^2 \mapsto \mathscr{C}$ such that for any $(m_0, m_1) \in \mathscr{M}^2$, for any $(c_0, c_1) \leftarrow_R (\mathsf{Enc}(\mathsf{pk}, m_0), \mathsf{Enc}(\mathsf{pk}, m_1))$, $\mathsf{Dec}(\mathsf{sk}, c_0 \boxplus c_1) = m_0 + m_1$. Note that this trivially implies than one can add a constant value to a ciphertext (by first encrypting it and then using $\boxplus$); one can also see that via a square-and-multiply algorithm, given an encryption of some m and an integer λ, one can compute an encryption of λm. We will denote $\bullet$ this external multiplication.

Randomizable Encryption Scheme. A randomizable encryption scheme is an encryption scheme with an additional algorithm Rand which, on input pk and an encryption of some plaintext m, outputs a ciphertext taken uniformly at random in the distribution $\{\mathsf{Enc}(\mathsf{pk}, m)\}$ of encryptions of m.

Expendable Plaintext Space. In our protocol, we require the message space to be of the form $\mathbb{Z}_P$, for some integer $P = 2^{\mathrm{poly}(\kappa)}$. In addition the plaintext space must be *expendable*, in the sense that one can specify a threshold T when calling $\mathsf{Setup}(1^\kappa, T)$, so that the message space $\mathscr{M} = \mathbb{Z}_P$ it specifies is of size $P \geq T$. For example, for the Paillier encryption scheme and its variants, this would simply correspond to taking the modulus bigger than T.

A.2 Batch Equality Test

We let $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Dec})$ denote a randomizable additively homomorphic encryption scheme with expendable plaintext space. Let n be the number of equality tests to be performed. As there is no possible confusion, we write $\mathsf{Enc}(m)$ for $\mathsf{Enc}(\mathsf{pk}, m)$.

Inputs: n pairs of ℓ-bit strings $(x^{(i)}, y^{(i)})_{i \leq n}$.
Outputs: n bits $(b_i^A)_{i \leq n}$ for Alice, and n bits $(b_i^B)_{i \leq n}$ for Bob, such that for all $i \leq n$, $b_i^A \oplus b_i^B = [x^{(i)} \leq y^{(i)}]$.

Batch reduction: In this step, Alice and Bob rely on the additively homomorphic encryption scheme to compute shares of the Hamming distances between each $x^{(i)}, y^{(i)}$, modulo coprime integers p_i.

- Let $(p_0, \cdots p_{n-1})$ be the n smallest pairwise coprime numbers such that $p_0 > \ell$; let $M \leftarrow \prod_i p_i$. Alice calls $\mathsf{Setup}(1^\kappa, 2^{\kappa+2|M|+2})$ and gets $(\mathsf{pk}, \mathsf{sk})$; pk implicitely defines a plaintext space $\mathbb{Z}_P$ of size $P \geq 2^{\kappa+2|M|+2}$. For $j = 0$ to $\ell - 1$, let $x_j \in \mathbb{Z}_M$ (resp. y_j) be the smallest integer satisfying $x_j = x^{(i)}[j] \bmod p_i$ (resp. $y_j = y^{(i)}[j] \bmod p_i$) for every $i \leq n-1$. Alice sends $c_j \leftarrow_R \mathsf{Enc}(x_j)$ for $j = 0$ to $\ell - 1$ to Bob.
- For $j = 0$ to $\ell - 1$, Bob picks $r_j \leftarrow_R \mathbb{Z}_{2^{\kappa+2}M^2}$, computes and sends $c'_j \leftarrow_R \mathsf{Rand}(\mathsf{pk}, y_j \bullet c_j \boxplus r_j)$ to Alice, who decrypts all the ciphertexts to get some values s_j.
- For $j = 0$ to $\ell - 1$, Alice sets $\sigma_j \leftarrow s_j \bmod M$ and Bob sets $\rho_j \leftarrow r_j \bmod M$. Note that it holds that for all $(i, j) \in [n-1] \times [\ell - 1]$,

$$2(\rho_j - \sigma_j) + x^{(i)}[j] + y^{(i)}[j] = x^{(i)}[j] \oplus y^{(i)}[j] \bmod p_i$$

  Hence, $(-2\sigma_j + x^{(i)}[j] \bmod p_i)$ and $(2\rho_i + y^{(i)}[j] \bmod p_i)$ form shares of the bits of $x^{(i)} \oplus y^{(i)}$ modulo p_i.
- Alice computes $\alpha_i \leftarrow \sum_{j=0}^{\ell-1} -2\sigma_j + x^{(i)}[j] \bmod p_i$ and Bob computes $\beta_i \leftarrow \sum_{j=0}^{\ell-1} -2\rho_j + y^{(i)}[j] \bmod p_i$. Note that as $p_i > \ell$ is greater than the Hamming distance H_d between $x^{(i)}$ and $y^{(i)}$, it holds that $\alpha_i + \beta_i = H_d(x^{(i)}, y^{(i)})$, which is 0 if and only if $x^{(i)} = y^{(i)}$. Hence, seeing from now on α_i and β_i as integers, the problem was reduced to finding whether $\alpha_i = p_i - \beta_i$, which are strings of size $O(\log \ell \log\log \ell)$.

Reduced Equality Test: Alice and Bob perform n ET with respective input size $|p_i|$, on respective inputs $(\alpha_i, p_i - \beta_i)$, to get the n outputs of the protocol.

Note that as for our protocol Section 3, this protocol can be executed on random inputs in a preprocessing phase; the online phase is then essentially the same than our previous ET protocol.

Intuition of the Protocol. The protocol exploits the following observation: given an index $j < \ell$, computing shares of $(x^{(i)}[j] \oplus y^{(i)}[j])_{i \leq n}$ (modulo various coprime numbers) can be reduced to performing a single multiplication protocol modulo $M = \prod_i p_i$. This protocol is performed over the integers by using an additively homomorphic scheme of sufficiently large plaintext space, the resulting shares masking statistically the result over the integer. The players then get all the shares of the $(x^{(i)}[j] \oplus y^{(i)}[j])_{i \leq n}$ by reducing there shares modulo M and using the chinese remainder theorem on there shares. This reduces n equality tests on ℓ-bit strings to n equality tests on strings of sizes ranging from $|p_1 + 1|$ to $|p_n + 1|$. As this method does not allow to reduce further the size of the inputs, n OT-based equality tests are then called in parallel on the reduced inputs.

Communication. The batch reduction involves 2ℓ ciphertexts, hence a total of $2\ell|\mathscr{C}|$ bits. Under the extended Riemann hypothesis, the nth prime number larger than ℓ is of size $O(\log(\ell + n \log n))$, hence $M = O(n \log(\ell + n \log n))$. Under this assumption, the n reduced equality tests transmit $O(n\kappa \log(\ell + n \log n)/\log \kappa)$ bits.

Most additively homomorphic that satisfy our requirements have ciphertexts of size $O(k + \kappa)$ for k-bit inputs with large enough k; taking this condition in account, the amortized communication becomes $O(\ell \log \kappa + \kappa)$ bits.

Concrete Efficiency. We now estimate the concrete efficiency of our protocol, and compare it to our previous solution. We use the Damgard-Jurik generalization [DJ01] of the Paillier encryption scheme, which enjoys better ciphertext over plaintext size ratio as the size of the plaintext space increases. More precisely, the Damgard-Jurik cryptosystem for an RSA modulus N is parametrized with an integer s, so that its plaintext space is $\mathbb{Z}_{N^s}$, and its ciphertext space is $\mathbb{Z}_{N^{s+1}}$. We consider a 2048-bit RSA modulus, as recommended by the NIST standard, and set arbitrarily the number n of parallel ETs to 100 and 1000 respectively. For the oblivious transfers, we use $\kappa = 128$.

For each value of ℓ in the table, s is taken to be the smallest integer such that $s \cdot 2048 \geq 2 \log \mathsf{P_n}(\ell) + 129$, where $\mathsf{P_n}(\ell)$ is the product of the smallest n pairwise coprime numbers, starting with $\ell + 1$. Each ciphertext is of size $(s+1) \cdot 2048$. Table 4 indicates the average number of bits transmitted per ET.

Table 4: Amortized communication of ℓ-bit ET over n executions

	Damgard-Jurik based ET				ET of Section 3	
	$n = 1000$		$n = 100$			
ℓ	length	rounds	length	rounds	length	rounds
Preprocessing Phase						
16	3339 bits	4 rounds	3183 bits	4 rounds	2945 bits	4 rounds
32	4199 bits	4 rounds	4568 bits	4 rounds	5212 bits	4 rounds
64	5913 bits	4 rounds	7275 bits	4 rounds	9863 bits	4 rounds
128	9342 bits	4 rounds	12670 bits	4 rounds	20194 bits	4 rounds
Online Phase						
16	96 bits	3 rounds	81 bits	3 rounds	54 bits	3 rounds
32	129 bits	3 rounds	115 bits	3 rounds	88 bits	3 rounds
64	193 bits	3 rounds	181 bits	3 rounds	154 bits	3 rounds
128	321 bits	3 rounds	313 bits	3 rounds	300 bits	3 rounds

The actual value of ℓ has very little influence on s; in fact, $s = 12$ is the optimal parameters for all the values of ℓ that we consider (hence the ciphertexts are of size 26624 bits). With those parameters, n ET on ℓ-bit strings are reduced to n ET on strings of bit-size 6 to 13 (as all the p_i are different, the reduction gives different bit-sizes); experimentally, it turns out that this improves over our OT-base ET for $\ell > 16$.

B Missing Proofs

B.1 Proof of Theorem 6

We prove Theorem 6, introduced in Section 3. We first recall the protocol $\Pi_{\mathsf{ET\text{-}prep}}$, and the functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ that it implements, on Figures 9 and 8.

Functionality $\mathscr{F}_{\mathsf{ET}}$

The functionality runs with two parties, Alice and Bob. Upon receiving (ET, x) from Alice and (ET, y) from Bob, set $\beta \leftarrow 1$ if $x = y$, and $\beta \leftarrow 0$ else. Set $(a, b) \leftarrow_R \langle \beta \rangle_2$. Return a to Alice and b to Bob.

Functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$

The functionality runs with two parties, Alice and Bob.

Size Reduction: Upon receiving (SR, j) from both players, the functionality picks $(x, y) \leftarrow_R (\mathbb{Z}_2^j)^2$ and sets $(\boldsymbol{a}, \boldsymbol{b}) \leftarrow_R \langle x \oplus y \rangle_{j+1}$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs $(x, \boldsymbol{a})$ to Alice and $(y, \boldsymbol{b})$ to Bob.

Product Sharing: Upon receiving (PS, n) from both players, the functionality picks $(x, y) \leftarrow_R (\mathbb{Z}_2^{2^n-2})^2$ and sets $(a, b) \leftarrow_R \langle x * y \rangle_2$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs (x, a) to Alice and (y, b) to Bob.

Fig. 8: Ideal Functionalities for Equality Test and Preprocessing

Security Analysis. Let $\diamond$ denote the composition operator; let Π be the protocol such that $\Pi \diamond \mathscr{F}_{\mathsf{ET\text{-}prep}} = \Pi_{\mathsf{ET}}$. The natural way of proving Theorem 6 would be to prove that $\Pi_{\mathsf{ET\text{-}prep}}$ UC implements $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model; as $\Pi \diamond \mathscr{F}_{\mathsf{ET\text{-}prep}}$ UC implements $\mathscr{F}_{\mathsf{ET}}$, it would follow from the composition theorem of the UC framework that $\Pi \diamond \Pi_{\mathsf{ET\text{-}prep}}$ UC implements $\mathscr{F}_{\mathsf{ET}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model.

However, this approach fails here. The reason is that $\Pi_{\mathsf{ET\text{-}prep}}$ does in fact *not* UC implement $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. To understand the issue, recall that a proof in the UC framework is done by exhibiting a simulator with access to the functionality (here $\mathscr{F}_{\mathsf{ET\text{-}prep}}$), so that the view produced by the simulator

Protocol $\Pi_{\mathsf{ET\text{-}prep}}$

Size-Reduction(ℓ)**:** Alice picks $(x, \boldsymbol{a}) \leftarrow_R \mathbb{Z}_2^\ell \times \mathbb{Z}_{\ell+1}^\ell$, and Bob picks $y \leftarrow_R \mathbb{Z}_2^\ell$. The players call $\mathscr{F}_{\mathsf{OT}}^{\ell,|\ell+1|}$, on input $(x[i] - \boldsymbol{a}[i] \bmod \ell+1, 1 - \boldsymbol{a}[i] - x[i] \bmod \ell+1)_{i\leq\ell}$ for Alice and y for Bob. Let $\boldsymbol{b}$ denote Bob's output. Alice outputs $(x, \boldsymbol{a})$ and Bob outputs $(y, \boldsymbol{b})$.

Product-Sharing(n)**:** Alice picks $(x, a) \leftarrow_R (\mathbb{Z}_2^{2^n-2})^2$, and Bob picks $y \leftarrow_R \mathbb{Z}_2^{2^n-2}$. The players call $\mathscr{F}_{\mathsf{OT}}^{2^n-2,2}$ on input $(a[i], a[i] \oplus x[i])_{i\leq 2^n-2}$ for Alice and y for Bob. Let b denote Bob's output. Alice outputs (x, a) and Bob outputs (y, b).

Fig. 9: Preprocessing Protocol for Equality Test

together with the outputs of the corrupted parties are indistinguishable from the view and the outputs in a real execution of the protocol. The simulator extracts the inputs of the corrupted parties, queries the ideal functionality on those inputs, and somehow forces the corrupted players to obtain the same outputs. Observe now that in the protocol $\Pi_{\mathsf{ET\text{-}prep}}$, parts of the outputs of both Alice and Bob are picked by the players themselves, independently of the behaviour of their opponent. Therefore, no simulator can possibly ensure that the corrupted player will compute the same output than what was returned by $\mathscr{F}_{\mathsf{ET\text{-}prep}}$. This is a bit counterintuitive, as the protocol $\Pi_{\mathsf{ET\text{-}prep}}$ clearly does "exactly what $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ does" when the players are semi-honest. We could solve this by adding some resharing step in $\Pi_{\mathsf{ET\text{-}prep}}$, but this would noticeably increase the communication of our protocol. Fortunately, this exact issue was handled in great details in [BLLP14]. Therefore, we start by recalling the results of [BLLP14], and use them to complete the proof of Theorem 6.

Input Private Protocols. The authors of [BLLP14] extend the universal composability framework, by defining the notion of *input-privacy*. Informally, a protocol is input-private if there is a simulator (with access to the corresponding ideal functionality) which produces a view indistinguishable from an execution of the protocol for any environment *that completely ignores the output of the protocol*. We refer to [BLLP14, Section 4] for a formal definition. The core result is a composition theorem, which states that an *ordered* composition $\Pi_\diamond$ of an input-private protocol Π_{ip} and a UC secure protocol Π_{UC} is a UC secure protocol (the model considered is that of static, passive corruption, as in the present paper). The term "ordered" refers to a restricted class of composition, in which there is a one-way communication from Π_{ip} to Π_{UC}. An additional requirement is that the outputs of $\Pi_\diamond$ must all come from Π_{UC} – in other words, Π_{ip} does not output any value in the composed protocol, but only intermediate random values, perfectly independent of the outputs. Intuitively, this result is tailored to protocols Π_{ip} that return random shares of some output, as the view of a corrupted player can be simulated without the output.

Security. The protocol $\Pi \diamond \Pi_{\mathsf{ET\text{-}prep}}$ immediatly satisfies the constraint of ordered composition with predicable output specified in [BLLP14] (outputs from $\Pi_{\mathsf{ET\text{-}prep}}$ are used in Π_{ET}, but there is no data dependency in the other direction). The functionality $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ can be virtually seen as a functionality that does not produce outputs, but only stores the result of some computation (this storage being represented by secret shares in the real protocol). The input-privacy of $\Pi_{\mathsf{ET\text{-}prep}}$ with respect to $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model is straightforward: it is trivial to see that the protocol is correct, and no messages are exchanged between the players during the protocol, which consists entirely in calls made to $\mathscr{F}_{\mathsf{OT}}$. Therefore, the simulator simply runs a local copy of $\mathscr{F}_{\mathsf{OT}}$ and answers honestly to the queries of the players. The view of any environment that ignores the output of the protocol contains only the responses of $\mathscr{F}_{\mathsf{OT}}$ to the corrupted player, hence the simulation is trivially perfect.

By the input-privacy of $\Pi_{\mathsf{ET\text{-}prep}}$, and the ordered composition with predictable output of $\Pi \diamond \Pi_{\mathsf{ET\text{-}prep}}$, as $\Pi \diamond \mathscr{F}_{\mathsf{ET\text{-}prep}}$ UC implements $\mathscr{F}_{\mathsf{ET}}$, the composition theorem mentioned above (Theorem 2 from [BLLP14, Section 7.2]) allows to conclude that $\Pi \diamond \Pi_{\mathsf{ET\text{-}prep}}$ UC implements $\mathscr{F}_{\mathsf{ET}}$ in the $\mathscr{F}_{\mathsf{OT}}$-hybrid model.

B.2 Proof of Lemma 3

We recall below the Lemma 3, introduced in Section 4, and prove it. Let (x, y) be two bitstrings of length k. Let $t \leftarrow 2^{k+1}$. Let $(x_A, x_B) \leftarrow_R \langle x \rangle_t$ and $(y_A, y_B) \leftarrow_R \langle y \rangle_t$; that is, (x_A, x_B) (resp. (y_1, y_B)) are random shares of x (resp. y) over $\mathbb{Z}_t$. Let $(z_A, z_B) \leftarrow (y_A - x_A \bmod t, y_B - x_B \bmod t)$.

Lemma 9. *If $x \neq y$, then it holds that*

$$[x \leq y] = b_A \oplus b_B \oplus [w_A \leq t/2 - w_B],$$

where $b_A = [z_A < t/2]$, $b_B = [z_B < t/2]$, $w_A = z_A \bmod t/2$, and $w_B = z_B \bmod t/2$.

Proof. To prove Lemma 3, we first observe that $[x \leq y] = [(y - x \bmod t) \leq t/2]$. Indeed, (x, y) are k-bit strings, hence $x < 2^k = t/2$, and $y < t/2$. Therefore, if $x \leq y$, then $(y - x \bmod t) = y - x$ belongs to $[0, t/2]$, which implies that $(y - x \bmod t) \leq t/2$; conversely, if $x > y$, then $[y - x \bmod t] = y - x + t$ belongs to $[t/2, t]$, which implies that $[y - x \bmod t] > t/2$. By this observation, to prove Lemma 3, it suffices to show that, when $x \neq y$,

$$[(y - x \bmod t) \leq t/2] = b_A \oplus b_B \oplus [w_A \leq t/2 - w_B].$$

Toward proving Lemma 3, we distinguish four complementary cases.

Case 1: $z_A < t/2$, and $z_B < t/2$ (therefore, $b_A = b_B = 1$). In this situation, it holds that $(z_A + z_B \bmod t) = z_A + z_B$ (that is, no modulo reduction occurs when summing z_A and z_B modulo t), and that $z_A = w_A$, $z_B = w_B$. Therefore,

$$\begin{aligned}
[(y - x \bmod t) \leq t/2] &= [(z_A + z_B \bmod t) \leq t/2] \\
&= [z_A + z_B \leq t/2] \\
&= [w_A \leq t/2 - w_B] \\
&= [w_A \leq t/2 - w_B] \oplus b_A \oplus b_B.
\end{aligned}$$

Case 2: $z_A \geq t/2$, and $z_B \geq t/2$ (therefore, $b_A = b_B = 0$). In this situation, it holds that $(z_A + z_B \bmod t) = z_A + z_B - t$ (a modulo reduction occurs when summing z_A and z_B modulo t), and that $z_A = w_A + t/2$, $z_B = w_B + t/2$. Therefore,

$$\begin{aligned}
[(y - x \bmod t) \leq t/2] &= [(z_A + z_B \bmod t) \leq t/2] \\
&= [z_A + z_B - t \leq t/2] \\
&= [w_A + t/2 + w_B + t/2 - t \leq t/2 - w_B] \\
&= [w_A \leq t/2 - w_B] \\
&= [w_A \leq t/2 - w_B] \oplus b_A \oplus b_B.
\end{aligned}$$

Case 3: $z_A < t/2$, and $z_B \geq t/2$ (therefore, $b_A = 0$ and $b_B = 1$). Let us show that in this situation,

$$[(y - x \bmod t) \leq t/2] = [(z_A + z_B \bmod t) \leq t/2] = [z_A + z_B > t].$$

Indeed, observe that as $z_A < t/2$ and $z_B \geq t/2$, it holds that $(z_A + z_B \bmod t) \leq t/2$ if and only if a modulo reduction occurs when computing $z_A + z_B \bmod p$ – that is, if and only if $[z_A + z_B \geq p]$. In addition, observe that $z_A + z_B = t$ if and only if $y_A - x_A + y_B - x_B = t$, if and only if $x - y = 0 \bmod t$. Therefore, as we assumed $x \neq y$ (which implies $x \neq y \bmod t$ as $x < t/2$ and $y < t/2$), this situation cannot happen, so $[z_A + z_B \geq p] = [z_A + z_B > p]$. From this, we get

$$\begin{aligned}
[(y - x \bmod t) \leq t/2] &= [(z_A + z_B \bmod t) \leq t/2] \\
&= [z_A + z_B > t] \\
&= 1 \oplus [z_A + z_B \leq t] \text{ as } 1 \oplus b \text{ is the logical negation of } b \\
&= 1 \oplus [w_A + (w_B + t/2) \leq t] \\
&= 1 \oplus [w_A \leq t/2 - w_B] \\
&= [w_A \leq t/2 - w_B] \oplus b_A \oplus b_B.
\end{aligned}$$

Functionality $\mathscr{F}_{\mathsf{SC}}$

The functionality runs with two parties, Alice and Bob. Upon receiving (SC, x) from Alice and (SC, y) from Bob, set $\beta \leftarrow 1$ if $x \leq y$, and $\beta \leftarrow 0$ else. Set $(a, b) \leftarrow_R \langle \beta \rangle_2$. Return a to Alice and b to Bob.

Functionality $\mathscr{F}_{\mathsf{shrink}}$

The functionality runs with two parties, Alice and Bob. Upon receiving $(\mathsf{shrink}, \ell, \lambda, x)$ from Alice and $(\mathsf{shrink}, \ell, \lambda, y)$ from Bob, where λ divides ℓ and (x, y) are ℓ-bit long, it divides the bits of x and y into ℓ/λ consecutive blocks of λ bit, and computes $\hat{x} \in \{0,1\}^\lambda$ (resp. $\hat{y} \in \{0,1\}^\lambda$) as the first block of x (resp. y) on which x and y differ. Then, it computes $(\hat{x}_A, \hat{x}_B) \leftarrow_R \langle \hat{x} \rangle_{2^{\lambda+1}}$, $(\hat{y}_A, \hat{y}_B) \leftarrow_R \langle \hat{y} \rangle_{2^{\lambda+1}}$, and returns $(\hat{x}_A, \hat{y}_A)$ to Alice and $(\hat{x}_B, \hat{y}_B)$ to Bob.

Functionality $\mathscr{F}_{\mathsf{SC\text{-}prep}}$

The functionality runs with two parties, Alice and Bob.

Size Reduction: Upon receiving $(\mathsf{SR}, \lambda, \mu)$ from both players, the functionality picks $(\boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u_0}, \boldsymbol{u_1}) \leftarrow_R (\mathbb{Z}_{\mu+1}^{\mu})^2 \times (\mathbb{Z}_{2^{\lambda+1}}^{\mu})^4$ and $(c, d, e) \leftarrow_R (\mathbb{Z}_2^{\mu})^3$. $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ sets

$$(\boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u}) \leftarrow \left(\left(\boldsymbol{s_{c[i]}}[i]\right)_{i \leq \mu}, \left(\boldsymbol{t_{d[i]}}[i]\right)_{i \leq \mu}, \left(\boldsymbol{u_{e[i]}}[i]\right)_{i \leq \mu} \right)$$

It outputs $(e, \boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u})$ to Alice and $(c, d, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u_0}, \boldsymbol{u_1})$ to Bob.

Product Sharing: Upon receiving (PS, n) from both players, the functionality picks $(\rho, \sigma) \leftarrow_R (\mathbb{Z}_2^{2^n-1})^2$ and sets $(a, b) \leftarrow_R \langle \rho * \sigma \rangle_2$. $\mathscr{F}_{\mathsf{ET\text{-}prep}}$ outputs (ρ, a) to Alice and (σ, b) to Bob.

Fig. 10: Ideal Functionalities for Secure Comparison and Preprocessing

Case 4: By applying the same demonstration as in Case 3 symmetrically, we get $[(y - x \bmod t) \leq t/2] = [w_A \leq t/2 - w_B] \oplus b_A \oplus b_B$.

This concludes the proof of Lemma 3. We make a last observation: straightforward calculations show that, as $x < t/2$ and $y < t/2$, $x \neq y$ if and only if $w_A \neq w_B$. This implies that it is sufficient to guarantee that the initial inputs are different to ensure that all inputs will remain different through recursive calls to the recursion lemma.

B.3 Security Analysis of Π_{shrink}

We recall Figure 10 the ideal functionalities $\mathscr{F}_{\mathsf{SC}}, \mathscr{F}_{\mathsf{SC\text{-}prep}}, \mathscr{F}_{\mathsf{shrink}}$ given Section 4, and Figure 11 the protocol Π_{shrink}. We now prove the following theorem:

Theorem 5 (Repeated from Section 4)**.** *The protocol Π_{shrink} securely implements $\mathscr{F}_{\mathsf{shrink}}$ in the ($\mathscr{F}_{\mathsf{ET}}$, $\mathscr{F}_{\mathsf{SC\text{-}prep}}$)-hybrid model, with respect to passive corruption of at most one of the players.*

Let us recall our correctness analysis of Subsection 4.3; the simulation will then be rather straightforward. The general idea of the protocol is that to compare two strings, it suffices to divide these strings in blocs, and to compare the first block on which they differ. The purpose of the compression step is for the players to obliviously select this block. The inputs (x, y) are first divided into μ blocks of size λ. At the end of step 1, the players obtain shares (α_j, β_j) (over $\mathbb{Z}_2$) of all the bits $[x_j \neq y_j]$. During step 2, the players compute values (x'_j, y'_j) whose difference modulo $\mu + 1$ is $\sum_{k=1}^{j} [x_k \neq y_k]$. This requires to use some preprocessed material. Let j^* be the first block on which x differs from y. Observe that $\sum_{k=1}^{j} [x_k \neq y_k] = 0$ for $j < j^*$, and $\sum_{k=1}^{j} [x_k \neq y_k] > 0$ afterward. The players perform in step 3 equality tests on the values (x'_j, y'_j). Therefore, they obtain shares of the bits $[x'_i - y'_j = 0]$. Observe that these bits are 1 for $j \leq j^*$, and 0 afterward. From these shares, the players can locally compute shares (γ_j, δ_j) of bits which are 0 for every $j \neq j^*$, and 1 only for $j = j^*$, by xoring pairs of consecutive bits. In step 4 and 5, using preprocessed material again, the players compute shares of $\sum_{j=1}^{\mu} (\gamma_j \oplus \delta_j) x_j = x_{j^*}$ and $\sum_{j=1}^{\mu} (\gamma_j \oplus \delta_j) y_j = y_{j^*}$, which correspond to the target outputs.

With the correctness argument in mind, as the protocols Π_{shrink} and Π_{SC} satisfy ordered composition with predictable outputs, we can apply the same argument as for $\mathscr{F}_{\mathsf{ET\text{-}prep}}$, and show that Π_{SC} UC-securely implements $\mathscr{F}_{\mathsf{SC}}$ when calls to $\mathscr{F}_{\mathsf{shrink}}$ are replaced by executions of Π_{shrink}, by proving that

Protocol Π_{shrink}

Let (ℓ, λ) be two integers such that $\lambda \leq \ell$. Let μ be the smallest integer such that $\lambda\mu \geq \ell$. On input x from Alice and y from Bob, both of size ℓ-bit,

Initialize: The players call $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ on input $(\mathsf{SR}, \lambda, \mu)$ to get outputs $(e, \boldsymbol{s_0}, \boldsymbol{s_1}, \boldsymbol{t_0}, \boldsymbol{t_1}, \boldsymbol{u})$ for Alice and $(c, d, \boldsymbol{s}, \boldsymbol{t}, \boldsymbol{u_0}, \boldsymbol{u_1})$ for Bob.

Compression: Let $(x_j)_{j\leq\mu} \in \mathbb{Z}_{2^\lambda}$ (resp. $(y_j)_{j\leq\mu} \in \mathbb{Z}_{2^\lambda}$) be the decomposition of x (resp. y) into μ blocs of size λ (i.e., $x = \sum_{j=1}^{\mu} x_j 2^{\lambda(j-1)}$ and $y = \sum_{j=1}^{\mu} y_j 2^{\lambda(j-1)}$). The players perform the following operations:

1. (Equality Tests.) For $j = 1$ to μ, the players call $\mathscr{F}_{\mathsf{ET}}$ on inputs (ET, x_j) and (ET, y_j). Let $(\alpha_j, 1 \oplus \beta_j)_{j\leq\mu} \in \mathbb{Z}_2^{2\mu}$ denote their respective outputs (that is, (α_j, β_j) form shares of 0 if $x_j = y_j$, and of 1 otherwise). This step allows Alice and Bob to (obliviously) identify the blocks of x and y which are not equal.
2. (Modulus Change.) Alice picks $\boldsymbol{r} \leftarrow_R \mathbb{Z}_{\mu+1}^{\mu}$. For $j = 1$ to μ, Bob sends $d_j \leftarrow \beta_j \oplus c[j]$ to Alice. For $j = 1$ to μ, Alice sends the pair

$$(a_{j,0}, a_{j,1}) \leftarrow (\boldsymbol{s_{d_j}}[j] + \alpha_j + \boldsymbol{r}[j], \boldsymbol{s_{1-d_j}}[j] + (1 - \alpha_j) + \boldsymbol{r}[j]) \bmod \mu + 1$$

   to Bob. For $j = 1$ to μ, Bob computes $y'_j \leftarrow \sum_{k=1}^{j} a_{k,c[k]} - \boldsymbol{s}[k] \bmod \mu + 1$ and Alice computes $x'_j \leftarrow \sum_{k=1}^{j} \boldsymbol{r}[k] \bmod \mu + 1$. This steps converts Alice and Bob's shares of the $[x_j \neq y_j]$ from shares modulo 2 to shares modulo $\mu + 1$, and computes all partial sums (from 1 to j) of these shares.
3. (Identiying the First Different Block.) For $j = 1$ to μ, the players call $\mathscr{F}_{\mathsf{ET}}$ on inputs (ET, x'_j) and (ET, y'_j). Let $(\alpha'_j, 1 \oplus \beta'_j)_{j\leq\mu} \in \mathbb{Z}_2^{2\mu}$ denote their respective outputs (that is, (α'_j, β'_j) form shares of 0 if $x'_j = y'_j$, and of 1 otherwise) and $(\alpha'_0, \beta'_0) \leftarrow (0, 0)$. For $j = 1$ to μ_i, Alice sets $\gamma_j \leftarrow \alpha_{j-1} \oplus \alpha_j$ and Bob sets $\delta_j \leftarrow \beta_{j-1} \oplus \beta_j$. The following steps 4 and 5 are executed in parrallel:
4. (Selecting the First Different Block – Alice's Step.) Alice picks $\boldsymbol{r_A} \leftarrow_R \mathbb{Z}_{2^{\lambda+1}}^{\mu}$. For $j = 1$ to μ, Bob sends $d_j^B \leftarrow \delta_j \oplus d[j]$ to Alice, and Alice sends the pair

$$(a'_{j,0}, a'_{j,1}) \leftarrow (\boldsymbol{t_{d_j^B}}[j] + \gamma_j x_j + \boldsymbol{r_A}[j], \boldsymbol{t_{1-d_j^B}}[j] + (1 - \gamma_j)x_j + \boldsymbol{r_A}[j]) \bmod 2^{\lambda+1}$$

   to Bob. Bob computes $\hat{y}_B \leftarrow \sum_{j=1}^{\mu} a'_{j,c[j]} - \boldsymbol{t}[j] \bmod 2^{\lambda+1}$, and Alice sets $\hat{y}_A \leftarrow -\sum_{j=1}^{\mu} \boldsymbol{r_A}[j] \bmod 2^{\lambda+1}$.
5. (Selecting the First Different Block – Bob's Step.) Bob picks $\boldsymbol{r_B} \leftarrow_R \mathbb{Z}_{2^{\lambda+1}}^{\mu}$. For $j = 1$ to μ, Alice sends $d_j^A \leftarrow \gamma_j \oplus e[j]$ to Bob, and Bob sends the pair

$$(b_{j,0}, b_{j,1}) \leftarrow (\boldsymbol{u_{d_j^A}}[j] + \delta_j y_j + \boldsymbol{r_B}[j], \boldsymbol{u_{1-d_j^A}}[j] + (1 - \delta_j)y_j + \boldsymbol{r_B}[j]) \bmod 2^{\lambda+1}$$

   to Alice. Alice computes $\hat{x}_A \leftarrow \sum_{j=1}^{\mu} b_{j,c[j]} - \boldsymbol{u}[j] \bmod 2^{\lambda+1}$, and Bob sets $\hat{y}_A \leftarrow -\sum_{j=1}^{\mu} \boldsymbol{r_B}[j] \bmod 2^{\lambda+1}$.

Output: Alice outputs $(\hat{x}_A, \hat{y}_A)$ and Bob outputs $(\hat{x}_B, \hat{y}_B)$.

Fig. 11: Compression Protocol for Secure Comparison

Π_{shrink} satisfies input-privacy. The simulation is straightforward (as before, we focus on the case of a corrupted Bob): the simulator $\mathscr{S}im$ runs local copies of $\mathscr{F}_{\mathsf{ET}}$ and $\mathscr{F}_{\mathsf{SC\text{-}prep}}$ and perform honestly the initialization phase, storing the outputs. In steps 1 and 3, $\mathscr{S}im$ simulates $\mathscr{F}_{\mathsf{ET}}$ by outputting random values of the appropriate size. In steps 2 and 4, $\mathscr{S}im$ sends random pairs $(a_{j,0}, a_{j,1})$ and $(a'_{j,0}, a'_{j,1})$ on behalf of Alice; as all element of the pairs in step 2 are either masked by a value $\boldsymbol{s_b}[j]$ unknown to Bob (for some bit b) or by a random mask $\boldsymbol{r}[j]$, and all element of the pairs in step 2 are either masked by a value $\boldsymbol{t_b}[j]$ unknown to Bob (for some bit b) or by a random mask $\boldsymbol{r_A}[j]$, this is perfectly indistinguishable from an honest run of the protocol. In step 5, $\mathscr{S}im$ sends μ random bits on behalf of Alice; as each bit is masked by a random bit $e[j]$ unknown to Bob in the real protocol, the simulation is again perfect. Therefore, the simulated protocol is perfectly indistinguishable from the real protocol from the viewpoint of any environment corrupting Bob that ignores the output of the protocol, which concludes the proof of input privacy.