Going Beyond Dual Execution: MPC for Functions with Efficient Verification

Carmit Hazay[1], abhi shelat[2], and Muthuramakrishnan Venkitasubramaniam[3]

[1] Bar-Ilan University
carmit.hazay@biu.ac.il
[2] Northeastern University
abhi@neu.edu
[3] University of Rochester
muthuv@cs.rochester.edu

Abstract. The *dual execution* paradigm of Mohassel and Franklin (PKC'06) and Huang, Katz and Evans (IEEE '12) shows how to achieve the notion of *1-bit leakage* security at roughly twice the cost of semi-honest security for the special case of *two-party secure computation*. To date, there are no multi-party computation (MPC) protocols that offer such a strong trade-off between security and semi-honest performance.

Our main result is to address this shortcoming by designing 1-bit leakage protocols for the multi-party setting, albeit for a special class of functions. We say that function $f(x,y)$ is *efficiently verifiable by* g if the running time of g is always smaller than f and $g(x,y,z) = 1$ if and only if $f(x,y) = z$.

In the two-party setting, we first improve *dual execution* by observing that the "second execution" can be an evaluation of g instead of f, and that by definition, the evaluation of g is asymptotically more efficient.

Our main MPC result is to construct a 1-bit leakage protocol for such functions from any passive protocol for f that is *secure up to additive errors* and any active protocol for g. An important result by Genkin et al. (STOC '14) shows how the classic protocols by Goldreich et al. (STOC '87) and Ben-Or et al. (STOC '88) naturally support this property, which allows to instantiate our compiler with two-party and multi-party protocols.

A key technical result we prove is that the passive protocol for distributed garbling due to Beaver et al. (STOC '90) is in fact *secure up to additive errors* against malicious adversaries, thereby, yielding another powerful instantiation of our paradigm in the constant-round multi-party setting.

As another concrete example of instantiating our approach, we present a novel protocol for computing *perfect matching* that is secure in the 1-bit leakage model and whose communication complexity is less than the honest-but-curious implementations of textbook algorithms for perfect matching.

Keywords: Secure Computation, Semi-honest Security, Dual Execution, Greedy Algorithms.

1 Introduction

Current approaches for designing secure two-party (2PC) and multi-party (MPC) protocols follow a generic *compiler* pattern that transforms the function f into either a Boolean circuit or a RAM program and then applies a method to securely evaluate each gate or RAM operation in the program. This approach has been successful at evaluating many interesting functions such as the AES function [MOR03, MNPS04, BNP08, LP12, NO09, NNOB12, KSS12, RHH14], edit distance [HEKM11], textbook RSA operations [KSMB13] and graph algorithms [LHS+14, KS14].

The main result in this paper is to introduce a new method for constructing secure computation protocols that exploits the properties of the function of interest f. Our resulting protocols are secure in the one-bit leakage security model[4] and are asymptotically more efficient than using generic compiler-based techniques. In particular, we study secure computation protocols for a natural class of functions f which have *efficient verifiers*, i.e., given x, y, z, it is more efficient to verify $f(x,y) = z$ than to compute $f(x,y)$.

Notable examples in this class include: (a) Frievald [Fre77]'s celebrated technique verifies a matrix multiplication in time $O(\ell^2)$ whereas the schoolbook algorithms for matrix multiplication require $O(\ell^3)$ operations, (b) although it takes $O(E^2V)$ or $O(V^3)$ to compute a maxflow for a graph $G = (V,E)$, given the flow f, one can verify its min-cut in time $O(E+V)$, (c) a minimum spanning tree can be verified in linear time [Kin95], (d) after solving any linear program, slack variables can be used to verify the optimal against the input constraints. Another interesting class includes sampling from complex distributions via rejection sampling. Here the procedure is to sample uniformly and apply a predicate to the result until the sample passes. Verification of the sample, on the other hand, requires only 1 application of the predicate, and can thus be asymptotically faster. Moreover, in a secure computation context, the parties' inputs are simply their random coins. Thus, 1-bit leakage can have essentially no security implication since the adversary can easily guess. One such example is sampling from the set of RSA moduli (product of two primes, as required for threshold RSA crypto systems). The best methods to (securely) sample [FLOP18] require roughly $O(\log^2(n))$ attempts to chose an $O(n)$-bit prime and then perform a multiplication and apply a bi-primality test, whereas verifying takes 1 multiplication and 1 bi-primality test.

On the other hand, we remark that there are some situations where leaking even 1 but could be harmful. For instance, in case of functions with one input bit, the adversary can leak the entire input of the honest party. Another example is when the secure computation is involved with medical or financial data where the attacker can extract a high order bit of information, such as whether an employee earns more than a certain amount of money or whether the DNA string includes a certain gene that causes a particular disease.

In the 1-bit leakage model, where the adversary is allowed to learn at most 1-bit of the honest party's input, the dual execution paradigm of Mohassel and Franklin [MF06]

[4] Where the security definition allows the adversary to submit an arbitrary leakage predicate such that the honest party learns its output condition on whether the predicate is true when applied on the parties' inputs.

is the most efficient black-box protocol to date, incurring twice the cost of the passive Yao protocol. Even though protocols for full security [WRK17a, WRK17b, KRRW18] have improved significantly over the recent past, dual execution offers much better concrete efficiency at low overhead without significantly compromising security [HKE12]. Security with 1-bit leakage is arguably sufficient in many cases, as the leakage legitimately allowed by the ideal functionality is often more significant than the very limited type of leakage in this model. For instance, secure computation with one bit of leakage has been adopted by Calctopia [cal] to perform secure computation on data stored in local private spreadsheets of the parties. In this paper, for the special class of functions with efficient verifiers, we are able to show an overhead that is *less than twice* the cost of passive protocol.

1-bit leakage for 2-party. To introduce the power of efficiently verifiable f, our first result is a black-box two-party protocol that securely computes a function f in the *1-bit leakage model* with communication complexity roughly $p(|f|,\kappa) + p(|g|,\kappa) + poly(\kappa)$ where p represents the cost of Yao's *garbled encoding* of a function [Yao86], κ is a security parameter and $|f|$ and $|g|$ represent the size of the circuits that compute those functions respectively. Prior work requires $2p(|f|,\kappa) + poly(\kappa)$, and thus, our methods offer improvement (up to a factor of 2) for functions with asymptotically faster verification (e.g., matrix multiplication). Our main insight is to modify the *dual execution* technique introduced by Mohassel and Franklin [MF06] and refined by Huang, Katz and Evans [HKE12] to exploit the efficient verifier. Our analysis also shows that the secure equality test that is used at the end of both prior protocols is not necessary. While this result is purely a concrete improvement for a special class of functions, it introduces the insight needed for our main results in this paper. Importantly, we stress that the verification circuit g is never more complex than f, as in the worst case it can be instantiated with f. Our protocol takes advantage of the case when *the (multiplicative) complexity of g is smaller than f* where checking is often easier than computing.

1-bit leakage beyond 2-party boolean computations via dual execution. Our main result is to show how to extend the dual execution technique to multi-party protocols such as [GMW87] and [BMR90]. These are the first general class of black-box protocols in the multi-party setting to achieve security against 1-bit leakage *where the cost of the protocol is $(1+\epsilon)$ times the passive counterparts*. Our insight here is that while prior work on dual execution required "one-sided security" in the analysis (where one-sided security implies active security for one party and passive security for the other), we can instead rely on the weaker property of *additive independent errors*—a useful notion introduced by Genkin et al. in [GIP+14]. We remark that prior work on the paradigm of dual execution explicitly applies to boolean computation via garbled circuits and *only* for the two-party setting.

In the multi-party setting, the recent maliciously secure protocol of Wang et al. [WRK17b] relies on cut-and-choose mechanism and incurs $\Omega(s/\log|C|)$ overhead for a statistical parameter s and circuit C. A crucial idea in this family of works toward achieving active security is to generate authenticated triples. Our paradigm can be used to improve their overheads (at the price of 1-bit leakage and verifiable functions) by requiring fewer triples that are authenticated. Our computation requires (unauthenticated) triples (secure up to additive attacks) for computing the function and authen-

ticated triples for verifying the result. If we relax the black-box requirement, the work of [KPR18] achieves malicious security using tailor-made zero-knowledge proofs, but, still incurs a significant overhead over the passive protocol.

While the maliciously secure MPC protocols of [IPS08, IPS09, CGH+18] have constant communication overhead, their computational overhead is *super-constant* in the circuit size. Meanwhile, the work of [HSS17] introduces an asymptotically good protocol that achieves constant communication overhead over a passive protocol, but only for a constant number of parties due to the restrictions imposed by the IPS compiler. Our work (similar to the dual-execution paradigm) achieves both *constant computation and communication overheads* theoretically and concretely in the most general setting, albeit for a restricted class of functionalities, but in both honest and dishonest majority settings.

Finally, Hazay et al. [HIV17] demonstrate a non-interactive two-party protocol with leakage for arbitrary functions with communication overhead that is strictly greater than $1 + s/k$ where s and k are the statistical and computational security parameters. In contrast, our work can achieve overheads $1 + 1/n$ if verification is $O(1/n)$ simpler than computing (which is the case for many of our examples). Secondly, the computational overhead of our approach is also close to 1 while the work of [HIV17] is $O(logn)$ due to Shamir Sharing.

The additive security of [BMR90]-style garbling. The work of [GIP+14] allows us to instantiate our protocol with the [GMW87] protocol in the OT and OLE hybrids for Boolean and arithmetic functionalities. However, these protocols require more than a constant number of rounds. As a third contribution, we show that we can instantiate our framework with the distributed garbling (BMR) protocol due to Beaver, Micali and Rogaway [BMR90]. Specifically, a key technical lemma we prove is that the BMR protocol is secure up to additive independent errors on the internal wires of the circuit. We remark that this is the first constant-round protocol for which such a result has been established. Furthermore, this result is of independent interest as it can enable communication efficient multi-party protocols in the full security (i.e., no leakage) setting assuming the exitance of efficient Binary AMD circuit transformations.

Additive security of perfect matching. As a case study for practitioners of our technique, we present a protocol for the problem of computing a *perfect matching* in a graph. This considers a scenario where the edges of a graph are distributed between the participants. Secure perfect matching and its generalization maximum matching are useful for assigned drivers to passengers in ridesharing services and more generally resource allocation. We show how this protocol satisfies the property that an adversary is limited to forcing additive, input-independent errors, and thus, we can apply our technique. Our protocol is iterative and requires secure computation of smaller independent tasks, most notably, matrix multiplication and matrix inversion for which we use Beaver triples techniques similar to Mohossel and Zhang [MZ17]. The communication complexity of our protocol is $O(V^2 \log V)$ which is *asymptotically more efficient than even using passive generic MPC* on the best algorithm to compute perfect matching [Har06], which would result in communication $O(V^\omega)$ where ω is the matrix-multiplication exponent (3 in the case of schoolbook algorithm). Passive generic MPC techniques ap-

plied to other algorithms for computing matchings (e.g., flow-based algorithms) would require ORAM datastructures and would be even less efficient than $O(V^\omega)$.

Constant-overhead cryptography A fundamental goal in cryptography is to obtain constant-overhead in computation (and consequently communication as well) over their "insecure" analogues. The recent work of Appelbaum et al. [ADI+17] provided the first construction of a passively secure protocol for evaluating an arithmetic circuit over a finite field $\mathbb{F}$ in the setting of secure two-party computation where the computational overhead is constant over naively computing via an insecure protocol. We are interested in a related fundamental problem of achieving constant-overhead in computation for constructing an actively secure protocol over the best passive analogue. To frame the question more precisely (and eliminate inefficient constructions based on stronger primitives), we can restrict to constructions in the OT -hybrid for boolean computations(resp., OLE-hybrid for arithemtic computations). Recent works [GIP+14, HIV17] have shown constant communication (as opposed to computation) overhead passive-to-active compilation. However, achieving constant computation overhead has largely remained an open problem. The dual execution framework provides the first construction of a constant computational overhead compilation boolean circuits in the two-party setting at the price of 1-bit leakage, where the constant, in fact, is 2. Our work can be seen as making significant progress in answering this fundamental questions, where, at the same price of 1-bit leakage, we demonstrate a $1 + o(1)$ computational overhead passive-to-active compilation for securely computing boolean and arithmetic computations in two party and multi-party settings where the computations are "easily" verifiable.

We now discuss our key insights for each of the above contributions.

1.1 Results in the 1-Bit Leakage Model

Brief summary of dual execution. In the dual execution technique introduced by Mohassel and Franklin in [MF06], the parties run Yao's 2-party protocol first with Alice as the *generator* and Bob as the *evaluator*, and a second time with the roles reversed. Thus, Alice and Bob have putative outputs from the protocol instance in which they act as evaluator. However, if one party constructs an incorrect circuit, then the other holds an incorrect result. To ensure correctness, Alice and Bob perform a maliciously secure protocol to check equality of their outputs. If this check passes, they both output their respective strings. Mohassel and Franklin further provided a definition of a k-leakage model. Intuitively, when $k = 1$, the adversary can learn "1-bit" of the counter party's input from the equality test. (See §2.2 for a formal definition.) In a followup, Huang, Katz and Evans [HKE12] used the same approach, providing a security proof for this notion. They also specified a custom-designed equality test in the random oracle model that relies on an additively homomorphic encryption scheme. Note that even if the final equality test passes, the adversary may still learn a bit about the honest party's input due *selective failure* attacks. For example, an adversary corrupting Alice may produce a garbled circuit that produces the correct answer if the first bit of Bob's input is 1. Then, in the case that Bob's input begins with 1, the protocol execution is indistinguishable from an honest execution, and yet, Alice concludes that Bob's input begins with 1. This

1 bit of leakage seems unavoidable when the adversary fully controls one of the garbled circuits.

Our contribution. As noted above, prior work requires two copies of the garbled circuit for the function f and then run a maliciously secure equality test between outputs.[5] Our first observation is that after the first execution, one of the parties has a putative answer, $z = f(x,y)$, which *can be used* in the next protocol execution. In particular, for functions whose outputs can be efficiently verified in less time than it takes to compute the output, we show that it is unnecessary to compute f again. Rather, it suffices for the second party to run the verification algorithm $g(x,y,z)$ instead. Despite the simplicity of the concept, we encountered a number of subtle issues before settling on the following high-level approach for the 2-party setting. We present a novel protocol that make black-box usage of its underlying primitives.

In the first execution, Bob, as the evaluator, learns wire labels w_z for the output z but it is important that Bob does not have the decoding information for w_z. Instead, Alice provides a commitment to the decoding information for these wire labels. In the second execution, Bob acts as the generator of the checking circuit and commits to the 2 output labels for the single-bit of its output. In this circuit, Alice inputs x and the decoding information whereas Bob inputs the wire labels w_z. The checking circuit verifies the correctness of the former, performs the decoding of w_z and runs the efficient verification $g(x,y,z)$. The output of this circuit is a single bit denoting whether the verification succeeded. Alice evaluates the garbled circuit and commits to the single output wire label $\tilde{v}$. Bob sends the decoding information, and if the verification passes (i.e., the output decodes to 1), then Alice decommits to $\tilde{v}$ and Bob decommits to w_z.

Notice that the above requires Alice to commit to decoding for wire labels w_z, and for the check circuit to decode the labels. One approach to implement our commitment scheme is using verifiable secret-sharing (VSS), which allows the verification circuit to be "non-cryptographic" (and in fact information theoretic) so that our overall protocol is black-box in the underlying cryptographic primitives.[6]. In particular, Alice uses a k-out-of-n VSS scheme to commit to the decoding information for these wire labels, and Alice and Bob use OT so that Bob can recover k such shares (and thus needs 1 more share to decode). In the second execution, Alice inputs x and the n VSS decoding shares whereas Bob inputs the wire labels w_z and his k decoding shares. Finally, the checking circuit verifies that Bob's k shares appear in the set of Alice's n shares.

Let us now argue correctness (omitting subtle issues that are handled in the simulation argument in the proof of Theorem 3). In essence, our protocol guarantees two properties. First, if an answer is revealed to Bob, then there must be a valid input x' for Alice such that the result is the computation of $f(x',y)$. Second, the leakage to Alice is whether an arbitrary pre-determined predicate chosen by Alice over Bob's input is equal to the value of the function under specific inputs for Alice and Bob. This leakage

[5] [HKE12] observed that their protocol need not achieve fully malicious security, but does satisfy a notion that is stronger than honest-but-curious security.

[6] One could instead use commitments for the translation table, but this would require the check circuit to implement the cryptographic verification procedure of the decommitments. In some circumstances AES-based commitments (or other methods) might be concretely better than decoding the VSS.

is morally similar to the leakage introduced in the Mohassel-Franklin dual execution approach. As we mentioned above, selective failure attacks still apply and thus, the best security we can guarantee against a malicious Alice is 1-bit leakage. We claim, however, a slightly stronger version of security than [HKE12], where the adversary learns the output of the function *only if* the 1-bit leakage predicate evaluates to 1.[7]

When Bob is malicious, then Bob holds one set of wire labels that is correct as the garbled circuit is correct. However, Bob could produce a malicious program checker. The basic guarantee of the protocol w.r.t Alice is that the only output that can be revealed to Alice is through the wire labels obtained by Bob in the first phase; since Alice is honest, this is a valid output of the function under some specific inputs of Alice and Bob. Bob can force Alice to reveal the decoding information, but with very high probability, Alice either outputs abort (because Bob denies Alice the output), or Alice outputs the correct output. In both cases, we can construct a simulator for Bob.

Our protocol does not need a maliciously secure equality test as per [MF06, HKE12]. While our protocol achieves less overall *communication and computation*, for some very small circuits our protocol may not achieve a faster *wall-clock*. However, for large functions, the factor of 2 savings in communication and computational can overcome this penalty.

1.2 Extending Dual Execution to Other Protocols

The dual execution technique has so far only applied to Yao's garbled circuit protocols because Yao's protocol offers a one-sided correctness property. Namely, the honest garbler can ensure that the counter-party computes a correct output. The main result in this paper is to answer the natural question of whether other secure computation protocols that do not offer one-sided correctness can be efficiently transformed into ones that offer 1-bit leakage security at a cost that is much less than fully malicious security. A second question is whether we can go beyond the two-party setting. It is not clear apriori how 1-bit leakage in the dual execution paradigm is possible in the multi-party setting where there is not a natural notion of "running the protocol in the other direction".

We answer these questions affirmatively for efficiently verifiable functions by showing how to construct novel 1-bit leakage protocols from classic secure computation protocols such as [GMW87] and [BGW88], extending the dual execution paradigm in the two domains. Our technique leverages the work of Genkin et al. [GIP+14] who shows that slight modifications of these protocols already offer security *up to additive errors*. Specifically, they show that for slight variants of the passive protocols, the attack space of a malicious adversary is limited to adding an input-independent value to any wire of the circuit. Whereas Genkin et al. then refine such protocols to be fully malicious, we present a lightweight alternative that achieves 1-bit leakage. Namely, after evaluating f modulo such additive errors, the parties perform a maliciously secure evaluation of $g(x,y,z)$, and determine the output based on the result of that computation. In contrast, the work of Genkin et al. [GIP+14] shows how to transform the function to another function that is immune to additive attacks. While this work and

[7] While this notion is suggested heuristically in [HKE12], we achieve it formally. This notion is similar to the 2^{-s}-CovIDA notion presented by Mohassel and Riva [MR13].

follow up works [GIP+14, GIP15, GIW16] demonstrate compilers with constant overhead for arithmetic circuits over a large field, for the case of Boolean circuits, the best compilation due to [GIW16] incurs a polylogarithmic overhead. Moreover, the previous works have worked only for non-constant round protocols, specifically those whose complexity is proportional to the depth of the circuit.

In this work, we make two important contributions. Our simple compiler offers lightweight malicious protocols for efficiently verifiable f for a wide variety of protocols both in the two-party and multi-party settings. The computation and communication complexities of our protocols are less than twice the cost of the passive counterparts (in the OT or OLE-hybrid models). Second, we provide a key technical lemma that shows the first constant-round protocol that fits the [GIP+14] paradigm. More precisely, we show that a variant of the distributed garbling protocol (BMR) due to Beaver et al. [BMR90] offers security *up to additive errors*. This result allows us to instantiate our paradigm with the multi-party BMR protocol as well.

Unlike in our Yao-based protocol, we here *require* a malicious evaluation of g. It would be preferable to use a simpler, additive error secure protocol for g, but we currently do not know how to securely combine the outputs of f and g if *both* have additive errors. Nonetheless, even the malicious evaluation of g can be substantially more efficient than the *honest-but-curious* evaluation of f. For example, when f grows as ℓ^3 and g grows as ℓ^2, as soon as ℓ exceeds the security parameter κ (i.e., for moderate input sizes), the overall communication for a malicious evaluation of g can be less than that of f. Second, our approach extends to the multi-party setting. Examples of such functions include max-flow, perfect-matching, linear program. Thus our technique offers an advantage for an interesting class of functions. We remark that the input-independent additive security property was crucially used when (sequentially) composing a protocol for f with a protocol g. Specifically, an attempt to weaken this security property requires the simulator to precisely obtain the "attack" from the adversary.

Finally, we highlight an interesting theoretical consequence of our result regarding the additive resilience of the BMR protocol. As mentioned before, for the case of Boolean circuits, the best AMD compilation is due to [GIW16] and incurs a polylogarithmic overhead. If this result [GIW16] can be improved from polylogarithmic overhead to a constant, then combined with our protocol will yield the *first* constant-round multi-party protocol for Boolean computations whose communication and computation complexity is a constant overhead over the passive counterpart, where previous protocols have incurred $\Omega(s)$ overhead for a statistical parameter s.

Beyond additively secure protocols. In all our instantiations (of the second result), we rely on an "additive security" property of the protocol implementing f. It is tempting to ask if our framework can work for other weaker variants. It is conceivable that the only demand one would need from the protocol for f is privacy against active adversaries, however, to formally prove 1-bit leakage, one needs to precisely capture the attack caused by an active adversary on the protocol in order to extract the leakage function. In this respect, additive security is one formulation that facilitates this. We leave it as future work to generalize this approach for other types of attacks.

On randomized functionalities. In this work, we prove our theorems for deterministic f and g. However, the techniques extend to some cases when f and g are randomized.

For example, Harvey's perfect matching algorithm is a randomized algorithm and it works in our framework because it has a unique output. We believe our framework will generalize to randomized f and g if the algorithm admits "unique" outputs. While we do not formalize "uniqueness" for algorithms, the reason our compilers require the output to be unique is because the "weaker" protocol we rely on for f could allow an active adversary to adaptively choose one of the solutions if more than one exist. Such an adversary will not be caught in our framework as g might return accept for all solutions and it is unclear how to simulate or how to formalize this attack via an ideal functionality. That being said, we believe that in some cases we can extend our framework beyond deterministic functionalities and leave it for future work.

2 Preliminaries

2.1 Verifiable Secret Sharing (VSS)

A verifiable secret sharing (VSS) [CGMA85] scheme is a two-stage secret sharing protocol for implementing the following functionality. In the first stage, denoted by $\texttt{Share}(s)$, a special player referred to as dealer, shares a secret s among n players, in the presence of at most t corrupted players. In the second stage, denoted by $\texttt{Recon}$, players exchange their views of the share stage, and reconstruct the value s. We use notation $\texttt{Recon}(\mathsf{S}_1, \ldots, \mathsf{S}_n)$ to refer to this procedure. The functionality ensures that when the dealer is honest, before the second stage begins, the t corrupted players have no information about the secret. Moreover, when the dealer is dishonest, at the end of the share phase the honest players would have realized it through an accusation mechanism that disqualifies the dealer. A VSS scheme can tolerate errors on malicious dealer and players on distributing inconsistent or incorrect shares, indeed the critical property is that even in case the dealer is dishonest but has not been disqualified, still the second stage always reconstructs the same string among the honest players. In this paper, we use a (n,t)-perfectly secure VSS scheme with a deterministic reconstruction procedure [GIKR01].

Definition 1 (VSS Scheme) *An $(n+1,t)$-perfectly secure VSS scheme consists of a pair of protocols $\texttt{VSS} = \langle \texttt{Share}, \texttt{Recon} \rangle$ that implement respectively the sharing and reconstruction phases as follows.*

$\texttt{Share}(s)$**.** *Player P_{n+1} referred to as dealer runs on input a secret s and randomness r_{n+1}, while any other player P_i, $1 \leq i \leq n$, runs on input a randomness r_i. During this phase players can send (both private and broadcast) messages in multiple rounds.*

$\texttt{Recon}(\mathsf{S}_1, \ldots, \mathsf{S}_n)$**.** *Each shareholder sends its view v_i of the sharing phase to each other player, and on input the views of all players (that can include bad or empty views) each player outputs a reconstruction of the secret s.*

All computations performed by honest players are efficient. The computationally unbounded adversary can corrupt up to t players that can deviate from the above procedures. The following security properties hold.

- **Commitment:** *if the dealer is dishonest then one of the following two cases happen: 1) during the sharing phase honest players disqualify the dealer, therefore they*

output a special value $\perp$ and will refuse to play the reconstruction phase; 2) during the sharing phase honest players do not disqualify the dealer, therefore such a phase determines a unique value s^* that belongs to the set of possible legal values that does not include $\perp$, which will be reconstructed by the honest players during the reconstruction phase.

- **Secrecy:** *if the dealer is honest then the adversary obtains no information about the shared secret before running the protocol* `Recon`.
- **Correctness:** *if the dealer is honest throughout the protocols then each honest player will output the shared secret s at the end of protocol* `Recon`.

We are interested in a deterministic reconstruction procedure, therefore we adopt the scheme of [GIKR01] that implements an $(n+1, \lfloor n/4 \rfloor)$-perfectly secure VSS scheme.

2.2 Secure Computation with 1-bit Leakage

In this section, we present a security definition that incorporates the notion of 1-bit leakage. For simplicity, we provide it for the two-party setting. It can easily be extended to multiparty setting. We consider static corruptions by malicious adversaries who may deviate from the protocol in an arbitrary manner. Our notion will follow the standard Goldreich's formalization of an Ideal and Real executions [Gol04] with the appropriate weakening from [HKE12] in which the adversary is allowed to submit a leakage predicate. However, our notion will be stronger than the definition in [HKE12] because the adversary learns the output in the optimistic case. These experiments will capture the idea of correctness and input independence: the honest party's output still corresponds to $f(x,y)$ and the adversary's input is independent of the honest party's input.

Real Execution. A two-party protocol Π is executed by Alice and Bob. The adversary A receives the inputs of the corrupted party and arbitrary auxiliary input z and sends all messages on behalf of the corrupted party. The honest party follows the instructions in Π. We define the random variable $\mathbf{View}_{\Pi,A(z)}(x,y,\kappa)$ to denote the entire view of adversary A in the execution of Π where Alice holds input x, Bob holds input y and the security parameter is 1^κ. We define the random variable $\mathsf{out}_{\Pi,A(z)}(x,y,\kappa)$ to denote the output of the honest party after the execution of the protocol. Finally, define the tuple

$$\text{REAL}_{\Pi,A(z)}(x,y,z) \equiv (\mathbf{View}_{\Pi,A(z)}(x,y,\kappa), \mathsf{out}_{\Pi,A(z)}(x,y,\kappa))$$

Ideal Execution. In the ideal execution, parties Alice and Bob interact with an ideal functionality; as before, the adversary has corrupted one of the parties, Alice holds input x, Bob holds input y and both hold the security parameter 1^κ. The adversary receives the input of the corrupted party and has an arbitrary auxiliary input string z. The honest party sends its input to the trusted party. The corrupted party controlled by A may send an arbitrary input $\tilde{y}$ to the trusted party. Denote the pair of inputs sent to the trusted party as $(\tilde{x},\tilde{y})$. The adversary also sends an arbitrary Boolean function g to the trusted party. The trusted party computes the predicate $g(\tilde{x},\tilde{y})$. If the predicate evaluates to 0, the trusted party sends "abort" to both parties. If the predicate evaluates to 1, the trusted party evaluates $f(\tilde{x},\tilde{y})$ and and gives both values to the adversary. If

the adversary sends the message "stop" to the trusted third party, then the honest party is given $\perp$. Otherwise, the honest party is given $f(\tilde{x}, \tilde{y})$. (This models the inherent lack of complete fairness.) The honest party outputs the message given by the trusted third party. The adversary can output an arbitrary string of its view. We define the random variable $\mathsf{out}^A_{f,A(z)}(x,y,\kappa)$ to denote the output of the adversary A and $\mathsf{out}^h_{f,A(z)}(x,y,\kappa)$ to denote the output of the honest party. Finally, define the tuple $\text{IDEAL}_{f,A(z)}(x,y,\kappa) \equiv (\mathsf{out}^A_{f,A(z)}(x,y,\kappa), \mathsf{out}^h_{f,A(z)}(x,y,\kappa))$.

Definition 1. *A protocol Π for the function f is said to securely compute f with 1-bit leakage if for every p.p.t. adversary A, there exists a p.p.t. simulator S in the ideal model such that*

$$\left\{\text{REAL}_{\Pi,A(z)}(x,y,\kappa)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}} \approx_c \left\{\text{IDEAL}_{f,S(z)}(x,y,\kappa)\right\}_{x,y,z\in\{0,1\}^*,\kappa\in\mathbb{N}}$$

Remark. We mention the security notion of ϵ-CovIDA introduced by Mohassel and Riva [MR13] which implies our notion for the correct parameters. Essentially, this notion requires that if a player is trying to cheat, the other players can catch it with probability $1-\epsilon$, but even if it is not caught (i.e., with probability ϵ) the cheater can only learn a single bit of extra information about the other players' inputs, and the correctness of the output is still guaranteed.

Extending to multiparty protocols. Similarly to the two-party case, we define the random variable $\mathbf{View}_{\Pi,A(z),\mathcal{I}}(x_1,\ldots,x_m,n)$ to denote the entire view of adversary A in the execution of Π where party P_i holds input x_i, the adversary corrupts the parties in $\mathcal{I}$, and the security parameter is 1^n. We define the random variable $\mathsf{out}_{\Pi,A(z),\mathcal{I}}(x_1,\ldots,x_m,n)$ to denote the output of the honest party $j \in [m]/\mathcal{I}$ after the execution of the protocol. Finally, define the tuple

$$\begin{aligned}&\text{REAL}_{\Pi,A(z),\mathcal{I}}(x_1,\ldots,x_m,n)\\ &\equiv (\mathbf{View}_{\Pi,A(z),\mathcal{I}}(x_1,\ldots,x_m,n), \mathsf{out}_{\Pi,A(z),\mathcal{I}}(x_1,\ldots,x_m,n))\end{aligned}$$

Analogously, in the ideal world, we allow the adversary to submit a leakage function g to the ideal functionality. We define $\mathsf{out}^A_{f,A(z),\mathcal{I}}(x_1,\ldots,x_m,n)$ to denote the output of the adversary A and $\mathsf{out}^h_{f,A(z),\mathcal{I}}(x_1,\ldots,x_m,n)$ to denote the output of the honest party. Finally, define the tuple

$$\begin{aligned}&\text{IDEAL}_{f,A(z)}(x_1,\ldots,x_m,n)\\ &\equiv (\mathsf{out}^A_{f,A(z),\mathcal{I}}(x_1,\ldots,x_m,n), \mathsf{out}^h_{f,A(z),\mathcal{I}}(x_1,\ldots,x_m,n))\end{aligned}$$

Finally, security is defined by requiring indistinguishability of **REAL** and **IDEAL**.

Remark 1. To achieve stand-alone (full) security our proofs only rely on sequential composition, which in turn requires the sub-protocols to only satisfy stand-alone security. Nevertheless, we note that our proofs can further achieve UC security if the underlying sub-protocols achieve UC security.

2.3 Garbled Circuits

Definition 2 (Garbling scheme) *A garbling scheme* $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ *consists of four polynomial-time algorithms that work as follows:*

- $(\widetilde{\mathsf{C}}, \mathbf{dk}, \mathsf{sk}) \leftarrow \mathsf{Grb}(1^\kappa, \mathsf{C})$*: is a probabilistic algorithm that takes as input a circuit* C *with* $2n$ *input wires and* n *output wires and returns a garbled circuit* $\widetilde{\mathsf{C}}$*, a set of decoding keys* $\mathbf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ *and a secret key* sk*.*
- $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathsf{sk}, \mathbf{x})$ *is a deterministic algorithm that takes an input a secret key* sk*, an input* $\mathbf{x}$ *and returns an encoded input* $\widetilde{\mathbf{x}}$*. We denote this algorithm by* $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathsf{sk}, \widetilde{\mathbf{x}})$*. In this work we consider* decomposable garbled schemes*. Namely, the algorithm takes multiple input bits* $\mathbf{x} = (x_1, \ldots, x_n)$*, runs* $\mathsf{Enc}(\mathsf{sk}, \cdot)$ *on each* x_i *and returns the garbled inputs* $\widetilde{x}_1$ *through* $\widetilde{x}_n$*, denoted by input labels.*
- $\widetilde{\mathbf{y}} := \mathsf{Eval}(\widetilde{\mathsf{C}}, \widetilde{\mathbf{x}})$*: is a deterministic algorithm that takes as input a garbled circuit* $\widetilde{\mathsf{C}}$ *and encoded inputs* $\widetilde{\mathbf{x}}$ *and returns encoded outputs* $\widetilde{\mathbf{y}}$*.*
- $\{\perp, y_i\} := \mathsf{Dec}(\mathsf{dk}_i, \widetilde{y}_i)$*: is a deterministic algorithm that takes as input a decoding key* dk_i *and an encoded output* $\widetilde{y}_i$ *and returns either the failure symbol* $\perp$ *or an output* y_i*. We write* $\{\perp, \mathbf{y}\} := \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{y}})$ *to denote the algorithm that takes multiple garbled outputs* $\widetilde{\mathbf{y}} = (\widetilde{y}_1 \ldots \widetilde{y}_n)$*, runs* $\mathsf{Dec}(\mathsf{dk}_i, \cdot)$ *on each* $\widetilde{y}_i$ *and returns the outputs* y_1 *through* y_n*.*

We remark that we only require that our garbling scheme maintains the privacy property, rather than stronger properties such as authenticity or obliviousness.

Correctness. We say that Garb is correct if for all $n \in \mathbb{N}$, for any polynomial-size circuit C, for all inputs $\mathbf{x}$ in the domain of C, for all $(\widetilde{\mathsf{C}}, \mathbf{dk}, \mathsf{sk})$ output by $\mathsf{Grb}(1^\kappa, \mathsf{C})$, for $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathsf{sk}, \mathbf{x})$ and $\widetilde{\mathbf{y}} := \mathsf{Eval}(\widetilde{\mathsf{C}}, \widetilde{\mathbf{x}})$ and for all $i \in [n]$, $y_i := \mathsf{Dec}(\mathsf{dk}_i, \widetilde{y}_i)$, where $(y_1, \ldots, y_n) = \mathsf{C}(\mathbf{x})$.

Privacy. We say that a garbling scheme Garb is secure if there exists a PPT algorithm SimGC such that for any family of polynomial-size circuits C_κ and sequence of inputs $\{\mathbf{x}_\kappa\}_\kappa$,

$$\{(\widetilde{\mathsf{C}}, \mathbf{dk}, \mathsf{sk}) \leftarrow \mathsf{Grb}(1^\kappa, \mathsf{C}_\kappa); \widetilde{\mathbf{x}} := \mathsf{Enc}(\mathsf{sk}, \mathbf{x}_\kappa) : (\widetilde{\mathsf{C}}, \widetilde{\mathbf{x}}, \mathbf{dk})\}_\kappa \overset{\mathrm{c}}{\approx} \{\mathbf{y} = \mathsf{C}(\mathbf{x}_\kappa) : \mathsf{SimGC}\,(1^\kappa, \mathsf{C}_\kappa, \mathbf{y})\}_\kappa.$$

2.4 The [BMR90] Garbling

An extension of Yao garbled circuits approach [Yao86] for any number of parties n introduced by Beaver, Micali and Rogaway in [BMR90] leading to the first constant-round protocol. This protocol has an offline phase in which the garbled circuit is created, and an online phase in which the garbled circuit is evaluated. The [BMR90] garbling technique involves garbling each gate separately using pseudorandom generators (or pseudorandom functions) while ensuring consistency between the wires. This method was recently improved by Lindell et al. in [LPSY15] which introduced an NC^0 functionality for this task, while demonstrating that the PRF values submitted by each party

need not be checked for consistency (or computed by the functionality), as inconsistency would imply an abort by at least one honest party. Moreover, an abort event is independent of the honest parties' inputs due to the way each gate is garbled. In more details, the garbling functionality used in [LPSY15] is a modification of the garbling functionality introduced in [BMR90] and is applicable for any number of parties n. Namely, let C denote the circuit computed by the parties. Then for every wire w, party P_i inputs to the functionality two keys $k^i_{w,0}, k^i_{w,1}$ and the PRF computations based on these keys, as well as a masking bit share λ^i_w. The functionality creates the garbling for each gate which includes four rows such that each row is combined of n ciphertexts.

We will now describe the technical details of the BMR garbling. Without loss of generality, we assume that C is a Boolean circuit comprising of fan-in two AND and XOR gates, a total number of W wires and G gates. Then for every AND gate $g \in G$ with input wires $1 \leq a, b \leq W$ and output wire c, the garbled row $r_1, r_2 \in \{0,1\}$ in gate g is expressed as the concatenation of $\mathbf{R}_{g,r_1,r_2} = \{R^j_{g,r_1r_2}\}_{j=1}^n$, where

$$R^{g,j}_{r_1r_2} = \underbrace{\bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k^i_{a,r_1}}(g,j,r_1,r_2) \oplus \mathsf{PRF}_{k^i_{b,r_2}}(g,j,r_1,r_2) \right)}_{\text{Ciphertext Padding}}$$
$$\underbrace{\oplus\, k^j_{c,0} \oplus \Big(\underbrace{\chi_{r_1,r_2}}_{\text{Perm. Bit}} \cdot \underbrace{(k^j_{c,1} \oplus k^j_{c,0})}_{\text{Wire's } \Delta} \Big)}_{\text{Plaintext}}$$

and PRF is a PRF, $k^i_{a,0}, k^i_{a,1}$ and $k^i_{b,0}, k^i_{b,1}$ are the respective input keys of party P_i, whereas $k^i_{c,0}, k^i_{c,1}$ are its output keys. Furthermore, for every a, b and r_1, r_2 as above the permutation bit χ_{r_1,r_2}, that "chooses" the output key to be encrypted, is defined by

$$\chi_{r_1,r_2} = ((\lambda_a \oplus r_1) \cdot (\lambda_b \oplus r_2)) \oplus \lambda_c$$

As specified above, the inputs to the [LPSY15]-style functionality may be inconsistent, implying an incorrect computation. We next describe their functionality for a general circuit C with n inputs $x_1, \ldots, x_n$ where x_i represents the value input by party P_i. Let $F = \{\mathsf{PRF}_k : \{0,1\}^\kappa \to \{0,1\}^\kappa\}_{k \in \{0,1\}^*, \kappa \in \mathbb{N}}$ be a family of PRFs. Then the encoding procedure takes the inputs $x_1, \ldots, x_n$ and additional random inputs $R_1, \ldots, R_n$ where each R_j is comprised of PRF keys $\{k^j_{w,0}, k^j_{w,1}\}_w$, masking bits shares $\{\lambda^j_w\}_w$ and PRF evaluations

$$\left\{\mathsf{F}^{g,j}_{w,0,0}, \mathsf{F}^{g,j}_{w,0,1}, \mathsf{F}^{g,j}_{w,1,0}, \mathsf{F}^{g,j}_{w,1,1}\right\}_{w \in W, g \in G}$$

that allegedly correspond to

$$\left\{\mathsf{PRF}_{k^j_{w,0}}(g,j,0,0), \mathsf{PRF}_{k^j_{w,0}}(g,j,0,1), \mathsf{PRF}_{k^j_{w,1}}(g,j,1,0), \mathsf{PRF}_{k^j_{w,1}}(g,j,1,1)\right\}_{w \in W, g \in G}$$

The encoding procedure $\mathsf{BMR.Encode}$ on input $((x_1, R_1), ..., (x_n, R_n))$ outputs

$$\underbrace{(R^{g,j}_{00}, R^{g,j}_{01}, R^{g,j}_{10}, R^{g,j}_{11})_{g \in G, j \in [n]}}_{\text{Garbled Tables}} \quad \underbrace{(\Lambda_w, k^1_{w,\Lambda_w}, \ldots, k^n_{w,\Lambda_w})_{w \in \mathsf{Inp}}}_{\text{keys and masks for input wires}} \quad \underbrace{(\lambda_w)_{w \in \mathsf{Out}}}_{\text{Output translation table}}$$

where

$$R^{g,j}_{r_1,r_2} = \Big(\bigoplus_{i=1}^{n} \mathsf{F}^{g,i}_{a,r_1,r_2}\Big) \oplus \Big(\bigoplus_{i=1}^{n} \mathsf{F}^{g,i}_{b,r_1,r_2}\Big) \oplus S^{g,j}_{r_1,r_2}$$

$$S^{g,j}_{r_1,r_2} = k^j_{c,0} \oplus \chi_{r_1,r_2} \cdot (k^j_{c,1} \oplus k^j_{c,0})$$

$$\chi_{r_1,r_2} = \mathsf{AND}(\lambda_a \oplus r_1, \lambda_b \oplus r_2) \oplus \lambda_c \;=\; [(\lambda_a \oplus r_1) \cdot (\lambda_b \oplus r_2)] \oplus \lambda_c$$

$$\lambda_w = \begin{cases} \lambda^{j_w}_w & \text{if } w \in \mathsf{Inp} \quad \texttt{// input wire} \\ \lambda^1_w \oplus \cdots \oplus \lambda^n_w & \text{if } w \in W/\mathsf{Inp} \; \texttt{// internal wire} \end{cases}$$

$$\Lambda_w = \lambda_w \oplus x_w \text{ for all } w \in \mathsf{Inp} \qquad \texttt{// masked input bit}$$

and wires a, b and $c \in W$ denote the input and output wires respectively for gate $g \in G$. $\mathsf{Inp} \subseteq W$ denotes the set of input wires to the circuit, $j_w \in [n]$ denotes the party whose input flows the wire w and x_w the corresponding input. $\mathsf{Out} \subseteq W$ denotes the set of output wires.

The decoding procedure basically corresponds to the evaluation of the garbled circuit. More formally, the decoding procedure $\mathsf{BMR.Decode}$ is defined iteratively gate by gate according to some standard (arbitrary) topological ordering of the gates. In particular, given an encoding information k^j_{w,Λ_w} for every input wire w and $j \in [n]$, of some input x, then for each gate g with input wires a and b and output wire c compute

$$k^j_c = R^{g,j}_{r_1,r_2} \oplus \bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k^i_{a,\Lambda_a}}(g, j, \Lambda_a, \Lambda_b) \oplus \mathsf{PRF}_{k^i_{b,\Lambda_b}}(g, j, \Lambda_a, \Lambda_b) \right)$$

Finally given Λ_w for every output wire w, compute the output carried in wire w as $\Lambda_w \oplus \left(\bigoplus_{j=1}^{n} \lambda^j_w\right)$.

Our proof makes use of the active key terminology originated from [LP09] which refers to a PRF key that is revealed to the adversary during the garbled circuit evaluation. Similarly, an inactive key refers to a wire key that is not revealed to the adversary during the evaluation. Each wire in the circuit is always associated with one active key and one inactive key (otherwise privacy would be violated). Developing this notion, an active path refers to the entire path visited throughout the evaluation. In the BMR-style garbling, the active path is chosen at random based on the masking bits that hide the actual wire value.

3 Dual Execution with Efficient Verification

In this section, we present a secure computation protocol for functions that have efficient verification that achieves security against active adversaries with 1-bit leakage. The protocol follows the spirit of the dual-execution [MF06]. However, we achieve greater efficiency as we do not require to garble the same circuit twice and our protocols do not require an extra secure equality test. Note, our technique can also be applied to functions that do not have efficient verification: namely, the predicate g can simply recompute f in addition to performing its other checks. In this sense, our framework subsumes prior dual-execution techniques for achieving 1-bit leakage.

Definition 2. *We say that function* $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ *can be* verified with *predicate* g*, if* $f(x_A, x_B) = z \Leftrightarrow g(x_A, x_B, z) = 1$

Our protocol is described in the $(\mathcal{F}_{\text{COM}}, \mathcal{F}_{\text{OT}})$-hybrid where $\mathcal{F}_{\text{COM}}$ is the *ideal* commitment functionality and $\mathcal{F}_{\text{OT}}$ is the *ideal* 1-out-of-2 oblivious transfer functionality. We require a garbling scheme $\mathsf{Garb} = (\mathsf{Grb}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ as per Definition 2 in our protocol. Let f be an arbitrary two-party functionality that can be verified with predicate g. Let P_x describe the circuit that on input y computes $f(x, y)$. Let $\mathrm{P}_{y,\widetilde{\mathbf{w}},\bar{\mathbf{t}},S}$ describe the circuit that on input $(x, \bar{\mathbf{s}})$ outputs 1 if and only if

1. $\bar{\mathbf{t}}$ and $\bar{\mathbf{s}}$ are consistent on the set $S \subset [n]$, and
2. $g(x, y, w) = 1$ where $w = \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$ and $\mathbf{dk}$ is the reconstruction of $\bar{\mathbf{s}}$, and
3. $w \neq \perp$.

and otherwise outputs $\perp$.

Theorem 3 *Assuming the existence of a garbling scheme, protocol* $\Pi^f_{1-\text{LEAK}}$ *described in Fig. 1 securely realizes* $\mathcal{F}^f_{1-\text{LEAK}}$ *in the* $(\mathcal{F}_{\text{COM}}, \mathcal{F}_{\text{OT}})$*-hybrid with communication complexity* $p(|f|) + p(|g| + O(\kappa)) + poly(\kappa)$.

Remark 2. Note that when $|g| = o(|f|)$, then our protocol achieves complexity that is $(1 + o(1))|f|$, whereas prior work requires $2|f|$ communication. In practice, this factor of 2 in complexity can be substantial.

Proof: We first describe simulators for Alice and Bob and then prove correctness of simulation.

Simulating Alice. In the ideal world, the simulator $\mathcal{S}^A$ internally simulates $\mathcal{F}_{\text{COM}}$ and $\mathcal{F}_{\text{OT}}$ for Alice.

In Phase 1, Bob does not send any message directly to Alice. Bob only provides inputs to the $\mathcal{F}_{\text{OT}}$. The calls made to the $\mathcal{F}_{\text{OT}}$ functionality are internally simulated by $\mathcal{S}^A$ where it collects all the sender's message to the functionality. At the end of Phase 1, $\mathcal{S}^A$ obtains $\mathsf{sk}, \bar{\mathbf{s}}$ from adversary Alice via the OT calls. Using $\bar{\mathbf{s}}$ it reconstructs $\mathbf{dk}$. It chooses a random t subset S of $[n]$ and sets $\bar{\mathbf{t}} = \{s_i\}_{i \in S}$.

In Phase 2, recall that Bob garbles the circuit $P_{y,\widetilde{\mathbf{w}},\bar{\mathbf{t}},S}$ where y is Bob's input, $\bar{\mathbf{t}}$ is the set of shares of $\mathbf{dk}$ obtained by Bob on set S, and $\widetilde{\mathbf{w}}$ is the output of the garbled circuit evaluation in Phase 1. It first obtains Alice's input $x, \overline{\mathbf{s}'}$ from its message to the $\mathcal{F}_{\text{OT}}$ functionality in the second step of Phase 2. In the third message, the simulator has to send a garbling to Alice. If $\overline{\mathbf{s}'}$ fails to agree with $\bar{\mathbf{s}}$ on the set of indices S or does not reconstruct to $\mathbf{dk}$, then $\mathcal{S}^A$ generates a Garbled Circuit that outputs 0 on all inputs. Otherwise, $\mathcal{S}^A$ computes the leakage function g^A as follows:

Function g^A: Input y, Parameters : $\widetilde{\mathrm{C}}, \mathbf{dk}, \mathsf{sk}, x$.

- Compute $\widetilde{\mathbf{y}} := \mathsf{Enc}(\mathsf{sk}, y)$, evaluate $\widetilde{\mathbf{w}} := \mathsf{Eval}(\widetilde{\mathrm{C}}, \widetilde{\mathbf{y}})$ and obtain $w = \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$.
- Then compute $g(x, y, w)$ and output the result.

Protocol $\Pi^f_{1-\text{LEAK}}$.

Phase 1:

1. Alice with input x, computes $(\widetilde{\mathsf{C}}, \mathbf{dk}, \mathsf{sk}) \leftarrow \mathsf{Grb}(1^\kappa, \mathrm{C}_x)$.
2. Alice and Bob engage in n parallel executions of the $\mathcal{F}_{\text{OT}}$-functionality where Alice sends (k_i^0, k_i^1) to $\mathcal{F}_{\text{OT}}$ in the i^{th} parallel instance and Bob sends y_i where $\mathsf{sk} = (k_1^0, k_1^1, \ldots, k_n^0, k_n^1)$ and Bob's inputs is $y = y_1 \cdots y_n$. Let $\widetilde{\mathbf{y}}$ be the concatenation of the outputs received from the n oblivious-transfer executions.
3. Next, Alice sends $\widetilde{\mathsf{C}}$ to Bob. Bob evaluates the garbled circuit by computing $\widetilde{\mathbf{w}} := \mathsf{Eval}(\widetilde{\mathsf{C}}, \widetilde{\mathbf{y}})$.
4. Alice encodes $\mathbf{dk}$ into $\overline{\mathbf{s}} = (s_1, \ldots, s_n)$ using a t-out-of-n VSS scheme. Then Alice and Bob engage in a k-out-of-n oblivious transfer, where Bob as the receiver picks k indices of n, say set, $S \subset [n]$ and the sender uses the VSS shares as its input. Let $\overline{\mathbf{t}} = \{s_i\}_{i \in S}$ the set of shares received by Bob.
5. Bob then commits to $\widetilde{\mathbf{w}}$ using $\mathcal{F}_{\text{COM}}$.

Phase 2:

1. Bob then computes garbling $(\widetilde{\mathrm{P}}, \mathbf{dk}_P, \mathsf{sk}_P) \leftarrow \mathsf{Grb}(1^\kappa, \mathrm{P}_{y, \widetilde{\mathbf{w}}, \overline{\mathbf{t}}})$.
2. Alice and Bob engage in n parallel executions of $\mathcal{F}_{\text{OT}}$ where Bob sends $(\widetilde{k}_i^0, \widetilde{k}_i^1)$ to $\mathcal{F}_{\text{OT}}$ in the i^{th} parallel instance and Alice sends the i^{th} bit of $(x, \overline{\mathbf{s}})$ where $\mathsf{sk}_P = (\widetilde{k}_1^0, \widetilde{k}_1^1, \ldots, \widetilde{k}_m^0, \widetilde{k}_m^1)$. Let $\widetilde{\mathbf{x}}$ be the concatenation of the outputs received from the m oblivious-transfer executions.
3. Next, Bob sends $\widetilde{\mathrm{P}}$ to Alice. Alice evaluates the garbled circuit by computing $\widetilde{\mathbf{v}} := \mathsf{Eval}(\widetilde{\mathrm{P}}, \widetilde{\mathbf{y}})$.
4. Alice commits to $\widetilde{\mathbf{v}}$ using $\mathcal{F}_{\text{COM}}$.
5. Bob sends $\mathbf{dk}_P$ to Alice. Using $\mathbf{dk}_P$, Alice computes the answer $v \leftarrow \mathsf{Dec}(\mathbf{dk}_P, \widetilde{\mathbf{v}})$.
6. If $v \neq 1$, Alice aborts. Otherwise it decommits $\widetilde{\mathbf{v}}$ and sends $(s_1, \ldots, s_n)$ to Bob.
7. If $\overline{\mathbf{t}}$ is not consistent with $\overline{\mathbf{s}}$, then Bob aborts. Otherwise it reconstructs $\mathbf{dk}$ from $\overline{\mathbf{s}}$ and computes $w \leftarrow \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$. Then it decommits its commitment to $\widetilde{\mathbf{w}}$ from Step 5, Phase 1 to Alice.
8. Alice computes w from $\widetilde{\mathbf{w}}$ and both parties output w.

Fig. 1. $\Pi^f_{1-\text{LEAK}}$: A 1-LEAK protocol for f

The simulator submits this leakage function g^A along with Alice's input x. Recall that the ideal functionality w.r.t Alice will return $f(x,y)$ if and only if $g(y) = 1$. If $g(y) = 1$ and the simulator obtains $w = f(x,y)$ then it will simulate a garbled circuit that outputs 1.

If Alice fails to send the result of the evaluation or sends inconsistent shares of $\mathbf{dk}$, the simulator does not allow the output to be delivered to Bob. Otherwise, it completes the execution with Alice and outputs its view by first computing $\widetilde{\mathbf{w}}$ such that $w = \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$ where w was received from the ideal functionality and faking the decommitment to $\widetilde{\mathbf{w}}$ in the end of Phase 2. If the protocol completes, then the Simulator lets the output to be delivered to Bob.

Correctness of simulation: It follows from the description of the simulator $\mathcal{S}^A$ that the only messages that the simulator fakes in the ideal world include the garbled circuit Bob generates in Phase 2 and the decommitment to $\widetilde{\mathbf{w}}$ in step 7. Indistinguishability will essentially follow from the fact that the simulator uses the correct output for the simulated garbled circuit in Phase 2 and the security of the garbling scheme.

More precisely, we consider an intermediate hybrid H and a simulator S_1 that receives Bob's input y. The simulation proceeds identically to the real world in Phase 1 with the exception that all the $\mathcal{F}_{\text{OT}}$ and $\mathcal{F}_{\text{COM}}$ calls are simulated by S_1 where it obtains all the messages between Alice and these functionalities. In the second phase, the garbled circuit is constructed according to the real simulator S where g is computed locally by S_1 as it has Bob's input. If in the computation by g, the value obtained for $\widetilde{\mathbf{w}}$ is inconsistent with what was decoded by Bob in Phase 1, then the simulator aborts.

Hybrid H_1. Indistinguishability of H_1 from the real world can be reduced to the security of the garbling scheme if we show that (1) the probability that the simulator aborts is negligible, and (2) the output used by the simulator S_1 to simulate the garbled circuit in the Phase 2 is indistinguishable distributed to the output obtained by Alice in the real world while evaluating the garbled circuit. There are two cases:

Case 1: Output is 0 because $\overline{\mathbf{s}'}$ is inconsistent with $\bar{\mathbf{t}}$: The probability that this event occurs is identical in the real and simulation (an in particular in this hybrid experiment).

Case 2: $\overline{\mathbf{s}'}$ is consistent with $\bar{\mathbf{t}}$: By a standard analysis, we can claim that except with negligible probability $\overline{\mathbf{s}'}$ is reconstructed correctly to $\mathbf{dk}$ in evaluation of program garbled program P in the real world, where $\mathbf{dk}$ is the reconstruction of $\bar{\mathbf{s}}$. Next, we observe that the first step in the computation of g^A proceeds identically to the actions of Bob in Phase 1. Therefore, the $\widetilde{\mathbf{w}}$ obtained in the computation by g^A will be indistinguishable distributed to $\widetilde{\mathbf{w}}$ seen in the protocol in Phase 1 in this hybrid. Conditioned on this, the second step of the computation of g^A follows identical to the evaluation of P because the w obtained by computing $\mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$ will result in the same value and the other values x and y are the same by construction.

This means that, except with negligible probability, the output of g conditioned on $\overline{\mathbf{s}'}$ being consistent with $\bar{\mathbf{t}}$ is identical to the evaluation of the program P in the real world. Therefore, we can conclude the view of the adversary in the real world and H_1 are computationally indistinguishable.

Next, we compare H_1 and the ideal world. The only difference between these two experiments is in Step 7 of Phase 2 where the message $\widetilde{\mathbf{w}}$ is decommitted to by Bob to Alice. In H_1 this is done according to the real world and in the ideal world, $\mathcal{S}$ computes $\widetilde{\mathbf{w}}$ from $\mathbf{dk}$ and w. To argue indistinguishability, we first remark that conditioned on $\overline{\mathbf{s}'}$ being consistent with $\bar{\mathbf{t}}$, then except with negligible probability, if g^A returns a 1, it holds that the value obtained by Bob in the first Phase must have been $\widetilde{\mathbf{w}}$ and that $f(x, y) = w$ by the correctness of the function g. This means when $\overline{\mathbf{s}'}$ is consistent with $\bar{\mathbf{t}}$ the simulation is identically distributed and the output received by Bob in the ideal world is correct. On the other hand, when $\overline{\mathbf{s}'}$ is inconsistent with $\bar{\mathbf{t}}$, the view of Alice in H_1 and the ideal world are identically distributed. This concludes the proof of indistinguishability and the correctness of the simulation.

Simulating Bob. To simulate a malicious Bob in Phase 1, $\mathcal{S}^B$ obtains Bob's input y from the $\mathcal{F}_{\text{OT}}$ calls. Next, it samples wire labels $\widetilde{\mathbf{w}}$ and creates a simulated garbled circuit that evaluates to set of wire labels $\widetilde{\mathbf{w}}$. Next Bob tries to retrieve a subset of the VSS shares of $\mathbf{dk}$. Let S be the set of indices. $\mathcal{S}^B$ creates t shares $\bar{\mathbf{t}}$ and provides that as the output to Bob in Step 4. Finally, it collects the message committed to by Bob in the last step of Phase 1.

In Phase 2, the simulator obtains $\mathsf{sk}_P, \mathbf{dk}_P$ from Bob and the garbling $\widetilde{\mathrm{P}}$. It constructs the following leakage function g^B.

Function g^B: Input x, Parameters : $\widetilde{\mathrm{P}}, \mathbf{dk}_P, \mathsf{sk}_P, y, \mathbf{dk}$.

- Compute $w = f(x,y)$ and extends the shares $\bar{\mathbf{t}}$ to set of shares $\bar{\mathbf{s}} = (s_1, \ldots, s_n)$ such that $\bar{\mathbf{s}}$ agrees with $\bar{\mathbf{t}}$ on set S and $\bar{\mathbf{s}}$ reconstructs to $\mathbf{dk}$ such that $\mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}}) = w$.
- Compute $\widetilde{\mathbf{x}} := \mathsf{Enc}(\mathsf{sk}_P, (x, \bar{\mathbf{s}}))$, evaluate $\widetilde{\mathbf{v}} := \mathsf{Eval}(\widetilde{\mathrm{P}}, \widetilde{\mathbf{x}})$ and obtain $v = \mathsf{Dec}(\mathbf{dk}_P, \widetilde{\mathbf{v}})$ and return v as the result.

$\mathcal{S}^B$ submits the leakage function to the ideal functionality. If the result is 0, then $\mathcal{S}^B$ makes Alice abort. If the result of the leakage function is 1, it obtains $w = f(x,y)$ from the ideal functionality. Alice sends $\widetilde{\mathbf{v}}$ to Bob such that $1 = \mathsf{Dec}(\mathbf{dk}_P, \widetilde{\mathbf{v}})$ (computed using $\mathbf{dk}_P$) and set of shares $(s_1, \ldots, s_n)$ such that $w = \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$ and $(s_1, \ldots, s_n)$ reconstructs to $\mathbf{dk}$ and agrees with $\bar{\mathbf{t}}$ on the set S.

Correctness of simulation: Briefly, we follow a similar approach as with the simulation of Alice. In other words, we argue that the leakage function g^B mimics the evaluation by Alice in Phase 2 on the right inputs. We consider intermediate hybrid experiments to argue indistinguishability.

Hybrid H_1: In this hybrid, we consider a simulator that receives Alice's input. The hybrid experiment proceeds identically to the real world with the exception that the simulator picks a translation table $\mathbf{dk}$ computes $\widetilde{\mathbf{w}}$ such that $w = \mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}})$ where $w = f(x,y)$. Recall that the simulator can extract y from Bob in Step 2 from the OT call. Next, it simulates a garbled circuit on behalf of Alice such that the evaluation by Bob results in $\widetilde{\mathbf{w}}$. The rest of the protocol follows identically to the real world. Indistinguishability of H_1 and the real world follows from the simulation of the garbling scheme.

Hybrid H_2: In this hybrid, we consider a simulator that only chooses $\widetilde{\mathbf{w}}$ to create a simulated garbled circuit. Then it samples only VSS shares received by Bob. Namely, it simulates t shares for the indexes received by Bob. Then it follows the algorithm g^B and extends the shares $\bar{\mathbf{t}}$ to set of shares $\bar{\mathbf{s}} = (s_1, \ldots, s_n)$ such that $\bar{\mathbf{s}}$ agrees with $\bar{\mathbf{t}}$ on set S and $\bar{\mathbf{s}}$ reconstructs to $\mathbf{dk}$ such that $\mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}}) = w$. The rest of the execution proceeds identically to H_1. Indistinguishability of H_2 and H_1 follows from the perfect security of the VSS scheme and the fact the distribution of $\mathbf{dk}$ is identically distributed in H_1 and H_2 conditioned on the event $\mathsf{Dec}(\mathbf{dk}, \widetilde{\mathbf{w}}) = w$.

Finally, we argue that Hybrid H_2 and the real simulation are identically distributed as the computation performed by g^B follows exactly the simulation in H_2. This completes the security proof. ∎

4 Additively Secure Protocols with Program Checkers

In this section, we show how to obtain security with one-bit leakage against malicious adversaries for a wide class of functionalities both in the two-party and the multi-party settings. On a high-level, we combine (1) an additively secure protocol for f, that is, a protocol that is secure against active adversaries up to additive attacks (cf. Definition 7) and, (2) a protocol for the verification algorithm g that is secure against malicious adversaries. We can rely on any malicious protocol for g.

To instantiate the additively secure protocol, we rely on a core lemma proving by Genkin et al. in [GIP+14], a work which introduced the notion of *additively secure circuits*. This lemma considers the malicious security of classic passively secure protocols, such as [GMW87, Bea91, BGW88], when executed in the presence of malicious adversaries. Informally speaking, Genkin et al. showed that for most classic honest-but-curious secure computation protocols for circuit evaluation, the effect of any active adversary corresponds precisely to an additive attack on the original circuit's wires where the additive attack is independent of the honest party's inputs. In particular, such protocols provide a mechanism to simulate adversaries where in addition to extracting an input from the adversary also extracts an additive attack to be supplied to the ideal functionality. Genkin et al. showed that slight variations of the passively secure protocols by Goldreich, Micali and Wigderson [GMW87] (Construction 5.8 [GIP+14]), Ben-Or, Goldwasser and Wigderson [BGW88] (Construction 5.5 [GIP+14]) and several others, are secure up to additive attacks.

Our contributions in this section are two fold:

1. In Section 4.2, we show that the distributed garbling protocol of Beaver et al. [BMR90] is in fact additively secure when the "offline" part of the protocol is executed using any additively secure protocol. This is the first constant-round protocol that has shown to be additively secure in the OT-hybrid. All previous works [GIP+14, GIP15, GIW16] considered protocols whose round complexity is proportional to the depth of the circuit or worked in the OLE-hybrid for large fields.
2. In Section 4.3, we provide a compiler that takes any additively secure protocol for f and combines it with a maliciously secure protocol for the "leaner" g to obtain a maliciously protocol for f that is secure against malicious adversaries up to 1-bit leakage. Roughly speaking the idea is that in an additively secure protocol, the adversary can only affect the computation in an input-independent and private manner. Therefore, a checking step can prevent an incorrect answer from being revealed.

4.1 Additive Attacks and AMD Circuits

In what follows we borrow the terminology and definitions verbatim from [GIP+14, GIW16]. We note that in this work we work with binary fields $\mathbb{F}_2$.

Definition 4 (AMD code [CDF+08]) *An (n,k,ε)-AMD code is a pair of circuits* $(\mathsf{Encode}, \mathsf{Decode})$ *where* $\mathsf{Encode} : \mathbb{F}^n \to \mathbb{F}^k$ *is randomized and* $\mathsf{Decode} : \mathbb{F}^k \to \mathbb{F}^{n+1}$ *is deterministic such that the following properties hold:*

– Perfect completeness. *For all* $\mathbf{x} \in \mathbb{F}^n$,

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(\mathbf{x})) = (0, \mathbf{x})] = 1.$$

– Additive robustness. *For any* $\mathbf{a} \in \mathbb{F}^k, \mathbf{a} \neq 0$, *and for any* $\mathbf{x} \in \mathbb{F}^n$ *it holds that*

$$\Pr[\mathsf{Decode}(\mathsf{Encode}(\mathbf{x}) + \mathbf{a}) \notin \mathsf{ERROR}] \leq \varepsilon.$$

Definition 5 (Additive attack) *An additive attack* $\mathbf{A}$ *on a circuit* C *is a fixed vector of field elements which is independent from the inputs and internal values of* C. $\mathbf{A}$ *contains an entry for every wire of* C, *and has the following effect on the evaluation of the circuit. For every wire* ω *connecting gates* a *and* b *in* C, *the entry of* $\mathbf{A}$ *that corresponds to* ω *is added to the output of* a, *and the computation of the gate* b *uses the derived value. Similarly, for every output gate* o, *the entry of* $\mathbf{A}$ *that corresponds to the wire in the output of* o *is added to the value of this output.*

Definition 6 (Additively corruptible version of a circuit) *Let* $\mathsf{C} : \mathbb{F}^{I_1} \times \ldots \times \mathbb{F}^{I_n} \to \mathbb{F}^{O_1} \times \ldots \times \mathbb{F}^{O_n}$ *be an n-party circuit containing* W *wires. We define the additively corruptible version of* C *to be the n-party functionality* $\tilde{f} : \mathbb{F}^{I_1} \times \ldots \times \mathbb{F}^{I_n} \times \mathbb{F}^W \to \mathbb{F}^{O_1} \times \ldots \times \mathbb{F}^{O_n}$ *that takes an additional input from the adversary which indicates an additive error for every wire of* C. *For all* $(\mathbf{x}, \mathbf{A})$, $\tilde{f}(\mathbf{x}, \mathbf{A})$ *outputs the result of the additively corrupted* C, *denoted by* $\mathsf{C}^{\mathbf{A}}$, *as specified by the additive attack* $\mathbf{A}$ *(*$\mathbf{A}$ *is the simulator's attack on* C*) when invoked on the inputs* $\mathbf{x}$.

Definition 7 (Additively secure implementation) *Let* $\varepsilon > 0$. *We say that a randomized circuit* $\widehat{\mathsf{C}} : \mathbb{F}^n \to \mathbb{F}^t \times \mathbb{F}^k$ *is an* ε*-additively-secure implementation of a function* $f : \mathbb{F}^n \to \mathbb{F}^k$ *if the following holds.*

– Completeness. *For every* $\mathbf{x} \in \mathbb{F}^n$, $\Pr[\widehat{\mathsf{C}}(\mathbf{x}) = f(\mathbf{x})] = 1$.
– Additive attack security. *For any additive attack* $\mathbf{A}$ *there exist* $a^{in} \in \mathbb{F}^n$, *and a distribution* $\mathbf{A}^{out}$ *over* $\mathbb{F}^k$, *such that for every* $\mathbf{x} \in \mathbb{F}^n$,

$$SD(\mathsf{C}^{\mathbf{A}}(\mathbf{x}), f(\mathbf{x} + a^{in}) + \mathbf{A}^{out}) \leq \varepsilon$$

where SD *denotes statistical distance between two distributions.*

Towards introducing our transformations, we conclude with definition of a *protocol compiler* to be a function Γ that takes as input the description of a functionality $\mathcal{F}$ and parameter param and gives a protocol specification $\Pi_{\mathcal{F}} = \Gamma(\mathsf{param}, \mathcal{F})$. Furthermore,

Definition 8 *Let* κ *be the security parameter. A protocol compiler* Γ *is said to be* secure up to additive attacks *if for any functionality* $\mathcal{F}$, $\Gamma(\mathcal{F}, \kappa)$ *realizes* $\widetilde{\mathcal{F}}$ *with security against active adversaries, where* $\widetilde{\mathcal{F}}$ *is defined to be the functionality that is identical to* $\mathcal{F}$ *with the exception that it additionally receives an additive attack* $\mathbf{A}$ *from the adversary, to be applied to each wire of the circuit.*

4.2 Additive Security of BMR Distributed Garbling

In this section we will prove that the BMR encoding is resilient to additive attacks. Recall that in the standard passive BMR protocol for the function f, the parties first jointly compute garbled tables in an offline phase for the circuit C that computes f. Then in an online phase, each party P_i reveals their masked input bits followed by the parties revealing their input key labels corresponding to the input bits. Upon receiving the values, the parties evaluate the garbled circuit and output the result of the evaluation. Now, we prove that if we replace the passive protocol to compute the garbled tables with an additively secure protocol then we obtain an additively secure protocol for the underlying function f. More formally, let $\pi_1 = \Gamma(\mathsf{C}_{\text{BMR}}, \kappa)$ be the additively secure protocol that computes the shares of the garbled tables in the distributed BMR garbling functionality C_{BMR} specified in Section 2.4 and let π_2 be the protocol obtained by replacing the offline phase of the passive BMR protocol with π_1. For example, one can instantiate this protocol with the GMW protocol. We prove the following theorem,

Theorem 9 *For an arbitrary n-party function f, let π_2 be as defined above. Then for any malicious adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that can simulate $\mathcal{A}$'s view in the $\tilde{f}_C(x_1, \ldots, x_n, \mathbf{A})$-hybrid where $\tilde{f}_C$ outputs the result of the additively corrupted C as specified by the additive attack $\mathbf{A}$.*

Before we proceed to the proof of Theorem 9, we illustrate an interesting application of our theorem. One of the main applications to additive resilient circuits was compiling secure computation protocols from passive to active security with low overhead. While the works [GIP+14, GIP15] resolve the question for arithmetic computations over large fields, the question remains open for Boolean computations. The work of Genkin et al. [GIW16] provides a passive to active compilation for Boolean circuits with polylogarithmic overhead. However, all the protocols that have been considered in previous work belong to the class of non-constant round protocols (protocols whose complexity depends on the depth of the circuit). We are the first to demonstrate this property for a constant-round protocol. Moreover, if optimal compilation of binary AMD circuits is achievable, then our result will imply communication optimal multi-party protocols for Boolean computations in constant-round. All previous works, incur at least an $\Omega(s/log|C|)$ overhead of compiling passive to active in the OT-hybrid in the multi-party setting.[8]

We next provide a high-level overview of the additive security against malicious adversaries of the BMR protocol in the (non-leaky) full security setting. Consider the BMR distributed garbling functionality C_{BMR} that outputs shares of the garbled tables. We need to translate an additive attack to the offline functionality to a corresponding attack on the wires of the original circuit C. Towards this, we first recall how a garbled row in the distributed garbling of BMR looks like. Recall that each party P_i contributes a pair of keys $(k^i_{w,0}, k^i_{w,1})$ for every wire w and mask λ^i_w. The combined mask (or color bit) of a gate is defined as $\lambda_w = \oplus_i \lambda^i_w$. Then the $(r_1, r_2)^{th}$ row for $r_1, r_2 \in \{0,1\}$ of the garbled gate g can be expressed as follows:

[8] In the two-party setting, the work of [HIV17] provides a constant overhead passive to active compiler for garbled circuits.

$$R^{g,j}_{r_1 r_2} = \underbrace{\bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k^i_{a,r_1}}(g,j,r_1,r_2) \oplus \mathsf{PRF}_{k^i_{b,r_2}}(g,j,r_1,r_2) \right)}_{\text{Ciphertext Padding}} \underbrace{\oplus k^j_{c,0} \oplus \Big(\underbrace{\chi_{r_1,r_2}}_{\text{Perm. Bit}} \cdot \underbrace{(k^j_{c,1} \oplus k^j_{c,0})}_{\text{Wire's } \Delta} \Big)}_{\text{Plaintext}}$$

where $\chi_{r_1,r_2} = ((\lambda_a \oplus r_1) \cdot (\lambda_b \oplus r_2)) \oplus \lambda_c$.

Next, we analyze an additive attack on the protocol computing the distributed garbling. We break this into two main parts: additive errors on the PRF values and additive errors on the plaintext part (containing the target keys). It was already shown in prior work [HSS17] that additive errors on the PRF values cannot affect the correctness of the computation if the plaintext is computed correctly. On a high-level, this is because for the computation at a gate to change, the adversary will have to guess the difference of the two keys of the honest party for the output wire. We next analyze an additive attack on the plaintext. The formula for computing the plaintext is:

$$\begin{aligned} &\left(\chi_{r_1,r_2} \cdot (k^j_{c,1} \oplus k^j_{c,0})\right) \\ &= \Big[\underbrace{\Big(\bigoplus_{j\in[n]} \lambda^j_a \oplus r_1 \Big)}_{\oplus\, e_1} \cdot \underbrace{\Big(\bigoplus_{j\in[n]} \lambda^j_b \oplus r_2 \Big)}_{\oplus\, e_2} \oplus \underbrace{\Big(\bigoplus_{j\in[n]} \lambda^j_c \Big)}_{\oplus\, e_3} \Big] \cdot \underbrace{\left(k^j_{c,1} \oplus k^j_{c,0}\right)}_{\oplus\, e_4}. \end{aligned}$$

The high-level goal here is that given an additive attack $\mathbf{A}^g_{\mathrm{BMR}}$ on the garbling of gate g, we need to extract a corresponding additive attack on the wires of the original computed circuit C. We can define additive errors e_1, e_2, e_3 and e_4 and express any generic additive attack on this part as follows:

$$\begin{aligned} \Big[\Big(\bigoplus_{j\in[n]} \lambda^j_a \oplus r_1 \oplus \mathrm{e}_1 \Big) \cdot \Big(\bigoplus_{j\in[n]} \lambda^j_b \oplus r_2 \oplus \mathrm{e}_2 \Big) \oplus \Big(\bigoplus_{j\in[n]} \lambda^j_c \oplus \mathrm{e}_3 \Big) \Big] \\ \cdot \left(k^j_{c,1} \oplus k^j_{c,0} \oplus \mathrm{e}_4\right) \end{aligned}$$

To argue that the additive error e_4 cannot render an incorrect computation, we observe that, this error can, at best, mess with the key being encrypted, but cannot change the value unless the adversary can guess the key chosen by the honest party. Therefore, this will not cause any additive error in the computation of the circuit wires. The remaining errors seem to correspond directly to corresponding wires of the circuit. While this is the basic intuition, formally arguing that given any additive attack on the computation of the distributed garbling, extracting a corresponding attack on the actual circuit (being garbled) turns out to be subtle and technical. We now proceed to the formal proof of security.

Proof: We will start with a protocol that is secure up to additive attacks for realizing the distributed garbling functionality. For example, we can rely on the passive [GMW87] protocol instantiated with a malicious OT protocol. Therefore, given any active adversary that attacks this protocol, we can consider an equivalent adversary in the $\mathsf{C}_{\mathrm{BMR}}^{\mathbf{A}}$-hybrid that provides its inputs and an attack vector for the distributed garbling functionality.

Description of the simulator. The simulator $\mathcal{S}$ begins a real execution with adversary $\mathcal{A}$. First, it extracts the adversary's inputs and an additive attack vector $\mathbf{A}_{\mathrm{BMR}}$ for functionality $\mathsf{C}_{\mathrm{BMR}}$. Next, the simulator determines the active path. The inactive rows for the garbled tables will then be replaced with random values. To determine the active path, it defines Λ_w values for all wires. For the input wires carrying honest party inputs, it will choose them at random. For the internal wires, it will proceed in a topological ordering of the gates, determining the Λ_w values for the output of the gates along this ordering. Towards this, it picks an honest party P_j. Let g be a gate in this ordering. Inductively, on the topological ordering, we will ensure that a Λ_w value has already been chosen for the output wires of gates $1, \ldots, (g-1)$. Let a and b denote the input wires of gate g. Then the simulator proceeds as follows. From the attack vector $\mathbf{A}_{\mathrm{BMR}}$, the simulator identifies the additive errors $e_1, e_2, e_3^j, e_4^j, e_5^j$ such that row number (Λ_a, Λ_b) in the garbled table for g can be written as:

$$\bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k_{a,r_1}^i}(g, j, r_1, r_2) \oplus \mathsf{PRF}_{k_{b,r_2}^i}(g, j, \Lambda_a, \Lambda_b) \right) \oplus \left[\left(\bigoplus_{j\in[n]} \lambda_a^j \oplus \Lambda_a \oplus \mathbf{e}_1 \right) \cdot \left(\bigoplus_{j\in[n]} \lambda_b^j \oplus \Lambda_b \oplus \mathbf{e}_2 \right) \oplus \left(\bigoplus_{j\in[n]} \lambda_c^j \oplus \mathbf{e}_3^j \right) \right] \cdot \left(k_{c,1}^j \oplus k_{c,0}^j \oplus \mathbf{e}_4^j \right) \oplus k_{c,0}^j \oplus \mathbf{e}_5^j$$

Next, it defines

$$\Lambda_c = \left(\bigoplus_{j\in[n]} \lambda_a^j \oplus \Lambda_a \oplus \mathbf{e}_1 \right) \cdot \left(\bigoplus_{j\in[n]} \lambda_b^j \oplus \Lambda_b \oplus \mathbf{e}_2 \right) \oplus \left(\bigoplus_{j\in[n]} \lambda_c^j \oplus \mathbf{e}_3^j \right)$$

Recall that the simulator needs to extract an additive attack vector $\mathbf{A}_C$ on the underlying circuit. The simulator includes the additive errors e_1, e_2, e_3^j respectively to the wires a, b and c in this vector.

Note that the set of Λ_w values for all wires specifies the active rows for all gates. Namely, the row (Λ_a, Λ_b) is the active row for gate g with input wires a, b. In simulating the inactive rows, the simulator sets the honest party's shares to be uniformly random. For the active rows, the simulator first picks one key k_c^j for the honest party P_j and sets the active row as follows:

- $\bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k_a^i}(g, j, \Lambda_a, \Lambda_b) \oplus \mathsf{PRF}_{k_b^i}(g, j, \Lambda_a, \Lambda_b) \right) \oplus k_c^j \oplus \mathbf{e}_5^j$ if $\Lambda_c = 0$.
- $\bigoplus_{i=1}^{n} \left(\mathsf{PRF}_{k_a^i}(g, j, \Lambda_a, \Lambda_b) \oplus \mathsf{PRF}_{k_b^i}(g, j, \Lambda_a, \Lambda_b) \right) \oplus k_c^j \oplus \mathbf{e}_4 \oplus \mathbf{e}_5^j$ else.

Based on this garbled table, the shares are revealed for the honest party. Finally, for the output translation table the simulator submits the attack vector $\mathbf{A}_C$ to the ideal functionality and receives the output. The simulator fixes λ_w^j so that $\Lambda_w \oplus (\oplus_{j=1}^n \lambda_w^j)$ is equal to the output received from the functionality.

The complete proof is provided in the full version.

■

4.3 Compiling Additively Secure Protocols

We now present a compiler that takes an additively secure protocol for $\mathcal{F}$ and a malicious protocol for the leaner verifier functionality and produces a protocol that is secure against active adversaries with 1-bit leakage for functionalities that have efficient verifiability.

Theorem 10 *Let Γ_1 be a protocol compiler that is secure up to additive attacks against static, active adversaries, Γ_2 a protocol compiler that is fully secure against corruption by static, active adversaries. Then, there exists a protocol compiler Γ to securely compute with abort a deterministic functionality $\mathcal{F} : \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^m$ verifiable with predicate $\mathcal{G}$ against static, active adversaries up to 1-bit leakage for the same corruption model. Furthermore, the computational and communication complexity of $\Gamma(\mathcal{F},\kappa)$ is proportional to sum of the respective measures of $\Gamma_1(\mathcal{F}^*,\kappa)$ and $\Gamma_2(\mathcal{G}^*,\kappa)$ where $|\mathcal{F}^*| = O(|\mathcal{F}|) + \mathsf{poly}(n,m,\kappa)$ and $|\mathcal{G}^*| = O(|\mathcal{G}|) + \mathsf{poly}(m,n,\kappa)$.*

Proof: Unlike our protocol for garbled circuits from Section 3, where a protocol with one-sided security suffices, here we require a fully-secure protocol for $\mathcal{G}$. Nonetheless, as we show in Section 5 that such a protocol can still lead to efficiency improvements over other techniques. The functionality $\mathcal{F}^*$ is a slight variant of $\mathcal{F}$, which outputs additive shares of the output to the parties instead of the output itself. Similarly, the functionality $\mathcal{G}^*$ is a slight variant of $\mathcal{G}$, that takes as input the additive shares of the output and applies the function on the reconstructed value. Our protocol compiler proceeds in the following steps given a security parameter κ and n-party functionality $\mathcal{F}$ that takes n inputs $x_1,\ldots,x_n$ and gives shares of the m-bit to all parties. Consider an arbitrary functionality $\mathcal{F}$ verifiable with predicate $\mathcal{G}$.

- Construct functionality $\mathcal{F}^*$ that takes input x_i from P_i ($i \in [n]$) and outputs $(s_1,\ldots,s_n)$ where party P_i receives output s_i such that $\sum_i s_i = f(x_1,\ldots,x_n)$.
- Let $\mathcal{G}^*$ be the function that takes as input (x_i,s_i) from party P_i ($i \in [n]$) and computes $s = \sum_i s_i$ and b=$\mathcal{G}(x_1,\ldots,x_n,s)$. Finally, It outputs s if and only if $b = 1$.

The protocol now proceeds as follows:

1. In the first step, the parties execute protocol $\Pi_1 = \Gamma_1(\mathcal{F}^*,\kappa)$ where P_i uses input x_i and receives s_i as the output.
2. The parties next engage in the protocol $\Pi_2 = \Gamma_2(\mathcal{G}^*,\kappa)$ where Alice uses (x_i,s_i) as its input. Their final output is their output from protocol Π_2.

We show that this protocol achieves security against active adversaries with 1-bit leakage. For simplicity, we consider a hybrid protocol Π^* in the $(\widetilde{\mathcal{F}}^*, \mathcal{G}^*)$-hybrid where $\widetilde{\mathcal{F}}^*$ is the functionality that besides the inputs for $\mathcal{F}^*$ also gets an additive attack $\mathbf{A}$ from the adversary. Note that we only need to rely on a sequential composition, which holds even in the simple stand-alone setting. The protocol proceeds in two steps. Honest parties provide inputs to $\widetilde{\mathcal{F}}^*$ as specified in Step 1 of the above protocol, receive their answer, and following that send their inputs to $\mathcal{G}^*$ as specified in Step 2 and receive their answers. We construct a simulator $\mathcal{S}$ for an arbitrary adversary $\mathcal{A}$ in this modified protocol Π^*. We remark that to consider the protocol in this hybrid, we crucially rely on the fact that both protocols admit standard security in the presence of active adversaries. In particular, both protocols provide a mechanism to extract the corrupted parties' inputs (and possibly other auxiliary information).

Let $\mathcal{A}$ be an adversary that corrupts the set of parties $\mathcal{I}$. The simulator begins an execution and receives from the adversary the inputs set $\{x_i\}_{i\in\mathcal{I}}$ as well as an additive attack vector $\mathbf{A}$. It provides as output random values $\{s_i\}_{i\in\mathcal{I}}$. Next it receives as input $\{(x_i^*, s_i^*)\}_{i\in\mathcal{I}}$ and computes the following leakage predicate described in Figure 2.

FUNCTION $g^B(x)$

Parameters: $\mathbf{A}, \{x_i, s_i, x_i^*, s_i^*\}_{i\in\mathcal{I}}$

Input: $\{x_i\}_{i\in[n]/\mathcal{I}}$.

Output: The functionality proceeds as follows:

- Let $\hat{y} = \widetilde{\mathcal{F}}^*(x_1, \ldots, x_n, \mathbf{A})$.
- Output $\mathcal{G}(\hat{x}_1, \ldots, \hat{x}_n, \hat{y} + \sum_{i\in\mathcal{I}}(s_i^* - s_i))$ where $\hat{x}_i = x_i^*$ for $i \in \mathcal{I}$ and $\hat{x}_i = x_i$ for $i \in [n]/\mathcal{I}$.

Fig. 2. Leakage function for $\mathcal{A}$.

The simulator submits $\{x_i^*\}_{i\in\mathcal{I}}$ and g^B to the ideal functionality and receives y which it feeds internally to $\mathcal{A}$. Recall that the functionality returns an answer if and only if the leakage predicate returns 1. We remark that even if $\mathcal{G}$ can be realized with a protocol that has guaranteed output delivery, the resulting protocol will only achieve security with abort as the adversary can make the computation fail by making the first part of the protocol output an incorrect answer.

Proof of correctness. As the leakage function simulates what happens in the real protocol and the additive sharing of the output information theoretically hides the output, our simulation is perfect. We only need to argue that if an output is received by an honest party then it corresponds to the right output. In other words, we need to argue that $\hat{y} = y$. This follows directly from the definition of efficiently verifiable functions and the fact that $\mathcal{F}$ is deterministic. ∎

5 Perfect Matching Protocol Secure up to Additive Attacks

A perfect matching of a graph is a matching in which every vertex of the graph is incident to exactly one edge of the matching. Algorithms for perfect matchings are well-studied. For instance, the classic Ford-Fulkerson algorithm for maxflow can be used to find a matching in $O(VE)$ for the specific case of bipartite graphs. However, when this algorithm is transformed into a secure protocol, each execution of breadth-first requires the use of special ORAM data structures.

In the general case, finding a perfect matching in G reduces to solving a system of V linear equations in E variables. Applying such general methods from prior work to construct a secure computation protocol results in a communication complexity of at least $O(VE)$ which for dense graphs could be $O(V^3)$. An alternative approach would be to construct a protocol from one of the many standard algorithms for solving perfect matching. The textbook algorithm for perfect matching due to Edmond runs in time $O(V^2E)$ and the Micali-Vazirani algorithm requires $O(E\sqrt{V})$. However, both these algorithms require complicated input-dependent memory accesses which when translated to a secure protocol incurs heavy communication and computational overhead. Rabin and Vazirani [RV89] gave a randomized algorithm with runtime $O(V^{\omega+1})$ where ω is the matrix multiplication exponent. Mucha and Sankowski [MS04] improve this approach to $O(V^\omega)$. In contrast, the algorithm that we present below in the matrix multiplication hybrid model runs in *local* time $O(V^\omega)$ where ω is the (best) matrix multiplication exponent and requires communication $O(V^2 \log V)$.

Our starting point is the work of Harvey [Har06] who showed an $O(V^\omega)$ algorithm to compute the perfect matching. Our first insight is that an oblivious algorithm can be extracted from this work and adapted to a secure computation protocol in a hybrid model where the parties have access to a matrix-multiplication and matrix-inverse functionalities that work on shared inputs. While Harvey's algorithm runs in time $O(V^\omega)$, our communication complexity is better because a secure computation protocol for matrix multiplication requires $O(n^2)$ communication using (additively) homomorphic encryption while locally computing it requires $O(n^\omega)$ time. Next, we show that by instantiating the above functionalities using a maliciously secure protocol, the passive version of the protocol is secure against active adversaries up to additive attacks, analogous to [GMW87] additive security in the OT-hybrid from [GIP+14].

Finally, to obtain a protocol with 1-bit leakage, we note that it is easy to verify a perfect matching. It suffices to ensure that each vertex appears at most once in the matching, the size of the matching is $V/2$ and the edges were present in E. In fact, it can be done in time $O(V+E)$ but it suffices for our application that the verification be done in $O(V^2)$. We can achieve this obliviously by scanning element by element in the adjacency matrix of the graph and verifying the above conditions. We conclude with the following theorem proven in the full version.

Theorem 11 *For a graph $G = (V, E)$, there exists a data-oblivious algorithm that verifies that a putative matching M for G is perfect in time $O(V^2)$.*

Acknowledgements. We thank the anonymous PKC 2020 reviewers for their valuable feedback. The first author is supported by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in

the Prime Minister's Office, and by ISF grant 1316/18. The second author is supported by NSF Awards 1664445 and 1646671. The third author is supported by Google Faculty Research Grant and NSF Award CNS-1618884. The views expressed are those of the authors and do not reflect the official policy or position of Google, the Department of Defense, the National Science Foundation, or the U.S. Government.

References

ADI+17. Benny Applebaum, Ivan Damgård, Yuval Ishai, Michael Nielsen, and Lior Zichron. Secure arithmetic computation with constant computational overhead. In *CRYPTO*, pages 223–254, 2017.

Bea91. Donald Beaver. Efficient multiparty protocols using circuit randomization. In *CRYPTO*, pages 420–432, 1991.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *STOC*, pages 1–10, 1988.

BMR90. Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513, 1990.

BNP08. Assaf Ben-David, Noam Nisan, and Benny Pinkas. Fairplaymp: a system for secure multi-party computation. In *CCS*, pages 257–266, 2008.

cal. http://www.calctopia.com.

CDF+08. Ronald Cramer, Yevgeniy Dodis, Serge Fehr, Carles Padró, and Daniel Wichs. Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors. In *EUROCRYPT*, pages 471–488, 2008.

CGH+18. Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *CRYPTO*, pages 34–64, 2018.

CGMA85. Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *FOCS*, pages 383–395, 1985.

FLOP18. Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *CRYPTO*, pages 331–361, 2018.

Fre77. Rusins Freivalds. Probabilistic machines can use less running time. In *IFIP Congress*, pages 839–842, 1977.

GIKR01. Rosario Gennaro, Yuval Ishai, Eyal Kushilevitz, and Tal Rabin. The round complexity of verifiable secret sharing and secure multicast. In *STOC*, pages 580–589, 2001.

GIP+14. Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *STOC*, pages 495–504, 2014.

GIP15. Daniel Genkin, Yuval Ishai, and Antigoni Polychroniadou. Efficient multi-party computation: From passive to active security via secure SIMD circuits. In *CRYPTO*, pages 721–741, 2015.

GIW16. Daniel Genkin, Yuval Ishai, and Mor Weiss. Binary AMD circuits from secure multiparty computation. In *TCC*, pages 336–366, 2016.

GMW87. Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

Gol04. Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

Har06. Nicholas J. A. Harvey. Algebraic structures and algorithms for matching and matroid problems. In *FOCS*, pages 531–542, 2006.

HEKM11. Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX*, 2011.

HIV17. Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Actively secure garbled circuits with constant communication overhead in the plain model. In *TCC*, pages 3–39, 2017.

HKE12. Yan Huang, Jonathan Katz, and David Evans. Quid-pro-quo-tocols: Strengthening semi-honest protocols with dual execution. In *IEEE Symposium on Security and Privacy*, pages 272–284, 2012.

HSS17. Carmit Hazay, Peter Scholl, and Eduardo Soria-Vazquez. Low cost constant round MPC combining BMR and oblivious transfer. In *ASIACRYPT*, pages 598–628, 2017.

IPS08. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In *CRYPTO*, pages 572–591, 2008.

IPS09. Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *TCC*, pages 294–314, 2009.

Kin95. Valerie King. A simpler minimum spanning tree verification algorithm. In *Algorithms and Data Structures*, pages 440–448, 1995.

KPR18. Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT*, pages 158–189, 2018.

KRRW18. Jonathan Katz, Samuel Ranellucci, Mike Rosulek, and Xiao Wang. Optimizing authenticated garbling for faster secure two-party computation. In *CRYPTO*, pages 365–391, 2018.

KS14. Marcel Keller and Peter Scholl. Efficient, oblivious data structures for MPC. In *ASIACRYPT*, pages 506–525, 2014.

KSMB13. Benjamin Kreuter, Abhi Shelat, Benjamin Mood, and Kevin R. B. Butler. PCF: A portable circuit format for scalable two-party secure computation. In *USENIX*, pages 321–336, 2013.

KSS12. Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX*, pages 285–300, 2012.

LHS+14. Chang Liu, Yan Huang, Elaine Shi, Jonathan Katz, and Michael W. Hicks. Automating efficient ram-model secure computation. In *IEEE Symposium on Security and Privacy*, pages 623–638, 2014.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of Yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

LP12. Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *J. Cryptology*, 25(4):680–722, 2012.

LPSY15. Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015.

MF06. Payman Mohassel and Matthew K. Franklin. Efficiency tradeoffs for malicious two-party computation. In *PKC*, pages 458–473, 2006.

MNPS04. Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay - secure two-party computation system. In *USENIX*, pages 287–302, 2004.

MOR03. Philip D. MacKenzie, Alina Oprea, and Michael K. Reiter. Automatic generation of two-party computations. In *CCS*, pages 210–219, 2003.

MR13. Payman Mohassel and Ben Riva. Garbled circuits checking garbled circuits: More efficient and secure two-party computation. In *CRYPTO*, pages 36–53, 2013.

MS04. Marcin Mucha and Piotr Sankowski. Maximum matchings via gaussian elimination. In *FOCS*, pages 248–255, 2004.

MZ17. Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *IEEE SP'17*, 2017.

NNOB12. Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, and Sai Sheshank Burra. A new approach to practical active-secure two-party computation. In *CRYPTO*, pages 681–700, 2012.

NO09. Jesper Buus Nielsen and Claudio Orlandi. LEGO for two-party secure computation. In *TCC*, pages 368–386, 2009.

RHH14. Aseem Rastogi, Matthew A. Hammer, and Michael Hicks. Wysteria: A programming language for generic, mixed-mode multiparty computations. In *IEEE Symposium on Security and Privacy*, pages 655–670, 2014.

RV89. Michael O. Rabin and Vijay V. Vazirani. Maximum matchings in general graphs through randomization. *J. Algorithms*, 10(4):557–567, 1989.

WRK17a. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Authenticated garbling and efficient maliciously secure two-party computation. In *CCS*, pages 21–37, 2017.

WRK17b. Xiao Wang, Samuel Ranellucci, and Jonathan Katz. Global-scale secure multiparty computation. In *CCS*, pages 39–56, 2017.

Yao86. Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.