Executable Proofs, Input-Size Hiding Secure Computation and a New Ideal World

Melissa Chase[1(✉)], Rafail Ostrovsky[2], and Ivan Visconti[3]

[1] Microsoft Research, Redmond, USA
melissac@microsoft.com
[2] UCLA, Los Angeles, USA
rafail@cs.ucla.edu
[3] University of Salerno, Fisciano, Italy
visconti@unisa.it

Abstract. In STOC 1987, Goldreich, Micali and Wigderson [GMW87] proved a fundamental result: it is possible to securely evaluate any function. Their security formulation consisted of transforming a real-world adversary into an ideal-world one and became a de facto standard for assessing security of protocols.

In this work we propose a new approach for the ideal world. Our new definition preserves the unconditional security of ideal-world executions and follows the spirit of the real/ideal world paradigm. Moreover we show that our definition is equivalent to that of [GMW87] when the input size is public, thus it is a strict generalization of [GMW87].

In addition, we prove that our new formulation is useful by showing that it allows the construction of protocols for input-size hiding secure two-party computation for any two-party functionality under standard assumptions and secure against malicious adversaries. More precisely we show that in our model, in addition to securely evaluating every two-party functionality, one can also protect the input-size privacy of one of the two players. Such an input-size hiding property is not implied by the standard definitions for two-party computation and is not satisfied by known constructions for secure computation. This positively answers a question posed by [LNO13] and [CV12]. Finally, we show that obtaining such a security notion under a more standard definition (one with a more traditional ideal world) would imply a scheme for "proofs of polynomial work", a primitive that seems unlikely to exist under standard assumptions.

Along the way, we will introduce the notion of "executable proof", which will be used in our ideal-world formulation and may be of independent interest.

Keywords: Secure computation · Ideal world · Input-size hiding · Proofs of work · FHE · PCP of proximity

1 Introduction

Goldreich, Micali and Wigderson proved in [GMW87] that secure computation is possible for any function, as long as there is a majority of honest players. They

© International Association for Cryptologic Research 2015
E. Oswald and M. Fischlin (Eds.): EUROCRYPT 2015, Part II, LNCS 9057, pp. 532–560, 2015.
DOI: 10.1007/978-3-662-46803-6_18

provided a compiler that on input a circuit computing the function produces a protocol that parties can run to obtain the correct output without revealing any additional information.

Following this result, a long line of works ([GMW87],[GL91],[MR92],[Bea92], [Can05]) developed what is now considered the de facto standard for proving security of a protocol. This notion, which we will refer to as the real/ideal-world paradigm, consists of showing that for any attack that can be carried out by a real-world adversary $\mathcal{A}$ during the execution of the protocol there is a corresponding attack which could be carried out by the ideal-world adversary Sim. Since the setting where Sim works is secure by definition, the real-world protocol must be secure against $\mathcal{A}$.

We note that for general functionalities, the real/ideal world is the only way we know to meaningfully capture security against arbitrarily malicious adversaries. In what follows, we will use *secure* to mean secure against malicious parties, unless otherwise stated, and we will focus, as in [GMW87] on the stand-alone setting.

Beyond the standard real/ideal-world definition. The real/ideal-world paradigm has long been the main measure to evaluate what can and can not be securely computed. The difficulties (and sometimes impossibilities) of proving security under the traditional real/ideal-world formulation have been considered an inherent price to pay for a solid security notion. This has motivated a variety of alternative definitions circumventing such difficulties/impossibilities by explicitly decreasing the security guaranteed by the standard real/ideal world definition. Examples of weaker security notions are those involving trusted third parties, set-up assumptions, superpolynomial-time simulation and so on. This motivates the following question:

Open problem 1: Are there other ways of defining the ideal/real world which would capture all the desirable properties mentioned above, but which might allow us to circumvent some difficulties and impossibilities of the traditional definition?

1.1 A Case Study: Hiding the Input Size

In 2003 Micali, Rabin and Kilian [MRK03] identified an important limitation of the traditional real/ideal-world paradigm. They noticed that in the real world there are interesting cases where a player would like to protect the *size* of his input. This seems increasingly relevant in today's world of big data: one might imagine settings where the number or sensor readings, the size of the customer database, the quantity of user profiles collected, or the total amount of information stored in an advertising database might be considered extremely private or confidential information.

[MRK03] models input-size hiding by saying that the protocol must hide the party's input in a setting where there is no fixed upper bound on the size of the input: although of course honest parties will run on polynomial-length inputs, there is no limit on what those polynomials may be. This guarantees that nothing

is revealed about the input size, and has the additional advantage that it requires protocols where parties' efficiency does not depend on an upper bound, but only on the size of *their actual input* and the complexity of the functionality for that input.[1] As discussed by [MRK03], known previous results do not allow one to obtain such security (e.g., the [GMW87] compiler inherently reveals the size of the input).

Previous work on input-size hiding. Micali et al. explored the case of a player P_0 holding a set Φ of polynomial but unrestricted size (i.e., not upperbounded by any fixed polynomial) and another player P_1 holding an element x. Their function f always outputs x to P_0 and outputs 1 to P_1 if $x \in \Phi$ and 0 otherwise. They gave a game-based solution called "Zero-Knowledge Sets" (ZK sets or ZKS), but achieving standard simulation-based security remained an open problem.

There have been a few other works in this direction. [IP07] studied the evaluation of branching programs as another interesting application of input-size hiding secure computation. Their solution to the above problem in the setting of secure two-party computation (2PC) is round efficient, however it does not provide input-size hiding by the above definition (the length of the program that corresponds to the input of P_0 is bounded), and they do not achieve simulation-based security. More recently, [ACT11] focused on achieving input-size hiding set intersection and obtained an efficient scheme. Their solution only addresses semi-honest adversaries, and security is proved in the random oracle model. [CV12] proposed a construction that satisfies a modified real/ideal-world definition specifically for the set membership functionality studied in [MRK03]; we will discuss below some of the challenges in extending this to cover general functionalities. The importance in practice of input-size hiding secure computation was considered in [CFT13] where the authors presented an efficient protocol for size and position hiding private substring matching[2].

Very recently, [LNO13] discussed the case of general functionalities, but again their constructions are limited to the case of semi-honest adversaries. This leads to the obvious question:

Open problem 2: Is it possible to construct input-size hiding protocols for general functionalities that are secure against malicious adversaries, or is revealing the size of players' inputs inherent in any general 2PC that achieves security against malicious adversaries?

Our variant of input-size hiding. Lindell et al. [LNO13] also provide a general definition of input-size hiding secure computation, essentially extending the

[1] It also has advantages in terms of concrete security in that it results in protocols where the efficiency of the simulator depends only on the complexity of the adversary (and not on some assumed upper bound on its input size).

[2] In this type of protocol the first party could for instance run on input a digitized genome while the second party could run on input a set of DNA markers. The goal in such an use of the protocol is to securely check whether the genome matches the markers, without revealing any additional information, not even the number of markers.

real/ideal-world paradigm in the natural way, and a classification of different types of input-size hiding. Here we will focus on the case where the output size is fixed, and where only one party wants to hide the size of his input. For example, consider the setting where one party wants to run a set of proprietary computations on each element of another party's large private database, and obtain some aggregate results. If the computations can be described in fixed size but the size of the database is private, then our results apply.

As noted by Ishai and Paskin in [IP07] and formally proven in [LNO13] there exist some interesting functionalities for which two-sided input-size hiding is not possible, and similar cases where it is impossible to hide the size of the players' output. Thus, given that we want to achieve results for general functionalities, we restrict ourselves to the one-sided input-size hiding, fixed output-size setting.

We stress that in this work we do not aim to construct protocols for functionalities that work on superpolynomial-sized inputs but only protocols that work for any polynomial-sized input, without a fixed polynomial upper bound on the possible input size.

1.2 Limitations of the Real/Ideal-World Paradigm in the Input-Size Hiding Setting

Why input-size hiding secure 2PC is so tough to achieve. We first begin by recalling the [LNO13] semi-honest input-size hiding protocol, again restricting to the case where only one party's input size is hidden (say P_0) and where the output size is fixed and known. The protocol proceeds roughly as follows: First P_1 generates an FHE key pair and sends the public key along with an encryption of his input x_1 to P_0. Then P_0 uses his input x_0 and the FHE evaluation to evaluate $f(x_0, x_1)$. He sends the result back to P_1, who decrypts and outputs the result.[3] The result is secure in the semi-honest model as long as the FHE is semantically secure and circuit private.

We can immediately see that this protocol is not secure against a malicious $P_0^\star$, since there is no guarantee that the ciphertexts that $P_0^\star$ returns corresponds to $f(x_0, \cdot)$ for some valid x_0. Instead $P_0^\star$ could return an incorrect ciphertext to influence P_1's output or to choose his input conditioned on x_1. More fundamentally, if we consider the real/ideal-world paradigm (which as we noted above is the only way we know to accurately capture security against malicious adversaries for general functionalities), we see that the simulator needs to be able to extract an input x_0 to send to the functionality.

Traditionally, this problem is solved by requiring that the malicious party includes a proof of knowledge (PoK) of his input. In the input-size hiding setting, we could imagine using an input-size hiding PoK where the proof does not reveal the size of the witness (although as we will discuss later, these are not so easy

[3] For simplicity we consider the case where only P_1 receives output, but similar issues occur in the setting where only P_0 receives output, or where both parties receive input.

to achieve). The simulator could then extract from this PoK and send the result to the functionality.

However, here we run into trouble: Recall that the protocol cannot fix any polynomial upper bound on the length of the prover's input (otherwise it would not really be input-size hiding). Now, suppose that an adversary $P_0^\star$ decides to run the protocol using a superpolynomial-length input. Note that using a superpolynomial-length input does not necessarily mean that the adversary must be superpolynomial; it may be that the adversary is working with a compact representation a much longer input. And suppose that the adversary can find a way to efficiently run the protocol and generate a proof given only this short representation. Then finding a corresponding polynomial-time ideal-world adversary will be impossible: this ideal adversary would have to send the functionality an equivalent input (of superpolynomial size!) and this would require superpolynomial time. (Concretely think of the set membership function; if the set Φ chosen by $P_0^\star$ consists of all k-bit strings such that the last bit is 0, then $P_0^\star$ has in mind an input of exponential size, and thus efficiently extracting this input will be impossible. Notice that $P_0^\star$ could still be efficient because it is possible to represent that huge set as a small circuit.)

We could of course simply assume that we can design a PoK or more generally a protocol which an honest party can run on any polynomial-sized input, but for which no polynomial-time adversary can execute the protocol on *any* superpolynomial input. However, we argue that this seems inherently non-standard: essentially, we would be assuming an object which is easy on any polynomial-sized input but becomes computationally infeasible as soon as the input is even slightly superpolynomial. This is therefore the inherent major difficulty. Even if $P_0^\star$ is guaranteed to be efficient (polynomial time), we have no way of guaranteeing that the *input that* $P_0^\star$ *is implicitly using* is of polynomial size.

Formalizing the limitation of the real/ideal-world definition w.r.t. hiding the input size. Lindell et al. [LNO13] conjectured that their construction (which is shown to be secure against semi-honest adversaries) could be augmented with a "proof of work" in order to be secure against malicious adversaries; essentially the idea was to solve the problem described above by requiring the adversary to prove that the size of his input is polynomial. Unfortunately currently there is no candidate construction for proofs of work under standard assumptions; moreover, as we just mentioned, non-standard assumptions seem inherent, in that a proof of work makes an assumption on the abilities of a polynomial-time adversary on a problem instance that is only guaranteed to be superpolynomial (rather than exponential). This prompts the following question.

Open problem 3: is it possible to achieve input-size hiding secure 2PC for all functionalities in the traditional real/ideal world without relying on proofs of work?

In Section 2.1, we formalize this notion of proofs of work, which we refer to as a proof of polynomial work or PPW. On the positive side, this seems to capture the intuition for what [LNO13] use in their proposed extension to security against malicious adversaries. On the other hand, as we have discussed,

these proofs of work seem inherently to require non-standard assumptions. This then presents the next open question:

Open problem 4: How can we meaningfully define input-size hiding secure 2PC if we are interested in constructions under standard assumptions?

1.3 A New Ideal World

Given the above limitations, we take a step back and consider what we need from our security definition. The traditional real/ideal-world paradigm has three key features: 1) In the ideal world it is clear from inspection what functionality is provided and what an adversary can learn or influence. Indeed, players are simply required to send their inputs to a trusted functionality $\mathcal{F}$ that will then evaluate the function and distribute the outputs to the players. This guarantees that even if P_0 is corrupted[4]: a) the output is correct and consistent with some valid input x_0, b) P_0's choice of input x_0 cannot depend on P_1's input x_1, and c) P_0 cannot cause a selective failure, i.e. P_0 cannot cause the protocol to fail conditioned on the input chosen by P_1. 2) In the ideal world, security holds unconditionally. Again, this is important because we want it to be obvious that the ideal world provides the desired security. 3) The ideal world is efficient (i.e., the ideal functionality $\mathcal{F}$ and the ideal adversary Sim are polynomial time). This is crucial when we want to use secure computation as a part of a larger protocol: if a protocol achieves a real/ideal-world definition, then we can argue that we could replace it with $\mathcal{F}$ and Sim. However if $\mathcal{F}$ and Sim are inefficient, then the resulting game will not be polynomial time, and any reductions to standard hardness assumptions will fail.

Our key observation then is that in order to enforce these properties the functionality does not actually need to receive P_0's input x_0. Instead it only needs a) to ensure that x_0 is fixed and independent of x_1, and b) an efficient way to compute $f(x_0, x_1)$ consistently with x_0.

The new ideal world. This leads to the following relaxation of the ideal world. Instead of requiring P_0 to send his input directly, we instead require P_0 to send an *implicit representation* of his input. The only requirement on this representation is that it must uniquely define the party's true input x_0.[5] We will use $\mathsf{Rep}(x_0)$ to denote some implicit representation corresponding to x_0. (Our ideal world will take the specification of this representation as a parameter; the definition will require that there exist a valid representation under which the real and ideal worlds are identical.)

Then in order to allow the functionality to compute P_1's output, the ideal P_0 will also send a circuit C describing how to compute the output given any input x_1 from P_1. Finally, we require that this circuit C on input x_1 produce not

[4] We use corrupt P_0 as an example: clearly the analogous properties also hold if P_1 is corrupt.

[5] For our ideal world to be realizable, it must hold that any polynomial-sized input has a polynomial-sized representation.

only the output $f(x_0, x_1)$, but also some evidence that this output is correctly computed according to x_1 and the input defined by $\mathsf{Rep}(x_0)$. Intuitively, because P_0 sends $\mathsf{Rep}(x_0)$, this guarantees that x_0 is fixed and independent of x_1, and because C provides evidence that its output is correct, we are guaranteed that (if the evidence is valid), the output provided to P_1 is indeed $f(x_0, x_1)$. Finally, we note that the only way for P_0 to cause a selective failure is for $C(x_1)$ to output invalid evidence - in all other cases the functionality will send to P_1 a valid output.

That leaves two issues to resolve: 1) how to formalize this "evidence", and 2) what happens if $C(x_1)$ produces evidence that is invalid.

Executable proofs. Ignoring the implicit input for a moment, we might consider an ideal world which works as follows: $\mathcal{F}$ receives from one of the parties P_0 not only the input x_0 but also a circuit C. Then, instead of computing the output directly, $\mathcal{F}$, having obtained input x_1 from P_1, runs $C(x_1)$ to obtain (y, w), where w is an NP witness for the statement $y = f(x_0, x_1)$. $\mathcal{F}$ verifies w and outputs y to P_1 iff the verification succeeds.[6] Clearly this is equivalent to the traditional notion of security because the NP witness *unconditionally* guarantees that $\mathcal{F}$ produces the correct output.

Now, what if we want $\mathcal{F}$ to be able to verify statements which may not be in NP? [7]

We might want to consider cases where, rather than an NP-witness, $\mathcal{F}$ is given *some other kind of proof.* As long as the proof is unconditionally sound and efficiently verifiable we still have a meaningful notion of security. If we want to consider more general languages, we might imagine providing $\mathcal{F}$ with a PCP to verify instead of an NP witness. Because PCPs are unconditionally sound, this would still satisfy the first property. However, even if they can be verified efficiently, the PCPs for the language could still be exponentially long while as argued above, we want our ideal parties and $\mathcal{F}$ to run in polynomial time, so it might be impossible for the ideal party to output an entire PCP. Thus we introduce the notion of an "executable proof". This is a new concept of proof that has similarities with classical proofs, interactive proofs and PCPs. However executable proofs differ from the above concepts since they focus on a giving to a verifier the capability of checking the veracity of a statement (potentially not in NP) by running a circuit (i.e., an executable proof) locally.

In particular we will make use of *executable PCPs.* An executable PCP is a circuit which on input i produces the ith bit of a PCP (i.e., a circuit representation of a PCP). Given a description of such a circuit, $\mathcal{F}$ can run the PCP

[6] Addressing the case where verification fails is somewhat more subtle. See below for more discussion.

[7] For example in the context of input-size hiding protocols, as discussed above, we might consider the case where $\mathcal{F}$ is given only a compact representation of x_0 while the actual x_0 may be of superpolynomial length. In this case it may be that verifying $f(x_0, x_1) = y$ given only x_1 and the representation of x_0 is not in NP. (Note that in this case a witness that includes the explicit x_0 would have superpolynomial length, and so would not be a valid NP witness.)

verifier to unconditionally verify the statement. If the circuit representation is polynomial sized, this verification is efficient.

The nice point here compared to just using a witness for the statement is that the description of the executable PCP can be much shorter than a standard witness (it essentially depends on the complexity of describing the given witness); this will play a fundamental role in our constructions.

Ideal Errors. We solve the second issue by slightly modifying the notion of real/ideal-world executions. In the ideal world $\mathcal{F}$ will verify the proof π and then complete the protocol only if the proof was accepting. If π failed in convincing $\mathcal{F}$, then $\mathcal{F}$ will send P_1 a special message "ideal error", indicating that the ideal input was invalid. Finally, our definition for secure computation will require that the real-world execution never produce this "ideal error" as output of P_1.[8] This guarantees that in any secure realization any real world adversary must have a corresponding ideal world adversary who causes this error to occur in the output of the ideal P_1 only with negligible probability. (This is because any ideal world execution in which the ideal honest player P_1 outputs such an error with non-negligible probability would be instantly distinguishable from the real world where such an output for P_1 cannot appear by definition).

We stress that the flow of data follows the standard notion of the real/ideal paradigm where players send data to $\mathcal{F}$ first, and then $\mathcal{F}$ performs a computation and sends outputs (waiting for the approval of $\mathcal{A}$ for fairness reasons) to players. By the unconditional soundness of the proof given by π, it is clear that the adversary is committed to an input and the output is consistent with that input (as long as "ideal error" does not occur). And finally, our ideal adversary runs in polynomial time, and the implicit input and the circuit C are produced by the ideal adversary; this means that $\mathcal{F}$ will also run in polynomial time[9]. Thus, our new formulation has all the desirable properties mentioned above. As a sanity check, we note that our definition is equivalent to the traditional real/ideal-world definition in the case where the input lengths are fixed known polynomials.

1.4 Constructing Input-Size Hiding Under the New Definition

Finally, we show that our new definition is realizable. We show a protocol that satisfies our input-size hiding secure 2PC and that builds on top of several recent advanced tools, and follows the outline from the semi-honest construction of [LNO13] described above. Roughly, we will use fully homomorphic encryption (FHE) as in [LNO13] so that the adversary P_0 can evaluate the function given only an encryption of P_1's input, which will give privacy for P_1. To guarantee

[8] Technically we can guarantee this by requiring that the real execution replaces any "ideal error" in its output by $\perp$.

[9] To see this note that a polynomial time adversary must produce polynomial sized circuits for the ideal input and the circuit C, $\mathcal{F}$'s operation consists of evaluating the circuit C and running a polynomial time verification algorithm on the proof produced by C, and polynomial sized circuits can always be evaluated in polynomial time and produce polynomial length output.

that the adversary P_0 must behave correctly we will require that P_0 commit to (via an input-size hiding commitment) and prove knowledge of his input before receiving the ciphertext from P_1. However, in designing this proof of knowledge, we are faced with the issue discussed above of extracting a potentially very long input from a short proof of knowledge. To address this we will use a special proof of knowledge introduced by [CV12] called a "universal argument of quasi knowledge" (UAQK) that has short communication complexity and provides a (non-black box) extractor that outputs a *short* "implicit" representation of the witness. Therefore the issue of efficiently extracting from an adversary that has in mind a very large input is solved by extracting a small implicit representation of this input. Looking ahead to the proof, it is this implicit represent that will be used as Rep in the ideal world. After applying the FHE evaluation to compute an encryption of $f(x_0, x_1)$, P_0 is required to give a UAQK to prove knowledge of a *PCP of proximity* proving that her work on ciphertexts was done correctly. With a PCP of proximity, a verifier can verify the proof while accessing only selected bits of the statement and proof. Again looking ahead, the simulator will use the code of an adversary P_0 and the extractor of the UAQK to generate a circuit which can output single bits of the PCP of proximity; such a circuit is used to instantiate the executable PCP in the ideal world. Here the use of PCPs of proximity is critical since they allow us to verify proofs by reading only small parts of statements and proofs. (This allows the functionality to verify correctness of the claimed output given only an implicit representation of P_0's input.)

Concretely, our protocol can be instantiated by assuming FHE and CRHFs, this proves that our notion can be achieved under standard assumptions, avoiding random oracles [BR93, CGH98], non-black-box extraction assumptions [Dam92] (see [Nao03] about their non-falsifiability), superpolynomial-time simulation, and proofs of work.

1.5 Short Summary of Our Results

In this work we embark on the challenging task of defining and achieving input-size hiding security against malicious adversaries under a simulation-based definition, following the spirit of [GMW87]. In Section 2.2 we give a new definition of an ideal world that has all the desirable properties of the real/ideal paradigm mentioned above, *and* allows us to capture input-size hiding, thus answering the last open question. We pair this result with another contribution: in Section 2.1 we show that achieving input-size hiding secure 2PC under the standard real/ideal-world formulation implies the existence of proofs of work. This solves the third problem and gives evidence of the power of our new formulation. Finally, in order to show that our size definition *is* realizable under standard assumptions, in Section 3 we provide a construction which can be instantiated under FHE and CRHFs. Thus we also provide a solution to the second open problem. All together these results provide an answer to our first question: a demonstration of how considering a modified security model can still give meaningful security

while allowing us to circumvent impossibility results inherent in the traditional definitions.

1.6 Discussion

Other variations on the ideal world. One obvious alternative would be to simply allow for an unbounded functionality $\mathcal{F}$: $\mathcal{F}$ could then extract the entire (potentially exponentially long) input from the implicit input C, and then compute the appropriate output. However, our aim here is a definition giving guarantees *as close as possible to standard secure 2PC* and, as mentioned above, one crucial property of secure 2PC is that the functionality is efficient.

We compare our variation of the ideal world with the simulation-with-aborts definition used for fairness issues. Simulation-with-aborts is a simple variation of [GMW87] and *must* be used in light of an impossibility result. However it introduces a security weakness in the ideal world that is reflected in the real world. While our variation is somewhat less straightforward, it can be used to achieve a stronger security notion under standard assumptions. Our variation applied to the simulation-with-aborts definition, does not introduce any additional security drawback that can be reflected in real-world protocols. Moreover, it allows us to capture input-size hiding, and as we will see, it has the indisputable benefit that it can be realized under standard assumptions.

Timing attacks: input-size hiding is relevant also in practice. While input-size hiding computation may be theoretically interesting, one might ask whether it makes sense in practice. As already pointed out in [IP07, CV12], any protocol may be vulnerable to a basic timing attack in which an adversary can guess the size of the honest player's input purely based on the time he takes to perform each computation.

We note, however, that there are many practical scenarios where such attacks are not applicable. Indeed, in order to mount such an attack the adversary needs to be able to accurately estimate the resources of the other player; in many cases this may be difficult. Furthermore, in many applications a player may have time for preprocessing or access to large clusters (as in computations that are run only infrequently, or protocols involving cloud computing companies). Since the adversary will not know how much precomputation/parallelization has been used, it cannot conclude much about the size of the honest party's input. For example, the standard protocols for ZK sets allow for preprocessing: all computation that depends of the size of the input can be performed before any interaction occurs. We note that our definition does not preclude the existence of protocols for interesting functionalities that make use of precomputation/parallelization to prevent these basic timing attacks.

Comparison with [CV12]. Chase and Visconti [CV12] made a first step in this direction by defining and realizing a real/ideal-world notion for a functionality modeling ZK sets called "secure database commitment". The solution of [CV12] is based on two key ideas: 1) defining a modified ideal world; 2) simulating by

relying on a new tool that implements a special form of proof of knowledge with short communication complexity. More specifically, they define an ideal world where (roughly) player P_0 sends $\mathcal{F}$ a circuit C which implicitly defines a set, while player P_1 directly sends its input x. $\mathcal{F}$ will then compute $(x, y = C(x))$ and send x to P_0 and y to P_1. This ideal world is still perfectly secure because any circuit C uniquely defines a set (i.e., the set of all strings on which the circuit outputs 1).

Given this ideal world, they are still faced with the issue discussed above of extracting a potentially very long input from a short proof of knowledge. This was the original motivation for the introduction of UAQKs: The issue of efficiently extracting from an adversary that has in mind a very large input is solved by extracting a small implicit representation of this input. [CV12] then shows how to construct an ad-hoc circuit that can be sent by Sim to $\mathcal{F}$ from such an implicit representation.

Unfortunately, while the result of [CV12] solves the problem left open by [MRK03], their solution does not extend to other functionalities. One limitation of the approach in [CV12] is that it gives input-size privacy and input privacy for P_0 but no input privacy at all for P_1. This is appropriate for the functionality they consider, but obviously undesirable in general. The more significant issue, however, is that in the "secure database commitment" functionality the correctness of P_0's input (the input whose size must be hidden) can be verified simply by inspection. Indeed $\mathcal{F}$ can simply check that the provided circuit C has k input wires and only one output wire. This means that for every k-bit string x, C decides the membership or non-membership of x with respect to the unique set that is implicitly defined by C itself. Note that C both defines P_0's input set and efficiently computes the set membership functionality. The obvious question then is whether this approach can be generalized.

Unfortunately for other functionalities the above approach fails spectacularly. The first issue is the possibility that P_0 might send a circuit C whose behavior is not consistent with any valid input. For example, consider the function $f(\Phi, x)$ that outputs the number of elements of Φ that are greater than x. Now, if an adversary P_0 sends a circuit C which is supposed to compute $f(\Phi, \cdot)$ to the functionality, there is no way for the functionality to verify that this circuit C does compute such a function. For example, it is possible that $C(x) > C(x')$ when $x' < x$, which clearly can not be consistent with any set Φ. Thus, a functionality $\mathcal{F}$ which simply receives a circuit C from $\mathcal{A} = P_0^\star$, and sends $C(x)$ to P_1, would be clearly insecure. A second issue involves $P_0^\star$ learning P_1's input. Indeed, consider a circuit C that on input x_1 instead of giving in output $(y_0, y_1) = f(x_0, x_1)$ (i.e., y_0 for P_0 and y_1 for P_1) outputs (x_1, y_1). In this case P_1 would get the correct output, but $P_0^\star$ manages to learn P_1's private input. A third issue is that of *selective failure*, in which $P_0^\star$ can adaptively corrupt P_1's output depending on P_1's input. For example, $P_0^\star$ could compute a circuit that depending on the input x_1 will output a value that is not in the output space of the function; $\mathcal{F}$ might send this invalid output to P_1 or send an error message, but in both cases P_1's reaction would allow $P_0^\star$ to learn something about his input. Notice that the 1st

and 3rd issues above also apply in case only P_1 receives an output, while the 2nd issue also applies when only $P_0^\star$ receives an output.

Our work avoids these issues by including the executable proof and an "ideal error" in the ideal world. A valid executable proof guarantees that the functionality will only output a result if it does correspond to $f(x_0, x_1)$ for the specified x_0. The "ideal error" will guarantee that if C ever produces and invalid executable proof then the two worlds are clearly distinguishable; this holds because in the ideal world the functionality sends "ideal error" to P_1 while by definition "ideal error" can not appear as output of P_1 in the real world.

1.7 Open Problems and Future Work

We observe that there are several new ideas here that might be interesting for future study. First, executable PCPs: what languages have polynomial-sized executable PCPs? Note that an efficient executable PCP is different from a PCP with efficient prover - the latter would require that the PCP be of polynomial length, whereas the PCP represented by an efficient executable PCP might be exponentially long; we only require that each bit be efficiently computable.

Our new ideal world also presents several interesting questions. First there is the use of executable PCPs (or more generally, any type of unconditionally sound proofs) in the ideal world: might this allow us to avoid some of the impossibility results in secure computation? Similarly, we propose an ideal world in which $\mathcal{F}$ only receives an implicit representation, evaluates a program produced by one of the parties to obtain the output, and then merely verifies that the result is correct. Again, one might ask whether this gives more power than the traditional ideal world where $\mathcal{F}$ receives the explicit input and computes the output itself.

We also introduce the idea of proofs of polynomial work (PPW) and show one possible application. Other interesting questions would be to look at possible constructions of PPW, or to formally show that PPW requires non-standard assumptions, or to consider other applications where PPW could be useful. One could also consider how to achieve input-size hiding using superpolynomial-time simulation.

We only study the simplest possible world here - we do not for example address the problem of composition, or of obtaining efficient protocols or those that would allow for preprocessing or parallelization. Finally, we leave the question of addressing the case of hiding the input-size of both players (at least for some functionalities) for future research.

2 Secure 2-Party Computation and Proofs of Work

2.1 Input-Size Hiding and Proofs of Polynomial Work

We now show that when the standard ideal world is considered, input-size hiding is hard to achieve under standard assumptions since it implies some form of proof of work that we call proof of polynomial work. There have been many works

studying proofs of work, going back to [DN92]. However, as far as we know our notion is not captured by any of the previous definitions.

Roughly, for security parameter k, a PPW is a protocol between P,V where P and V take as input a number n represented as a k-bit string. For honest players we expect n to be polynomial in k. For any polynomial time $\mathcal{A}$ playing as P we want a polynomial upper bound for the n's on which he can cause V to accept. Intuitively this means that if $\mathcal{A}$ can convince V to accept an n, we know that it is bounded by a polynomial (i.e., there exists a polynomial p such that V will reject any $n > p(k)$).

Definition 1. *A proof of polynomial work (PPW) is a pair (P, V) such that the following two properties hold: 1) (correctness) there exist fixed polynomials* poly *and* poly′ *such that the running time of P and V for security parameter k and input $n < 2^k$, is respectively* poly(n) *and* poly′(k)*, and the output of V is 1. 2) (security) for every polynomial-time adversary $\mathcal{A}$, there exists a polynomial p and a negligible function μ such that for sufficiently large k, for any $n \geq p(k)$, V on input n, interacting with $\mathcal{A}$ outputs 1 with probability at most $\mu(k)$.*

Theorem 1. *One-sided input-size hiding secure 2PC (secure with respect to the standard ideal world) implies the existence of a proof of polynomial work.*

Proof. First, consider a PPW with a somewhat weaker security property, which only guarantees that $\mathcal{A}$ succeeds on $n \geq p(k)$ with probability at most $1/2+\mu(k)$ for some negligible function μ. Note that given a PPW with this weak security property, we could easily construct a PPW with the above property just by sequential composition. Thus, we focus here on showing that input-size hiding with a standard ideal/real-world security definition would imply a weak PPW.

We now show how to construct a weak PPW (P, V) by starting from a protocol for input-size hiding secure 2PC (P_0, P_1). Consider the functionality $\mathcal{F}$ that receives a set of integers from P_0 and an integer n from P_1 and proceeds as follows: if the set is the list of numbers from 1 to n, then $\mathcal{F}$ outputs 1 to P_1, otherwise it outputs $\perp$ to P_1. Now, suppose we had a protocol for $\mathcal{F}$ that hides the size of P_0's input under the standard simulation-based definition of secure 2PC. If there exists an input-size hiding 2PC protocol for $\mathcal{F}$ then we obtain the following PPW (P, V): P plays as P_0 on input $\Phi = \{1, \ldots, n\}$ and wants to make V running on input n output 1; both parties use security parameter k. By efficiency of the 2PC we have that if n is represented as a k-bit strong, then V runs in time polynomial in k and P runs in time polynomial in n. And clearly by observation honest $V(n)$ outputs 1 when interacting with honest $P(n)$. Therefore correctness is satisfied.

We then consider security. Suppose for contradiction that the weak PPW property does not hold. Then there exists a polynomial-time adversary $\mathcal{A}$ such that for all polynomials p and all negligible functions μ, there are infinitely many k such that on some $n > p(k)$ (where n is represented as a k-bit string), $\mathcal{A}$ causes $V(n)$ to output 1 with probability greater than $1/2 + \mu(k)$.

2PC guarantees that for any $\mathcal{A}$ there is a 2PC simulator and a negligible function μ' such that for all sufficiently large k, for all inputs n represented as

k-bit strings, the real and ideal executions can be distinguished with probability at most $1/2 + \mu'$.

Let p' be the running time of the 2PC simulator for $\mathcal{A}$, let $p = 2p'$, consider $\mu = 2\mu'$, and let D be the 2PC distinguisher that outputs 1 if $V(n)$ outputs 1, and a random guess otherwise. Note that the simulator is assumed to have expected running time $p'(k) = p(k)/2$, so with probability at least $1/2$, it will run in time at most $p(k)$. However, in order to cause $V(n)$ to output 1 it must output the set $\{1, \ldots, n\}$ which requires at least n time. Thus, for $n > p(k)$, in the ideal game $V(n)$ outputs 1 with probability at most $1/2$.

Now, if the weak PPW property does not hold, as explained above there are infinitely many k such that on some $n > p(k)$, $\mathcal{A}$ causes $V(n)$ to output 1 with probability greater than $1/2 + 2\mu'(k)$. For every such n, D clearly succeeds in distinguishing real and ideal executions with probability greater than $1/2+\mu'(k)$. However, by 2PC security as we have said before, for all sufficiently long n, D must have advantage at most $1/2+\mu'(k)$. Thus, we have reached a contradiction.

We do not have any candidate constructions based on standard assumptions, and in fact this seems difficult. We could of course construct them by direct relying on some number-theoretic assumptions, but this would require strong generic-group assumptions.

2.2 Our New Definition: A New Ideal World

We give a new general formulation for 2-party computation; we will show later that under this formulation we can achieve 1-side input-size hiding secure computation for any functionality.

First let us informally review the standard ideal-world for secure 2PC of an efficient function $f = (f_0(\cdot,\cdot), f_1(\cdot,\cdot))$, considering the simulation-with-aborts variation. An ideal-world player P_b for $b \in \{0,1\}$ sends his input x_b to a functionality $\mathcal{F}$ and gets as output $f_b(x_0, x_1)$. An adversary $P_b^\star$ after getting her output, can decide whether P_{1-b} should receive his output or not. The communication is always through $\mathcal{F}$.

In this work we consider the setting of *static* inputs and corruptions (e.g., inputs of both parties are specified before the execution of any protocol and the adversary corrupts one of the two players before the protocol starts). Furthermore, we consider secure computation protocols with *aborts* and no *fairness*. This notion is well studied in literature [Gol04]. More specifically, the adversary can abort at any point and the adversary can decide when (if at all) the honest parties will receive their output as computed by the function.

Before presenting our new definition, we will discuss a few necessary concepts.

Input sizes. In the discussion and definition below we will assume w.l.o.g. that P_0 is the party who wishes to hide the size of his input. We use k to denote the security parameter. Honest P_0's input length is assumed to be $\mathsf{poly}(k)$ for some polynomial poly, although this polynomial is not fixed by the protocol or known to P_1. The input of P_1 is then of a fixed length, so w.l.o.g. we assume it is a k-bit string; we also assume that the output of the function f is always a k-bit string.

All parties (honest and malicious) run in time polynomial in k. As discussed in Section 1, we want our ideal functionality $\mathcal{F}$ to run in time polynomial in the size of the messages it receives from the parties; since the ideal parties are polynomial time (in k), the functionality will be as well (this polynomial may depend on the adversary). Throughout the discussion, unless otherwise specified "polynomial" means a polynomial in the security parameter k which may depend on $P_0^\star$.

Implicit representation of data. As discussed in Section 1, we will consider an ideal world in which one party only submits an implicit representation of this input. The only properties that we require for our definition is that this representation be efficiently computable for any polynomial-size input, and that the input is uniquely defined by its implicit representation. More formally, we say that an implicit representation is defined by a potentially unbounded function $\mathsf{Decode} : \{0,1\}^* \to \{0,1\}^*$, which maps each implicit representation back to an input string, and an efficiently computable injective function $\mathsf{Rep} : \{0,1\}^* \to \{0,1\}^*$ which computes an implicit representation of each input. We require that for all $x \in \{0,1\}^*$, $\mathsf{Decode}(\mathsf{Rep}(x)) = x$, and for any string x we refer to $\mathsf{Rep}(x)$ as an implicit representation of the explicit string x.

One implicit representation which we will use often is the circuit representation, by which we mean a circuit that when queried on input i outputs the i-th bit of the explicit string.[10] This representation can be of much shorter size (depending on the data), even potentially logarithmic in the length of the string. As an example, consider the 2^k-bit string s where all odd positions are 0s while all even positions are 1s. Clearly, one can construct a circuit of size $O(k)$ (therefore of size logarithmic in the size of s) that on input i outputs the i-th bit of s. Given a circuit representation s' of a string s, we denote by $s'(i)$ the i-th bit of the underlying explicit string.

Ideal errors. As discussed in Section 1, our formulation is based on an ideal-world experiment whose output clearly specifies whether the computation has been performed correctly or whether something went wrong. Therefore, the honest player's output will be either (this is the good case) the canonical output corresponding to an element in the range of the function, or (this is the bad case) a special message `ideal error` that does not belong to the output space of the function. The ideal-world adversary in standard 2PC is allowed to stop the delivery of the output to the honest player, (to avoid fairness impossibility results proved in [Cle86]), but will not be allowed to stop delivery of the `ideal error` message. (This `ideal error` represents a failure that should never happen in a secure real-world implementation, and we need it to be visible in the output of the ideal-world experiment.) This is similar to the way that ideal world inconsistencies are handled in UC definitions (see for example $\mathcal{F}_{SIG}$ in [Can03][11]).

[10] To make this formally injective we can define all invalid circuits to be implicit representations of 0.

[11] That functionality outputs an error and halts, which also makes the two games clearly distinguishable (the functionality never produces output for the honest party). We chose to make this explicit by defining the ideal error, but it has the same effect.

We require by definition that `ideal error` does not occur in the real world. A real protocol is considered to be secure for a function f if: 1) for all real-world adversaries there exists an ideal-world adversary such that the executions in the real and ideal worlds are indistinguishable: 2) the honest parties in the real world execution do not ever output `ideal error`.[12]

This means that for every real-world adversary, there must be an ideal-world adversary that causes the ideal honest party to output `ideal error` with at most negligible probability, but produces the same execution output (i.e., adversary's view and honest player output). This is because any ideal adversary that causes `ideal error` with non-negligible probability would make the two executions clearly distinguishable.

Our ideal error helps to define security because it characterizes the "bad" events of our ideal world; combining this with the fact that the real-world protocol never produces an ideal error and with the fact that any adversary of the real-world can be simulated in the ideal world, we get the desired security guarantees.

Special unconditionally sound proofs: executable PCPs. Our ideal functionality will be parameterized by an unconditionally sound proof system. We will say that a protocol satisfies input-size hiding for a given function if there exists an unconditionally sound proof system such that (for all adversaries there exists a simulator such that) the executions of the ideal and real worlds are indistinguishable. The only requirements we will make on this proof system are that it be unconditionally sound, with negligible soundness error, and that verification time be polynomial in the size of the proof. This does mean that we have some cryptographic tools in the ideal model, however the unconditional soundness means that security of the ideal model is not based on any computational assumption. As discussed in the intro, the ideal P_0 will send to the functionality a circuit C which computes both the output and a proof; this proof will allow the functionality to verify correctness of that output in polynomial time (since the ideal-world adversary is polynomial time, this proof will have polynomial size). We require unconditionally sound proofs so that in the ideal world correctness holds unconditionally. Computational assumptions may risk the basic guarantee that the functionality be trivially secure.

When we prove security of our size-hiding protocol, we will instantiate the proofs in the ideal world with *executable PCPs*. An executable PCP for language L is defined by a probabilistic polynomial-time verifier V, and satisfies perfect completeness ($\forall x \in L$, $\exists \pi$ such that $\Pr[V(x, \pi) = 1] = 1$) and unconditional soundness ($\forall x \notin L, \forall \pi, \Pr[V(x, \pi) = 1]$ is negligible in the length of x), where the probabilities are over V's random coins. Equivalently, we can view it as a circuit version of a PCP: the prover sends the verifier a circuit "computing" the PCP. In order to verify the PCP, the verifier will run the circuit on inputs corresponding to his PCP queries; the circuit's output will be viewed as the corresponding bit

[12] This can be enforced by requiring that a real-world experiment checks whether the honest party outputs `ideal error`, and if so replaces this output with $\perp$.

of the PCP; hence the name "executable PCP". (We will equivalently use the term executable proof.) We emphasize that the soundness is unconditional, so as argued above, we can use it in the ideal world. We will denote by EX_π a scheme implementing an executable proof, and by π an individual proof that can be verified by $\mathcal{F}$.

Functionalities. In 2PC, the output computed by the functionality $\mathcal{F}$ is a function $f(x_0, x_1)$ of the inputs x_0 and x_1 received from P_0 and P_1; traditionally f is described by a circuit. This means that the size (or at least an upperbound) of inputs and outputs is fixed as specified by the domain and range of f. For instance, if f is the set intersection function, we have that the domain of f fixes a bound on the number of elements in each player's set. Instead, in our setting we want honest players to have inputs of unrestricted length, because any fixed polynomial bound inherently reveals some information about the size of the inputs. For every input length there should be a different function for implementing the *same* functionality. We need therefore a concise formalization of a functionality that corresponds to a family of functions accommodating different input lengths.

The natural way to formalize this consists of describing the functionality as a deterministic efficient machine M, than on input a pair $(1^i, 1^j)$ outputs the circuit $C_{f_{i,j}}$. For each i, j, such a circuit describes the function $f_{i,j}$ that has the appropriate domain to accommodate inputs of size i and j. (Note that we will assume throughout that the length of the output is always bounded and independent of i and j.) We will focus here on the case of hiding the size of the input of only one player, so we will only need to consider families indexed by a single size parameter (i.e., $M(1^i), C_{f_i}$ and f_i). We denote by f the family of functions.

On using circuits instead of Turing machines. Circuits have the advantage that any circuit produced by a polynomial time machine will run in polynomial time on all inputs; using them for implicit input makes clear that the functionality is efficient. Using circuits for the functions $f_{i,j}$ is arbitrary but closer to other 2PC work.

Putting this together. We follow the outline suggested in Section 1. Formally, we require P_0 to send to $\mathcal{F}$ an implicit representation $\bar{x}_0$ of its input x_0, and a circuit C that for any input x outputs the pair $((y_0, y_1), \pi)$ where $(y_0, y_1) = f(x_0, x)$. Here π is a proof that can be used by $\mathcal{F}$ to check unconditionally that (y_0, y_1) is correct according to f, x and $\bar{x}_0$. $\mathcal{F}$ will therefore check the correctness of the proof given by π. If π convinces $\mathcal{F}$, then y_0 is sent to P_0 and y_1 is sent to P_1, keeping in mind that in this case output delivery to an honest player is conditioned on the approval of the adversary. If instead $\mathcal{F}$ is not convinced by π, then $\mathcal{F}$ does not send any output to P_0 and sends `ideal error` to P_1, without waiting for P_0's approval.

Execution in the ideal world. We now describe an ideal execution with a PPT adversary Sim who has auxiliary input z and controls one of the parties. The PPT trusted third party that computes $f = \{f_i\}_{i\in\mathcal{N}}$ will be denoted by $\mathcal{F}$. Without loss of generality, let P_0 be the player interested in keeping the size of his input private. We will first describe the execution in the setting in which the adversary Sim controls P_1, then follow with the setting in which the adversary controls P_0.

1. *Inputs:* Sim receives as input x_1 and auxiliary input z, party P_0 receives as input x_0 and constructs a triple $(i, \bar{x}_0, \mathrm{C})$, where $i = |x_0|$, $\bar{x}_0$ is an implicit representation of x_0, C as described above computes $f_{|x_0|}(x_0, \cdot)$ and the executable PCP.
2. P_0 *sends input to trusted party:* upon activation, P_0 sends $(i, \bar{x}_0, C)$ to $\mathcal{F}$.
3. Sim *sends input to* $\mathcal{F}$ *and receives output:* whenever Sim wishes, it may send a message x_1' to $\mathcal{F}$, for any x_1' of its choice. Upon receiving this message, $\mathcal{F}$ computes $((y_0, y_1), \pi) = C(x_1')$, verifies π, and sends y_1 to Sim. (Once an output has already been sent to Sim, all further input messages are ignored by $\mathcal{F}$.)
4. Sim *instructs* $\mathcal{F}$ *to answer* P_0*:* when Sim sends a message of the type `end` to $\mathcal{F}$, $\mathcal{F}$ sends y_0 to P_0. If inputs have not yet been received by $\mathcal{F}$, then $\mathcal{F}$ ignores message `end`.
5. *Outputs:* P_0 always outputs y_0 that it received from $\mathcal{F}$. Sim may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, the initial input x_1, and the output obtained from $\mathcal{F}$.

The ideal execution of $f = \{f_i\}_{i\in\mathcal{N}}$ using implicit representation Rep and an executable PCP EX_π (with security parameter k, initial inputs (x_0, x_1) and auxiliary input z to Sim), denoted by
$\mathrm{IDEAL}_{f,\mathsf{Sim},\mathsf{Rep},EX_\pi}(k, x_0, x_1, z)$ is the output pair of P_0 and Sim from the above execution. We now consider the case in which the adversary Sim corrupts P_0 while P_1 is honest.

1. *Inputs:* P_1 receives as input x_1, Sim receives as input x_0 and auxiliary input z.
2. P_1 *sends input to* $\mathcal{F}$*:* upon activation P_1 sends x_1 to $\mathcal{F}$.
3. Sim *sends input to* $\mathcal{F}$ *and receives output:* whenever Sim wishes, it may send a message $(i', \bar{x}_0', C')$ to $\mathcal{F}$. Upon receiving this message, $\mathcal{F}$ computes $((y_0, y_1), \pi) = C'(x_1)$. $\mathcal{F}$ then verifies π to check unconditionally that the pair (y_0, y_1) corresponds to $f_{i'}(x_0', x_1)$ where x_0' is the explicit representation of $\bar{x}_0'$ and $i' = |x_0'|$. If the proof given by π is not accepting, $\mathcal{F}$ sends `ideal error` to P_1. If the proof given by π is accepting, $\mathcal{F}$ sends y_0 to Sim. (Once output has been sent to either P_1 or Sim, all further input messages are ignored by $\mathcal{F}$.)
4. Sim *instructs* $\mathcal{F}$ *to answer* P_1*:* when Sim sends a message of the type `end` to $\mathcal{F}$, $\mathcal{F}$ sends y_1 to P_1. If inputs have not yet been received by $\mathcal{F}$ or `ideal error` was sent to P_1, $\mathcal{F}$ ignores message `end`.

5. *Outputs:* P_1 outputs whatever it received from $\mathcal{F}$ (i.e., `ideal error` or y_1). Sim may output an arbitrary (probabilistic polynomial-time computable) function of its auxiliary input z, the initial input x_0, and the output y_0 obtained from $\mathcal{F}$.

The ideal execution of $f = \{f_i\}_{i \in k}$ using implicit representation Rep and an executable PCP EX_π (with security parameter k, initial inputs (x_0, x_1) and auxiliary input z to Sim), denoted by
$\text{IDEAL}_{f,\mathsf{Sim},\mathsf{Rep},EX_\pi}(k, x_0, x_1, z)$ is the output pair of Sim and P_1 from the above execution.

Execution in the real world. We next consider the real world in which a real two-party protocol is executed (and there exists no trusted third party). Formally, a two-party protocol $\Pi = (\Pi_0, \Pi_1)$ is defined by two sets of instructions Π_0 and Π_1 for parties P_0 and P_1, respectively. A protocol is said to be polynomial time if the running times of both Π_0 and Π_1 are bounded by fixed polynomials in the security parameter k and in the size of the corresponding inputs.

Let f be as above and let Π be a PPT two-party protocol for computing f. In addition, assume that a non-uniform PPT adversary (with non-uniform input z) controls either P_0 or P_1. We describe the case in which the party P_1 is corrupted, and therefore we will denote it as $\mathcal{A} = P_1^\star$. The setting in which party P_0 is corrupted proceeds in a similar manner. The adversary $\mathcal{A} = P_1^\star$ on input x_1 starts by activating P_0, who uses his input x_0 and follows the protocol instructions Π_0 while the adversary $\mathcal{A} = P_1^\star$ follows any arbitrary polynomial time strategy. Upon the conclusion of this execution P_0 writes its output from the execution on its output-tape while the adversary $\mathcal{A} = P_1^\star$ may output any arbitrary polynomial time function of its view of the computation. To enforce the condition that `ideal error` never occurs in the output of the real execution we require that if `ideal error` does occur in the output of the honest party, the real-world execution will replace it with $\perp$. The real-world execution of Π (with security parameter k, initial inputs (x_0, x_1), and auxiliary input z to the adversary $\mathcal{A} = P_1^\star$), denoted by $\text{REAL}_{\Pi,P_1^\star}(k, x_0, x_1, z)$, is defined as the output pair of P_0 and $\mathcal{A} = P_1^\star$, resulting from the above process.

Security as emulation of real-world attacks in the ideal world. We can now define security of protocols. Loosely speaking, a 2-party protocol Π is 1-sided input-size secure if there exist Rep and EX_π such that for every real-world PPT adversary $\mathcal{A} = P^\star$, there exists an ideal-world PPT adversary Sim such that for all pairs of initial inputs (x_0, x_1), the outcome of the ideal execution using Rep and EX_π with adversary Sim is computationally indistinguishable from the outcome of a real protocol execution with $\mathcal{A} = P^\star$. We now present a formal definition.

Definition 2. *Let f and Π be as above. Π is said to securely compute f if there exists an executable PCP EX_π and an implicit representation* Rep *such that for every $b \in \{0, 1\}$ and every real-world non-uniform PPT adversary $\mathcal{A} = P^\star$ controlling party P_b there exists an ideal-world non-uniform probabilistic expected*

polynomial-time adversary Sim *controlling* P_b *such that*

$$\{\mathrm{IDEAL}_{f,\mathsf{Sim},\mathsf{Rep},EX_\pi}(k,x_0,x_1,z)\}_{k\in\mathcal{N};z\in\{0,1\}^*;x_0\in\{0,1\}^{\mathsf{poly}(k)};x_1\in\{0,1\}^k} \approx$$

$$\{\mathrm{REAL}_{\Pi,P^\star}(k,x_0,x_1,z)\}_{k\in\mathcal{N};z\in\{0,1\}^*;x_0\in\{0,1\}^{\mathsf{poly}(k)};x_1\in\{0,1\}^k}.$$

A sanity check. We note that our definition implies standard 2PC (with NBB simulator) when the input size is a fixed polynomial: in that case the implicit input is equivalent to explicit input (one can efficiently extract all the bits) and the unconditional proofs ensure the correctness.

Discussion. This definition has all the desirable properties mentioned in Section 1. First, it is clear that no information is revealed to the adversary beside the output y_b. Moreover, as long as `ideal error` does not occur, it is clear by unconditional soundness of EX_π that the output is indeed equal to $f(x_0,x_1)$ for the input implicitly defined by $\bar{x_0}$. (Again, since the real protocol cannot output `ideal error`, any attack on a real protocol translates into an ideal attack in which `ideal error` does not occur.) We obtain the second property (unconditional security) directly because EX_π is unconditionally sound. The third property follows because the ideal adversary must be efficient, and $\mathcal{F}$ just runs the circuit received from Sim. (Efficient algorithms cannot produce inefficient circuits.)

3 Realizing Input-Size Hiding Secure 2-Party Computation

Here we show that input-size hiding is possible. We begin by introducing the tools that are used in our construction. Then we give an informal description of our protocol, followed by a more formal specification.

Special commitment schemes. We will require a special type of commitment scheme that we call "size hiding", which will allow the sender to commit to a string and later to open only some bits of the string, in such a way that the commitment (and opening) does not reveal the size of the committed message. We will denote it by $(\mathsf{Gen},\mathsf{Com},\mathsf{Dec},\mathsf{Ver})$ where Gen is run by the receiver to generate parameters[13], Com commits to a string whose length is not a priori bounded, Dec reveals a bit of the committed string and Ver verifies the opening of a bit. A size-hiding commitment scheme can be constructed by using any commitment scheme along with a Merkle tree. One can also use a zero-knowledge[14] set scheme [MRK03]. For more details see the full version.

Error-correcting codes. We will use error-correcting codes (ECC) with constant distance. See the full version.

[13] Note that this is a 2-message commitment rather than a CRS-model commitment, so the hiding properties must hold even against adversarially chosen parameters.

[14] Actually indistinguishability as discussed in [CDV06] is sufficient here.

ZKUAQKs. We use the standard definitions (see the full version) of interactive proof systems, zero knowledge (ZK) and proofs of knowledge. We also use zero-knowledge universal arguments of quasi knowledge (ZKUAQKs) as introduced by [CV12]. In the full version we give the definitions introduced in previous work for zero knowledge for interactive arguments, for the proof of knowledge property for a universal argument (UA) and for quasi-knowledge for universal arguments. Informally, a universal argument of quasi knowledge (UAQK) is a universal argument with a special proof of knowledge property. Being a universal argument, it can be used to prove that a machine on input a certain string produces a certain output in a given number of steps T. The communication complexity and the running time of the verifier do not depend on T, which means that one can have a polynomial-time verifier even when T and the witness used by the prover are superpolynomial in the size of the statement. The special proof of knowledge property guarantees that for any polynomial time adversarial prover there always exists an extractor that runs in expected polynomial time and outputs bits of a valid witness. Moreover if the success probability of the prover is non-negligible, then for any polynomially computable set of indexes Φ, the extractor queried on each input $i \in \Phi$ with overwhelming probability outputs the i-th bit of a valid witness.

[CV12] gives a constant-round construction of a ZKUAQK based on the existence of CRHFs, building on the zero-knowledge universal argument of [Bar04]. (Since a universal argument is an interactive argument, the definition of a ZKUAQK is simply a UAQK which also satisfies the ZK property.) Indeed by plugging the zero-knowledge UA of [Bar04] in the witness indistinguishable UAQK of [CV12] (this works since ZK implies WI) we have that the quasi knowledge property follows directly. To see the ZK property, note that in the protocol in [CV12] the prover runs a ZK verifier, sends some commitments, and runs several ZK proofs sequentially. Thus, a simulator can easily run such steps by making use of the the simulator for the ZK proofs and sending commitments of random messages. Zero knowledge therefore follows from sequential composition of the ZK property of the UA [Bar04], and from the hiding property of the commitment.

Fully homomorphic encryption [Gen09]. A fully homomorphic encryption (FHE) scheme is a semantically secure encryption scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ augmented with an additional algorithm Eval that allows for computations over encrypted ciphertexts. See the full version for details.

Probabilistic checkable proofs of proximity. A "PCP of proximity" (PCPP) proof [BSGH+06] is a relaxation of a standard PCP, that only verifies that the input is close to an element of the language. It has the property that the PCP verifier needs only the description of the Turing machine $\mathcal{M}$ deciding the language and oracle access to bits of the input (x_0, x_1) of $\mathcal{M}$ and to bits of the PCP proof π. See the full version for details.

3.1 High-Level Overview of the Protocol

Here we describe a protocol which provides size-hiding for P_0 for functions in which only P_1 receives output. See the full version for protocols which allow either or both parties to receive output. At a high level, our protocol follows the outline described in Section 1 and consists of the following 3 steps:

1st Part. In the first part, P_1 sends to P_0 parameters for a size-hiding string commitment scheme. Then P_0 uses this scheme in order to send to P_1 the commitment com of its input x_0 expanded by means of an error correcting code. P_0 then proves "quasi knowledge" of the committed value. This is done by using a ZKUAQK and ends the first part. The above first part of the protocol implements the idea of P_0 committing to its input along with a zero knowledge argument of "quasi knowledge". Looking ahead the simulator will use this argument to extract from $P_0^\star$ a reliable implicit representation of the committed input. The ECC guarantees that the explicit input will be well defined and correct even if the extracted circuit is incorrect in a few positions. (Recall that quasi knowledge allows the extracted circuits to be incorrect on any negligible fraction of the positions.)

2nd Part. In the second part of the protocol, P_1 sends the public key for FHE. This key will later be used by P_0 to perform the correct computation on encrypted data. P_1 also proves (in ZK) knowledge of the corresponding secret key, and then sends an encryption e of its input x_1. Intuitively, the ciphertext e combined with the proof of knowledge of the secret key guarantees that P_1 "knows" his input. Looking ahead to the proof, this is useful because it will allow the simulator to decrypt e and extract $P_1^\star$'s input. This ends this part of the protocol, which focuses on P_1 committing to its input in a way that is both extractable and usable to perform computations over encrypted data.

3rd Part. P_0 uses the FHE evaluation algorithm and the ciphertext e received from P_1 in order to compute the encryption e' of P_1's output according to the function f with inputs x_0 and x_1. Then e' is sent to P_1. P_0 also computes a PCPP proof π proving that e' has been correctly computed by applying the function $f(x_0, \cdot)$ to e. Then P_0 proves by means of a ZKUAQK, "quasi knowledge" of a value x_0 that is committed (after ECC expansion) in com and of a PCPP proof π as described above corresponding to that x_0. Finally, P_1 decrypts e' therefore obtaining its output. This part of the protocol thereby focuses on P_0 computing P_1's output e' and sending it to P_1, and proving with a ZKUAQK that e' was computed correctly and there is a PCPP proof confirming it. Here, circuit privacy of the encryption scheme guarantees that P_1 learns nothing about P_0's input. "Quasi knowledge" of the ZKUAQK allows the simulator to obtain from $P_1^\star$ a circuit representation of a PCPP. Very roughly, the PCPP properties allow the functionality to verify that e' is correct given only this circuit representation of the PCPP and the implicit representation of x_0 that we extracted in step 1. This is possible because the verifier a of PCPP (i.e., the functionality, in our case) only needs access to few bits of the the PCP proof and only part of the statement.

3.2 Our Protocol for Input-Size Hiding Secure 2PC

We now give a specification of our protocol. For function family $\{f_i\}$ defined by circuit $M(\cdot)$, our protocol appears in Fig. 1 and uses the following ingredients, as defined above. (For extensions, see the full version.)

- ▷ A circuit private FHE scheme $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$.
- ▷ A size-hiding string commitment scheme $(\mathsf{Gen}, \mathsf{Com}, \mathsf{Dec}, \mathsf{Ver})$.
- ▷ An error correcting code $(\mathsf{ECC}, \mathsf{ECCDec})$ with polynomial expansion and constant distance δ.
- ▷ A PCPP $(\mathsf{Prove}_{\mathsf{PCPP}}, \mathsf{Verify}_{\mathsf{PCPP}})$ with proximity parameter δ for the following pair language: $((i, e, e', \mathsf{pk}), x') \in \mathcal{L}_{\mathsf{PCPP}}$ if there exists randomness r' and input x such that $e' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, e, r') \wedge x' = \mathsf{ECC}(x) \wedge i = |x| \wedge f_i$ is as defined by $M \wedge C$ is the circuit for $f_i(x, \cdot)$.
- ▷ A ZKPoK $(\mathsf{P}_{\mathsf{POK}}, \mathsf{V}_{\mathsf{POK}})$ for the following relation: $(\mathsf{pk}, \sigma) \in R_3$ iff $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\sigma)$.
- ▷ A ZKUAQK $(\mathsf{Prover}_{\mathsf{UAQK}}, \mathsf{Verifier}_{\mathsf{UAQK}})$ for the following relations.

  R_1**:** accept $((\mathsf{com}, \eta), (i, x', x, \mathsf{dec}))$ iff $\mathsf{Ver}(\eta, \mathsf{com}, \mathsf{dec}, x') = 1 \wedge x' = \mathsf{ECC}(x) \wedge |x| = i)$. (By $\mathsf{Ver}(\eta, \mathsf{com}, \mathsf{dec}, x') = 1$, we mean that verification succeeds on all bits of x'.) We denote by L_1 the corresponding language.

  R_2**:** accept $((\mathsf{com}, \eta, e, e', \mathsf{pk}), (i, \mathsf{dec}, x', r', x, r'', \pi))$ iff $\mathsf{Ver}(\eta, \mathsf{com}, \mathsf{dec}, x') = 1$ and $\pi = \mathsf{Prove}_{\mathsf{PCPP}}(((i, e, e', \mathsf{pk}), x'), (r', x); r'')$ is an honestly generated proof for $((i, e, e', \mathsf{pk}), x')) \in \mathcal{L}_{\mathsf{PCPP}}$ (generated using randomness r''). We denote by L_2 the corresponding language.

Theorem 2. *The protocol in Fig. 1 securely computes function family f defined by M, under Definition 2 (i.e., for one-sided size-hiding secure 2-party computation).*

Here we give the main ideas for the proof. For more details, see the full version.

First, we need to describe an implicit representation and an executable PCP scheme that we will use for the ideal execution.

The implicit representation. In the ideal model we require that the implicit representation uniquely determines the explicit input, i.e. there must exist some (potentially inefficient) decoding algorithm Decode that maps every implicit representation x' to some explicit input. In our construction, this will consist of taking an implicit circuit representation x', extracting all the bits of the corresponding string $\tilde{x}$ by evaluating x' on $1, \ldots i'$, where i' is the length of an encoding under ECC of a string of length i, and then running the ECC decoding algorithm $\mathsf{ECCDec}(\tilde{x})$ to produce x. $\mathsf{Rep}(x)$ simply outputs a circuit representation of $\mathsf{ECC}(x)$.

The executable PCP. The ideal world requires an executable PCP for checking membership of a statement in the language: $(i, x'_0, y, x_1) \in \mathcal{L}_{\mathsf{Ideal}}(\mathsf{Decode})$ if there exists x_0 such that $\mathsf{Decode}(x'_0) = x_0, \wedge |x_0| = i \wedge y = f_i(x_0, x_1)$. The executable PCP EX_π will be instantiated through a circuit representation of a PCPP proof. Indeed proofs given by a PCPP are unconditionally sound but of size polynomial in the length of the witness and this could be too long. However, a circuit representation of a PCPP proof can have short size and still can be

Players: P_0 and P_1 with private inputs x_0 and x_1 respectively.

Common input: the description of the machine $\mathcal{M}$ describing the function family $f = \{f_i\}_{i \in \mathcal{N}}$.

1. $P_1 \to P_0$: P_1 picks random coins σ, runs $\eta \leftarrow \mathsf{Gen}(\sigma)$ and send η to P_0.
2. $P_0 \to P_1$: P_0 computes $(\mathsf{com}, \mathsf{dec}) = \mathsf{Com}(\eta, x_0' = \mathsf{ECC}(x_0), \mathtt{r})$ with random coins $\mathtt{r}$ and sends com to P_1.
3. $P_0 \leftrightarrow P_1$: P_0 and P_1 run the ZKUAQK $\langle \mathsf{Prover}_{\mathsf{UAQK}}((\mathsf{com}, \eta), (|x_0|, x_0', x_0, \mathsf{dec})), \mathsf{Verifier}_{\mathsf{UAQK}}((\mathsf{com}, \eta))\rangle$ for language L_1 to prove quasi knowledge of an opening to com that can be correctly decoded. P_1 aborts if $\mathsf{Verifier}_{\mathsf{UAQK}}$ rejects.
4. $P_1 \to P_0$: P_1 chooses random coins σ' and runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\sigma')$. P_1 sends pk to P_0.
5. $P_1 \leftrightarrow P_0$: P_1 and P_0 run the ZKPoK $\langle \mathsf{P}_{\mathsf{POK}}(\mathsf{pk}, \sigma'), \mathsf{V}_{\mathsf{POK}}(\mathsf{pk})\rangle$ to prove knowledge of the coins used to generate pk. P_0 aborts if $\mathsf{V}_{\mathsf{POK}}$ does not accept.
6. $P_1 \to P_0$: P_1 picks random coins r_e and computes $e \leftarrow \mathsf{Enc}(\mathsf{pk}, x_1, r_e)$. P_1 sends e to P_0.
7. $P_0 \to P_1$: P_0 computes the circuit C for $f_{|x_0|}(x_0, \cdot)$, chooses a sufficiently long random string r' and then computes $e' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, e, r')$. P_0 sends e' to P_1.
8. $P_0 \leftrightarrow P_1$: P_0 runs $\pi \leftarrow \mathsf{Prove}_{\mathsf{PCPP}}((|x_0|, e, e', \mathsf{pk}, x_0'), (r', x_0); r'')$ after picking random coins r'', thereby generating a PCPP π showing that e' is correctly computed from e, pk and x_0 such that $x_0' = \mathsf{ECC}(x_0)$. Then P_0 and P_1 run the ZKUAQK $\langle \mathsf{Prover}_{\mathsf{UAQK}}((\mathsf{com}, \eta, e, e', \mathsf{pk}), (|x_0|, \mathsf{dec}, x_0', r', x_0, r'', \pi)), \mathsf{Verifier}_{\mathsf{UAQK}}(\mathsf{com}, \eta, e, e', \mathsf{pk})\rangle$ for language L_2 to prove quasi knowledge of an opening x_0', dec to com and of a PCPP proof π computed with randomness r'' showing that e' is correctly computed with randomness r' and inputs e, pk and x_0 such that $x_0' = \mathsf{ECC}(x_0)$. P_1 aborts if $\mathsf{Verifier}_{\mathsf{UAQK}}$ rejects.
9. P_1: P_1 runs $y = \mathsf{Dec}(\mathsf{sk}, e')$ and outputs y.

Fig. 1. A secure protocol for with input-size hiding for P_0 and output for P_1

easily used to unconditionally verify the validity of the intended statement since it can be executed a polynomial number (independent of the witness size) of times to obtain the bits of the PCPP proof needed by the PCPP verifier. In fact, the proof system that we use will be somewhat more involved: the proof will include additional values σ, e, e', x_0'' as well as the circuit representing the PCPP $C_{\pi'}$. Formally, we can define the required executable proof by defining its verification algorithm.

The verifier $\mathsf{Verify}((i, x_0', y, x_1), \bar{\pi})$ for the executable proof proceeds as follows (recall that (i, x_0', C) will be the input received from the player that aims to hide the size of his inputs, and $(y, \bar{\pi})$ is the output of $C(x_1)$)

1. Parse $\bar{\pi} = (\sigma, e, e', x_0'', C_{\pi'})$, and view both x_0' and x_0'' as circuit representations of i'-bit strings (again, i' is the length of an encoded string of length i using ECC). If $\bar{\pi}$ cannot be parsed this way, output 0.
2. Choose k random locations $i_1, \ldots, i_k \in \{0, \ldots, i' - 1\}$, and halt and output 0 if $x_0'(i) \neq x_0''(i)$ for any $i \in i_1, \ldots, i_k$.

3. Compute $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\sigma)$, and halt and output 0 if $\mathsf{Dec}(\mathsf{sk}, e) \neq x_1$ or $\mathsf{Dec}(\mathsf{sk}, e') \neq y$.
4. Run the PCPP verifier $\mathsf{Verify}_{\mathsf{PCPP}}((i, e, e', \mathsf{pk}), x_0')$ for language $\mathcal{L}_{\mathsf{PCPP}}$, to obtain index sets I_π, I_z and decision circuit D. Compute $x_0'(j)$ on each index $j \in I_z$ and compute $C_{\pi'}(j)$ on each index $j \in I_\pi'$; call the resulting list of bits X. Finally output the result of $D(X)$.

Note that an honest prover when given x_0 can efficiently generate an accepting proof for each valid x_1, y. Note also that PCPP proofs require the verifier to access only a few bits of the statement and the proof in order to be convinced, so we obtain an efficient verifier.

Lemma 1. *The above proof system is unconditionally sound for the language* $\mathcal{L}_{\mathsf{Ideal}}(\mathsf{Decode})$.

Proof. The soundness of the PCPP guarantees that x_0' is close to a valid codeword. Then we have that if $\mathsf{ECCDec}(x_0') \neq \mathsf{ECCDec}(x_0'')$ then the probability that we sample k random positions without finding any difference among x_0' and x_0'' is negligible. Then, by soundness of the PCPP, if the verifier accepts, we know that with all but negligible probability $e' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, e, r')$ for some r', where C is the circuit for $f_i(\mathsf{ECCDec}(x_0'), \cdot)$ and $i = |x_0|$. Finally, the verifier will generate $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\sigma)$ and check that $x_1 = \mathsf{Dec}(\mathsf{sk}, e)$ and $y = \mathsf{Dec}(\mathsf{sk}, e')$. By the circuit privacy property, we then get that the distribution of e' is statistically close to $\mathsf{Enc}(\mathsf{pk}, C(x_1))$ and if we let $x_0 = \mathsf{ECCDec}(x_0'') = \mathsf{ECCDec}(x_0')$, this means that $\mathsf{Dec}(\mathsf{sk}, e') = C(x_1) = f_i(x_0, x_1) = y$ as desired.

Security with Corrupt P_1. Now we are ready to sketch the proof of security for the case where P_1 is corrupt. For any real-world adversary P_1^*, we will show an ideal-world adversary Sim such that the ideal and real-world outputs are indistinguishable. Sim will simulate P_1^* internally. It receives parameters for the commitment scheme from P_1^* in step 1, commits to 0 in step 2, and runs the ZK simulator for the UAQK with P_1^* in step 3. In step 4, it receives pk, and in step 5 it runs the PoK extractor to extract σ'. Then in step 6 it receives e and computes $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\sigma')$, and $x_1 = \mathsf{Dec}(\mathsf{sk}, e)$. It sends x_1 to the functionality and receives y. In step 7, it sends $e' = \mathsf{Enc}(\mathsf{pk}, y)$, and in step 8 it runs the ZK simulator for the UAQK.

We argue that the ideal-world output with this simulator is indistinguishable from the real-world output with P_1^* through a series of games. We define Game 0 to be the real game, and then proceed as follows:

- Game 1: UAQK Simulator. This is identical to the real game except that in steps 3 and 8, P_0 uses the ZK simulator. (This is indistinguishable by the Zero Knowledge property of the ZKUAQK.)
- Game 2: Simulated commitment. This is identical to game 1 except that in step 2 P_0 forms a commitment to 0. (This is indistinguishable by hiding property of the commitment scheme.)

- Game 3: Extracting σ'. This is identical to game 2 except that in step 5 P_0 uses the PoK extractor to extract σ', and aborts if $\mathsf{KeyGen}(\sigma') = (\mathsf{pk}', \mathsf{sk}')$ such that $\mathsf{pk}' \neq \mathsf{pk}$. (This is indistinguishable by the PoK property.)
- Game 4: Decrypting e. This is identical to game 3 except that in step 7 P_0 computes $x_1 = \mathsf{Dec}(\mathsf{sk}, e)$ and then sends $e' = \mathsf{Enc}(\mathsf{pk}, f_{|x_0|}(x_0, x_1))$. (This is indistinguishable by the circuit privacy property of the FHE scheme.)
- Game 5: Ideal Game. The only difference is that in Game 4 P_0 computes $f_{|x_0|}(x_0, x_1)$ and encrypts it, while in the ideal game Sim sends x_1 to the ideal functionality, receives $y = f_{|x_0|}(x_0, x_1)$, and encrypts that, so the two games are identical.

Corrupt P_0. Finally we consider the case when party P_0 is corrupted. For any real world adversary $\mathcal{A}$, we will show an ideal-world adversary Sim such that the ideal and real-world outputs are indistinguishable. Here we present the intuition behind the proof. (For a more detailed treatment, see the full version.)

At a high level the idea is that the simulator will use the extractor for the UAQK in step 3 to construct an implicit representation x_0' of x_0. Then, for C it will construct a circuit which on input x_1 1) continues to run $\mathcal{A}$ with an honest P_1 using input x_1 in steps 4-7, 2) decrypts the ciphertext e' received in step 7 to obtain y, and 3) uses the extractor for the UAQK in step 8 to construct an implicit representation of the statistically sound proof (in particular of x_0'' and the PCPP π'). Finally, in order to ensure that the protocol is aborted with the same probability in both worlds, the simulator will first run $\mathcal{A}$ once with honest P_1; it will rewind to step 3 and construct x_0' and C as described above only if this first run is successful. Since the simulator does not know x_1, it will instead use $x_1 = 0$ for this run, and we will argue that the result will be indistinguishable.

The ideal world never produces `ideal error` *except with negligible probability.* Recall that to verify the proof π produced by $C(x_1)$, the functionality will have to check several things: 1) that x_0' and x_0'' are close, 2) that e, e' decrypt to x_1 and y, and 3) that the PCPP π' verifies for the statement $(i, e, e', \mathsf{pk}), x_0''$.

Thus, we need only show that these three checks will succeed. The argument goes as follows. First, we argue that x_0'' extracted in step 8 must be very close to x_0' extracted in step 3. If not, we could use the UAQK extractors to extract some pair of bits that differ along with a pair of valid openings for those bits, and thus produce a contradiction to the binding property. The second check is clearly satisfied by the way that e and y are computed. Finally, the quasi-knowledge property of the UAQK implies that almost all of the bits of the extracted PCPP will be correct, so the verification should succeed with all but negligible probability.

P_1's output in the two games is indistinguishable given that the functionality does not send `ideal error`. The issue here is that the functionality will run the circuit C to obtain P_1's output, which is essentially like rewinding and rerunning steps 4-8 of the protocol. (This is an issue because Sim will produce its ideal output using $\mathcal{A}$'s initial run, so we need to make sure that P_1's output is still consistent with this initial run.) Thus, we have to argue that rewinding to step 4 can not change P_1's output. Above, we argued that C produces a valid proof

that $y = f_{|x_0|}(x_0, x_1)$ where x_0 is as defined by the decoding of the implicit string produced by the UAQK extractor in step 3. Similarly, we can argue that if we extract from the UAQK in the first run-through, we can also produce a valid proof that y produced in that run is also equal to $y = f_{|x_0|}(x_0, x_1)$ for the same x_0. Finally, by the unconditional soundness of the proof described in Section 3.2, we conclude that P_1's output y will be the same in both games.
The adversary's output in the two games is indistinguishable. The main difference is that in the real game, $\mathcal{A}$ is interacting with P_1 whose input is x_1, while in the ideal game, Sim's output is what $\mathcal{A}$ produces when run with P_1 whose input is 0. This follows fairly directly from semantic security. Note that we also need zero knowledge to ensure that the proof in step 5 does not reveal anything about sk or x_1.

A technical issue. There is one technical issue that occurs because we are building on UAQKs. The issue is that for our simulator to work, we need to ensure that the UAQK extractors run in polynomial time, and succeed with overwhelming probability. By definition, we are guaranteed that from a prover with success probability p, a UAQK extractor will run in time $1/p$ and successfully extract a witness with overwhelming probability assuming p is non-negligible. Here, we need to ensure that the UAQK is given a prover that succeeds with non-negligible probability. (Unless we have a real world adversary that aborts with all but negligible probability - in that case our simulator will also abort with all but negligible probability.) The issue is that that the adversary's probability of aborting may depend on the random coins that P_1 uses up until that point, and in particular on the randomness used in forming the key pair pk, sk and the encryption e. Recall that our simulator uses the first run, with $e = \mathsf{Enc}(\mathsf{pk}, 0, \cdot)$ to determine whether to abort, and then rewinds and gives the ideal functionality a circuit that extracts from a prover who is sent $e = \mathsf{Enc}(\mathsf{pk}, x_1, \cdot)$ (under a fresh public key). It is possible that for some values of e, $\mathcal{A}$ always aborts; we may get unlucky and in the first run get an e on which $\mathcal{A}$ successfully completes the protocol (so we continue and form C), and then when C is run it ends up with an e for which $\mathcal{A}$ aborts with high probability thus causing the extractor to fail and the circuit C to not produce a valid proof π. However, if there is a non-negligible probability with which $\mathcal{A}$ does not abort on the first run, then there must be a non-negligible chance that it will produce valid proofs on the rewinding as well. (This follows from semantic security because the only difference in the runs is the value encrypted in e.) Thus, we will rewind many times, find one on which $\mathcal{A}$ does produce valid proofs, and extract from that run. To determine how many times to rewind (so that we can argue that at least one will be successful but the process still runs in expected polynomial time), we will use estimation techniques from [CV12]. For a more detailed proof see the full version.

Acknowledgments. Research supported in part by MIUR Project PRIN "GenData 2020", NSF grants CCF-0916574; IIS-1065276; CCF-1016540; CNS-1118126; CNS-1136174; US-Israel BSF grant 2008411, OKAWA Foundation Research Award, IBM Faculty Research Award, Xerox Faculty Research Award, B. John Garrick Founda-

tion Award, Teradata Research Award, and Lockheed-Martin Corporation Research Award. This material is also based upon work supported by the Defense Advanced Research Projects Agency through the U.S. Office of Naval Research under Contract N00014-11-1-0392. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[ACT11] Ateniese, G., De Cristofaro, E., Tsudik, G.: (If) size matters: size-hiding private set intersection. In: Catalano, D., Fazio, N., Gennaro, R., Nicolosi, A. (eds.) PKC 2011. LNCS, vol. 6571, pp. 156–173. Springer, Heidelberg (2011)

[Bar04] Barak, B.: Non-black-box techniques in cryptography. Ph.D Thesis (2004)

[Bea92] Beaver, D.: Foundations of secure interactive computing. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 377–391. Springer, Heidelberg (1992)

[BR93] Bellare, M., Rogaway, P.: Random oracles are practical: a paradigm for designing efficient protocols. In: ACM Conference on Computer and Communications Security, pp. 62–73. ACM (1993)

[BSGH+06] Ben-Sasson, E., Goldreich, O., Harsha, P., Sudan, M., Vadhan, S.P.: Robust pcps of proximity, shorter pcps, and applications to coding. SIAM J. Comput. **36**(4), 889–974 (2006)

[Can03] Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003). http://eprint.iacr.org/

[Can05] Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. Cryptology ePrint Archive, Report 2000/67 (version 13 Dec 2005) (2005). http://eprint.iacr.org/2000/067/20051214:064128

[CDV06] Catalano, D., Dodis, Y., Visconti, I.: Mercurial commitments: minimal assumptions and efficient constructions. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 120–144. Springer, Heidelberg (2006)

[CFT13] De Cristofaro, E., Faber, S., Tsudik, G.: Secure genomic testing with size- and position-hiding private substring matching. In: Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013, Berlin, Germany, November 4, 2013, pp. 107–118 (2013)

[CGH98] Canetti, R., Goldreich, O., Halevi, S.: The random oracle methodology, revisited (preliminary version). In: Vitter, J.S. (ed.), Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, 23–26 May, 1998, pp. 209–218. ACM (1998)

[Cle86] Cleve, R.: Limits on the security of coin flips when half the processors are faulty (extended abstract). In: STOC, pp. 364–369. ACM (1986)

[CV12] Chase, M., Visconti, I.: Secure database commitments and universal arguments of quasi knowledge. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 236–254. Springer, Heidelberg (2012)

[Dam92] Damgård, I.B.: Towards practical public key systems secure against chosen ciphertext attacks. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 445–456. Springer, Heidelberg (1992)

[DN92] Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In: Brickell, E.F. (ed.) CRYPTO 1992. LNCS, vol. 740, pp. 139–147. Springer, Heidelberg (1993)

[Gen09] Gentry, C.: Fully homomorphic encryption using ideal lattices. In: STOC, pp. 169–178. ACM (2009)

[GL91] Goldwasser, S., Levin, L.A.: Fair computation of general functions in presence of immoral majority. In: Menezes, A., Vanstone, S.A. (eds.) CRYPTO 1990. LNCS, vol. 537, pp. 77–93. Springer, Heidelberg (1991)

[GMW87] Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: STOC, pp. 218–229. ACM (1987)

[Gol04] Goldreich, O.: Foundations of cryptography, vol. 2: Basic applications (2004)

[IP07] Ishai, Y., Paskin, A.: Evaluating branching programs on encrypted data. In: Vadhan, S.P. (ed.) TCC 2007. LNCS, vol. 4392, pp. 575–594. Springer, Heidelberg (2007)

[LNO13] Lindell, Y., Nissim, K., Orlandi, C.: Hiding the input-size in secure two-party computation. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part II. LNCS, vol. 8270, pp. 421–440. Springer, Heidelberg (2013)

[MR92] Micali, S., Rogaway, P.: Secure computation. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 392–404. Springer, Heidelberg (1992)

[MRK03] Micali, S., Rabin, M.O., Kilian, J.: Zero-knowledge sets. In: FOCS, pp. 80–91. IEEE Computer Society (2003)

[Nao03] Naor, M.: On cryptographic assumptions and challenges. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 96–109. Springer, Heidelberg (2003)